

Digital Principles and System Design

P. S. MANOHARAN

Thiagarajar College of Engineering
Madurai - 625 015.

CHARULATHA PUBLICATIONS

New No.24, Thambiah Road,
West Mambalam, Chennai - 600 033.
Phone : 24745589, 24746546
Email: charulathapublications@yahoo.com
URL : www.charulathapublication.com

Revised Edition - June 2019

© Charulatha Publications

Price : 325/-

Copies can be had from

CHARULATHA PUBLICATIONS

New No.24, Thambiah Road,
West Mambalam, Chennai - 600 033.
Phone : 24745589, 24746546
Email: charulathapublications@yahoo.com
URL : www.charulathapublication.com

PREFACE

The present fourth edition is an outcome of the various suggestions sent by many Students and Professors. The author feels highly indebted to them for their useful and valuable suggestions.

This book is intended to be used as a text book that provides comprehensive coverage of all the topics relevant to the subject of "**Digital Principles and System Design**" for the students of CSE and IT.

Part A and Part B questions are given for each unit collected from many Anna University question papers.

Some printing errors in the previous editions have been removed and in case, still some errors might be there, the author will be indeed grateful if such ambiguities are brought to his notice through e-mail at *psmeee@tce.edu*.

I would like to express my thanks to my teachers, and management of Thiagarajar college of Engineering, Madurai and M/S, Charulatha Publications, Chennai.

P . S . MANOHARAN

SYLLABUS

CS8351	DIGITAL PRINCIPLES AND SYSTEM DESIGN	L T P C
		4 0 0 4

Objectives:

- To design digital circuits using simplified Boolean functions
- To analyze and design combinational circuits
- To analyze and design synchronous and asynchronous sequential circuits
- To understand Programmable Logic Devices
- To write HDL code for combinational and sequential circuits

UNIT – I: BOOLEAN ALGEBRA AND LOGIC GATES **12**

Number Systems - Arithmetic Operations - Binary Codes- Boolean Algebra and Logic Gates - Theorems and Properties of Boolean Algebra - Boolean Functions - Canonical and Standard Forms - Simplification of Boolean Functions using Karnaugh Map - Logic Gates – NAND and NOR Implementations.

UNIT – II: COMBINATIONAL LOGIC **12**

Combinational Circuits – Analysis and Design Procedures - Binary Adder-Subtractor - Decimal Adder - Binary Multiplier - Magnitude Comparator - Decoders – Encoders – Multiplexers - Introduction to HDL – HDL Models of Combinational circuits.

UNIT – III: SYNCHRONOUS SEQUENTIAL LOGIC **12**

Sequential Circuits - Storage Elements: Latches , Flip-Flops - Analysis of Clocked Sequential Circuits - State Reduction and Assignment - Design Procedure - Registers and Counters - HDL Models of Sequential Circuits.

UNIT – IV: ASYNCHRONOUS SEQUENTIAL LOGIC **12**

Analysis and Design of Asynchronous Sequential Circuits – Reduction of State and Flow Tables – Race-free State Assignment – Hazards.

UNIT – V: MEMORY AND PROGRAMMABLE LOGIC **12**

RAM – Memory Decoding – Error Detection and Correction - ROM - Programmable Logic Array – Programmable Array Logic – Sequential Programmable Devices.

TOTAL : 60 PERIODS

OUTCOMES:

On Completion of the course, the students should be able to:

- Simplify Boolean functions using KMap
- Design and Analyze Combinational and Sequential Circuits
- Implement designs using Programmable Logic Devices
- Write HDL code for combinational and Sequential Circuits

TEXT BOOK:

- 1) M. Morris R. Mano, Michael D. Ciletti, —Digital Design: With an Introduction to the Verilog HDL, VHDL, and SystemVerilog , 6th Edition, Pearson Education, 2017.

REFERENCES:

- 1) G. K. Kharate, Digital Electronics, Oxford University Press, 2010
- 2) John F. Wakerly, Digital Design Principles and Practices, Fifth Edition, Pearson Education, 2017.
- 3) Charles H. Roth Jr, Larry L. Kinney, Fundamentals of Logic Design, Sixth Edition, CENGAGE Learning, 2013
- 4) Donald D. Givone, Digital Principles and Design , Tata Mc Graw Hill, 2003

CONTENTS

UNIT I – BOOLEAN ALGEBRA AND LOGIC GATES

1.1	Introduction	1.1
1.2	Decimal Number System	1.1
1.3	Binary Number System	1.1
1.4	Octal Number System	1.2
1.5	Hexadecimal Number System	1.2
1.6	Number Base Conversions	1.3
1.7	Complements	1.12
1.8	Signed Binary Numbers	1.20
1.9	Binary Arithmetic	1.22
1.10	Other Number Systems	1.28
1.11	Binary Codes	1.28
1.12	Error Detection Codes	1.33
1.13	Alphanumeric code	1.35
1.14	Boolean Postulates and Laws	1.37
1.15	De Morgan's Theorems	1.44
1.16	Boolean Rules for Simplification	1.46
1.17	Minimization of Boolean Expressions	1.50
1.18	Duality	1.56
1.19	Boolean Functions	1.58
1.20	Product of Sums (POS) Method	1.59
1.21	Sum of Products (SOP) Method	1.61
1.22	Minterms	1.63
1.23	Maxterms	1.64
1.24	Canonical form	1.65
1.25	Conversion between canonical forms	1.71
1.26	Karnaugh Maps	1.72
1.27	Tabulation Method (Quine-McCluskey Method)	1.98
1.28	Basic Logic Gates	1.114
1.29	Other Logic Gates	1.115
1.30	Gate Conversions	1.118
1.31	Digital ICS	1.120

1.32	Universal Gates	1.121
1.33	Implementation of Logic Function	1.122
1.34	NAND and NOR Implementation	1.128
1.35	Multilevel Gate	1.134
1.36	Multioutput Gate	1.136
1.37	Logic Families	1.137
1.38	Characteristic of Digital ICs	1.138
	Anna University Questions	1.141

UNIT II – COMBINATIONAL LOGIC

2.1	Introduction	2.1
2.2	Design Procedure	2.1
2.3	Half Adder	2.4
2.4	Full Adder	2.5
2.5	Half Subtractor	2.8
2.6	Full Subtractor	2.9
2.7	Parallel Binary Adders	2.12
2.8	Parallel Subtractor	2.13
2.9	Binary Adder/Subtractor	2.14
2.10	Serial Adder	2.15
2.11	Serial Subtractor	2.17
2.12	Serial Adder/Subtractor	2.17
2.13	Binary Multiplier	2.18
2.14	Carry Look Ahead Adder	2.20
2.15	BCD Adder	2.23
2.16	Magnitude Comparator	2.26
2.17	Parity Generator and Checker	2.29
2.18	Code Converters	2.31
2.19	Decoders	2.39
2.20	Encoders	2.43
2.21	Multiplexers	2.47
2.22	Demultiplexers	2.51
2.23	Implementations of Combinational Logic	2.53
2.24	Introduction to Hardware Description Language	2.59
2.25	HDL Based Design Flow	2.61

2.26	VHDL Building Blocks	2.63
2.27	Library	2.72
2.28	Types and Constants	2.72
2.29	Predefined Types	2.73
2.30	User-Defined Types	2.74
2.31	Sequential Statements	2.75
2.32	Challenges in HDL	2.78
2.33	HDL for Combinational Circuits	2.79
	Anna University Questions	2.89

UNIT III – SYNCHRONOUS SEQUENTIAL LOGIC

3.1	Introduction	3.1
3.2	Classification of Logic Circuits	3.2
3.3	Flip-flops	3.2
3.4	SR Fliflop	3.3
3.5	Clocked SR Flipflop	3.5
3.6	Triggering of flipflops	3.6
3.7	D Flipflop	3.8
3.8	JK Flipflop	3.9
3.9	T Flipflop	3.11
3.10	Master Slave JK Flipflop	3.11
3.11	Excitation Table	3.12
3.12	Realization of One Flipflop using other Flipflops	3.15
3.13	Classification of synchronous sequential circuits	3.19
3.14	State Equation	3.20
3.15	State Table	3.20
3.16	State Diagram	3.20
3.17	Analysis of Synchronous Sequential Circuits	3.21
3.18	State Minimization	3.31
3.19	State Assignment	3.34
3.20	Design of Synchronous Sequential Circuits	3.35
3.21	ASM Chart	3.43
3.22	Shift Registers	3.46
3.23	Counters	3.54
3.24	Synchronous Counters	3.58

3.25 Modulus-N Counter	3.64
3.26 Shift Register Counters	3.65
3.27 Design of Counters	3.68
3.28 Up/Down Ripple Counter	3.89
3.29 HDL for Sequential Logic Circuits	3.91
Anna University Questions	3.100

UNIT IV – ASYNCHRONOUS SEQUENTIAL CIRCUITS

4.1 Introduction	4.1
4.2 Types of Asynchronous Sequential Circuits	4.1
4.3 Transition Table	4.2
4.4 Flow Table	4.2
4.5 Primitive Flow Table	4.3
4.6 Analysis of Fundamental Mode Circuits	4.3
4.7 Analysis of Pulse Mode Circuits	4.7
4.8 Races	4.12
4.9 Cycles	4.14
4.10 Race Free State Assignment	4.14
4.11 Minimization of Primitive Flow Table	4.18
4.12 Design of Fundamental Mode Asynchronous Circuits	4.22
4.13 Design of Pulse Mode Asynchronous Circuits	4.29
4.14 Hazards	4.36
4.15 Design of Hazard Free Circuits	4.37
Anna University Questions	4.44

UNIT V – MEMORY & PROGRAMMABLE LOGIC

5.1 Introduction	5.1
5.2 Memory Unit	5.1
5.3 Write Operation	5.2
5.4 Read Operation	5.3
5.5 Classification of Memories	5.3
5.6 RAM Organization	5.5
5.7 Static RAM Cell	5.6
5.8 Bipolar RAM Cell	5.7
5.9 MOSFET RAM Cell	5.8
5.10 Dynamic RAM Cell	5.9

5.11 ROM	5.12
5.12 ROM Cell	5.12
5.13 ROM Organization	5.14
5.14 PROM	5.15
5.15 EPROM	5.17
5.16 UV EPROM	5.18
5.17 EEPROM	5.18
5.18 Memory Cycles	5.19
5.19 Memory Decoding	5.20
5.20 Memory Expansion	5.22
5.21 Advantages of RAM	5.25
5.22 Advantages of ROM	5.25
5.23 Disadvantages of ROM	5.25
5.24 Comparison Between RAM/ROM	5.26
5.25 Comparison Between SRAM/DRAM	5.26
5.26 Comparison of Types of Memories	5.26
5.27 Implementation of Combinational Logic Circuits	5.27
5.28 Programmable Logic Devices	5.29
5.29 Classification of PLDS	5.30
5.30 Programmable ROM	5.32
5.31 Programmable Logic Array	5.34
5.32 Implementation of Combinational Using PLA	5.36
5.33 Programmable Array Logic	5.46
5.34 Implementation of Combinational	5.47
5.35 Field Programmable Gate Arrays	5.52
5.36 Comparison Between PROM, PLA And PAL	5.54
5.37 Error Detection and Correction	5.55
5.38 Sequential programmable devices	5.59
5.39 Application Specific Integrated Circuits	5.65
Anna University Questions	5.68

Appendix	1 – 8
References	(i)
Index	(i) – (iii)

INDEX

A

Advantages of RAM	3.42
Advantages of ROM	3.42
Alphanumeric code	1.38
Analysis of Fundamental Mode Circuits	5.3
Analysis of Pulse Mode Circuits	5.7
Analysis of Synchronous Sequential Circuits	4.21
ASM Chart	4.43

B

Basic Logic Gates	1.101
BCD Adder	2.20
Binary Adder/Subtractor	2.14
Binary Arithmetic	1.22
Binary Codes	1.28
Binary Number System	1.1
Bipolar Ram Cell	3.24
Boolean Functions	1.60
Boolean Postulates and Laws	1.40
Boolean Rules for Simplification	1.49

C

Canonical form	1.67
Carry Look Ahead Adder	2.18
Challenges in HDL	2.55
Characteristic of Digital	1.123
Classification of Logic Circuits	4.2
Classification of Memories	3.20
Classification of PLDS	3.47
Classification of synchronous sequential circuits	4.19
Clocked SR Flipflop	4.5
Code Converters	2.28
Comparison between PROM, PLA and PAL	3.69
Comparison Between RAM/ROM	3.43
Comparison Between SRAM/DRAM	3.43

Comparison of Types of Memories	3.43
Complements	1.12
Conversion between canonical forms	1.73
Counters	4.54
Cycles	5.14

D

D Flipflop	4.8
De Morgan's Theorems	1.47
Decimal Number System	1.1
Decoders	3.1
Demultiplexers	3.13
Design of Counters	4.68
Design of Fundamental Mode Asynchronous Circuits	5.22
Design of Hazard Free Circuits	5.37
Design of Pulse Mode Asynchronous Circuits	5.29
Design of Synchronous Sequential Circuits	4.35
Design Procedure	2.1
DIGITAL ICS	1.107
Disadvantages of ROM	3.42
Duality	1.58
Dynamic Ram Cell	3.26

E

EEPROM	3.35
Encoders	3.5
EPROM	3.34
Error Detection Codes	1.33
Excitation Table	4.12

F

Field Programmable Gate Arrays	3.67
Flip-flops	4.2
Flow Table	5.2
Full Adder	2.5
Full Subtractor	2.9

G		Memory Expansion	3.39
Gate Conversions	1.105	Memory Unit	3.18
		Minimization of Boolean Expressions	1.53
		Minimization of Primitive Flow Table	5.18
		Minterms	1.65
		Modulus-N Counter	4.64
H		MOSFET Ram Cell	3.25
Half Adder	2.4	Multilevel Gate	1.119
Half Subtractor	2.8	Multioutput Gate	1.121
Hamming code	1.35	Multiplexers	3.9
Hazards	5.36		
HDL Based Design Flow	2.38		
HDL for Combinational Circuits	3.70		
HDL for Sequential Logic Circuits	4.82		
Hexadecimal Number System	1.2		
		N	
		NAND and NOR Implementation	1.113
I		Number Base Conversions	1.3
Implementation of Combinational	3.62		
Implementation of Combinational			
Logic Circuits	3.44	Octal Number System	1.2
Implementation of Combinational		Other Logic Gates	1.102
Using PLA	3.53	Other Number Systems	1.28
Implementation of Logic Function	1.109		
Implementations of Combinational Logic	3.15		
Introduction to Hardware			
Description Language	2.36	Parallel Binary Adders	2.12
		Parallel Subtractor	2.13
J		Parity Generator and Checker	2.26
JK Flipflop	4.9	Predefined Types	2.50
		Primitive Flow Table	5.3
K		Product of Sums (POS) Method	1.61
Karnaugh Maps	1.74	Programmable Array Logic	3.61
		Programmable Logic Array	3.51
L		Programmable Logic Devices	3.46
Library	2.49	Programmable ROM	3.49
Logic Families	1.122	PROM	3.32
M		R	
Magnitude Comparator	2.23	Race Free State Assignment	5.14
Master Slave JK Flipflop	4.11	Races	5.12
Maxterms	1.66	RAM Organization	3.22
Memory Cycles	3.36	Realization of One Flipflop using other Flipflops	4.15
Memory Decoding	3.37	Road Operation	3.20
Memory Device	3.18	ROM	3.29

UNIT – I

BOOLEAN ALGEBRA AND LOGIC GATES

- Number systems
- Arithmetic operations
- Binary codes
- Boolean Algebra and Theorems
- Boolean Functions
- Karnaugh Map
- Tabulation Method
- Logic Gates
- NAND and NOR Implementations

BOOLEAN ALGEBRA AND LOGIC GATES

1.1 INTRODUCTION

The number systems are used quite frequently in the field of digital electronics and computers. Number systems of a given radix or base provide the means of quantifying information for processing by digital systems. There are several number systems but the following are the important ones in the field of digital electronics:

- ◆ Decimal number system
- ◆ Octal number system
- ◆ Binary number system
- ◆ Hexadecimal number system

1.2 DECIMAL NUMBER SYSTEM

The decimal number system has 10 numerals or symbols. These symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. The decimal system is also called the base-10 system because it has 10 digits. The position weights in decimal number system is shown in **Figure 1.1**.

10^3	10^2	10^1	10^0	•	10^{-1}	10^{-2}	10^{-3}	10^{-4}
--------	--------	--------	--------	---	-----------	-----------	-----------	-----------

Fig. 1.1: Position weights in binary number system

For example, the decimal number 183 represents one hundred, eight tens and three ones. Any number has two parts, one part is integer part and the other part is fractional part. The decimal point is used to separate the integer and fractional parts of the number. The number 8265.14 is equal to,

$$(8 \times 10^3) + (2 \times 10^2) + (6 \times 10^1) + (5 \times 10^0) \cdot (1 \times 10^{-1}) + (4 \times 10^{-2})$$

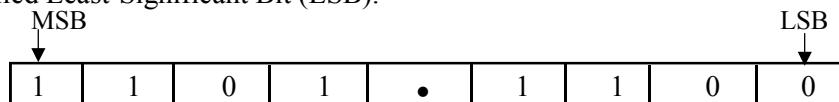
1.3 BINARY NUMBER SYSTEM

The binary number system is a base 2 number system. This number system has two digits 0 and 1. These digits are called BITS. The position of a 1 or 0 in a binary number indicates its weight or value within the number. The weights in a binary number are based in powers of two. The position weights in the binary number system is given in **Figure 1.2**.

2^3	2^2	2^1	2^0	•	2^{-1}	2^{-2}	2^{-3}	2^{-4}
-------	-------	-------	-------	---	----------	----------	----------	----------

Fig. 1.2: Position weights in binary number system

The bit at the left most position is called Most-Significant Bit (MSB) and the bit at the right most position is called Least-Significant Bit (LSB).



1.4 OCTAL NUMBER SYSTEM

The octal number system is a base 8 number system. It has eight digits 0, 1, 2, 3, 4, 5, 6 and 7. Beyond 7, this number system goes as 10, 11, 12, ... and so on. The position weights in an octal number system is shown in **Figure 1.3**.

8^3	8^2	8^1	8^0	•	8^{-1}	8^{-2}	8^{-3}
-------	-------	-------	-------	---	----------	----------	----------

Fig. 1.3: Position weights in an octal number system

1.5 HEXA DECIMAL NUMBER SYSTEM

The hexa decimal number system has a base (radix) of sixteen. It is composed of 16 digits, 0 to 9 and alphabetic characters A, B, C, D, E and F. Hexa decimal numbers are widely used in computer and microprocessor applications. Most digital systems process binary data in groups that are multiples of four bits, making the hexadecimal number very convenient. The digit position in a hexadecimal number system has weights as shown in **Figure 1.4**.

16^3	16^2	16^1	16^0	•	16^{-1}	16^{-2}	16^{-3}
--------	--------	--------	--------	---	-----------	-----------	-----------

Fig. 1.4: Position Weights in Hexa decimal number

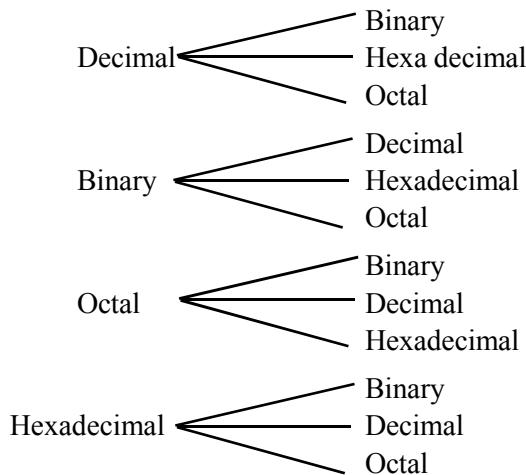
Hexa decimal, Binary and Octal numbers corresponding decimal numbers are given in **Table 1.1**.

Table 1.1: Number Systems

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

1.6 NUMBER BASE CONVERSIONS

Man uses decimal number system while computer uses binary number system. Therefore it is necessary to convert decimal number into its equivalent binary number while feeding the number into the computer. While the computer displaying the result, it is necessary to convert binary number into its equivalent decimal number. When the result is in a large quantity of binary numbers of many bits, like 110011001100001111100, it is inconvenient. Therefore hexa decimal and octal numbers are used as a short hand means of expressing large binary numbers. The following possible conversions can be performed in digital systems:



1.6.1 Decimal to Binary Conversion

Decimal-to-Binary conversion can be done by using repeated division by 2 for integers.

Example 1.1: $15_{10} = ?_2$

$$\begin{array}{r} 2 \mid 15 \\ 2 \mid 7 - 1 \\ 2 \mid 3 - 1 \\ 1 - 1 \end{array}$$

$$15_{10} = 1111_2$$

Example 1.2: $108_{10} = ?_2$

$$\begin{array}{r} 2 \mid 108 \\ 2 \mid 54 - 0 \\ 2 \mid 27 - 0 \\ 2 \mid 13 - 1 \\ 2 \mid 6 - 1 \\ 2 \mid 3 - 0 \\ 1 - 1 \end{array}$$

$$108_{10} = 1101100_2$$

Example 1.3: $0.85_{10} = ?_2$

$$\begin{array}{ll}
 0.85 \times 2 = 1.7 = 0.7 & \text{with a carry of 1} \\
 0.7 \times 2 = 1.4 = 0.4 & \text{with a carry of 1} \\
 0.4 \times 2 = 0.8 = 0.8 & \text{with a carry of 0} \\
 0.8 \times 2 = 1.6 = 0.6 & \text{with a carry of 1} \\
 0.6 \times 2 = 1.2 = 0.2 & \text{with a carry of 1} \\
 0.2 \times 2 = 0.4 = 0.4 & \text{with a carry of 0} \\
 \end{array}$$

$0.85_{10} = 0.110110_2$ (approximate)



Example 1.4: $0.3125_{10} = ?_2$

$$\begin{array}{ll}
 0.3125 \times 2 = 0.625 = 0.625 & \text{with a carry of 0} \\
 0.625 \times 2 = 1.25 = 0.25 & \text{with a carry of 1} \\
 0.25 \times 2 = 0.50 = 0.50 & \text{with a carry of 0} \\
 0.50 \times 2 = 1.00 = 0.00 & \text{with a carry of 1} \\
 \end{array}$$

$0.3125_{10} = 0.0101_2$



Example 1.5: $0.625_{10} = ?_2$

$$\begin{array}{ll}
 0.625 \times 2 = 1.25 = 0.25 & \text{with a carry of 1} \\
 0.25 \times 2 = 0.50 = 0.50 & \text{with a carry of 0} \\
 0.50 \times 2 = 1.00 = 0.00 & \text{with a carry of 1} \\
 \end{array}$$



$0.625_{10} = 101_2$

1.6.2 Decimal-to-Hexa Decimal Conversion

The decimal to hexadecimal conversion is explained with the following examples:

Example 1.6: $2479_{10} = ?_{16}$

$$\begin{array}{r}
 16 \overline{)2479} \\
 16 \overline{)154 - 15} \quad (\text{F}) \\
 \phantom{16 \overline{)154}} \overline{9 - 10} \quad (\text{A})
 \end{array}
 \qquad \qquad \qquad 2479_{10} = 9AF_{16}$$

Example 1.7: $3507_{10} = ?_{16}$

$$\begin{array}{r}
 16 \overline{)3507} \\
 16 \overline{)219 - 3} \\
 \phantom{16 \overline{)219}} \overline{13 - 11} \quad (\text{B})
 \end{array}
 \qquad \qquad \qquad 3507_{10} = DB3_{16}$$

Example 1.8: $0.62_{10} = ?_{16}$

$$\begin{array}{ll}
 0.62 \times 16 = 9.92 = 0.92 & \text{with a carry of 9} \\
 0.92 \times 16 = 14.72 = 0.72 & \text{with a carry of 14(E)} \\
 0.72 \times 16 = 11.52 = 0.52 & \text{with a carry of 11(B)} \\
 0.52 \times 16 = 8.32 = 0.32 & \text{with a carry of 8} \\
 0.62_{10} = 0.9EB8_{16} &
 \end{array}$$

**Example 1.9:** $4019.257_{10} = ?_{16}$

$$\begin{array}{ll}
 \begin{array}{r}
 16 \overline{)4019} \\
 16 \overline{)251 - 3} \\
 \hline 15 - 11 (\text{B})
 \end{array} & 4019_{10} = FB3_{16} \\
 0.257 \times 16 = 4.112 = 0.112 & \text{with a carry of 4} \\
 0.112 \times 16 = 1.792 = 0.792 & \text{with a carry of 1} \\
 0.792 \times 16 = 12.672 = 0.672 & \text{with a carry of 12(C)} \\
 0.672 \times 16 = 10.752 = 0.752 & \text{with a carry of 10(A)}
 \end{array}$$



$$4019.257_{10} = FB3.41CA_{16}$$

1.6.3 Decimal-to-Octal conversion

The decimal to octal conversion can be done by using repeated division by 8.

Example 1.10: $153_{10} = ?_8$

$$\begin{array}{r}
 8 \overline{)153} \\
 8 \overline{)19 - 1} \\
 \hline 2 - 3
 \end{array}$$

$$153_{10} = 231_8$$

Example 1.11: $265_{10} = ?_8$

$$\begin{array}{r}
 8 \overline{)265} \\
 8 \overline{)33 - 1} \\
 \hline 4 - 1
 \end{array}$$

$$265_{10} = 411_8$$

Example 1.12: $0.55_{10} = ?_8$

$$\begin{array}{ll}
 0.55 \times 8 = 4.4 = 0.4 & \text{with a carry of 4} \\
 0.4 \times 8 = 3.2 = 0.2 & \text{with a carry of 3} \\
 0.2 \times 8 = 1.6 = 0.6 & \text{with a carry of 1} \\
 0.6 \times 8 = 4.8 = 0.8 & \text{with a carry of 4} \\
 0.8 \times 8 = 6.4 = 0.4 & \text{with a carry of 6}
 \end{array}$$



$$0.55_{10} = 0.43146_8 \text{ (approximate)}$$

Example 1.13: $2479.64_{10} = ?_8$

$$\begin{array}{r}
 8 \overline{)2497} \\
 8 \quad \boxed{312 - 1} \\
 8 \quad \boxed{39 - 0} \\
 \hline
 4 - 7
 \end{array}
 \qquad
 2479_{10} = 4701_8$$

0.64 \times 8 = 5.12 = 0.12 with a carry of 5
 0.12 \times 8 = 0.96 = 0.96 with a carry of 0
 0.96 \times 8 = 7.68 = 0.68 with a carry of 7
 0.68 \times 8 = 5.44 = 0.44 with a carry of 5
 $2497.64_{10} = 4701.5075_8$

1.6.4 Binary-to-Decimal Conversion

The positional weight for the binary number system and decimal equivalents are given in **Figure 1.5.**

16	8	4	2	1	•	0.5	0.25	0.125	0.0625	0.03125
2^4	2^3	2^2	2^1	2^0	•	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}

Fig. 1.5: Positional weight and decimal equivalents

Example 1.14: $111011_2 = ?_{10}$

$$\begin{aligned}
 & (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\
 & = 32 + 16 + 8 + 0 + 2 + 1 = 59_{10}
 \end{aligned}$$

Example 1.15: $0.1101_2 = ?_{10}$

$$\begin{aligned}
 & (1 \times 2^{-1}) + (1 \times 2^{-2}) + (0 \times 2^{-3}) + (1 \times 2^{-4}) \\
 & = 0.5 + 0.25 + 0 + 0.0625 = 0.8125_{10}
 \end{aligned}$$

Example 1.16: $10010.011_2 = ?_{10}$

$$\begin{aligned}
 & (1 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) \cdot (0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}) \\
 & = (16 + 0 + 0 + 2 + 0) \times (0 + 0.25 + 0.125) = 18.375_{10}
 \end{aligned}$$

Example 1.17: $01010110.0110_2 = ?_{10}$

$$\begin{aligned}
 & (0 \times 2^7) + (1 \times 2^6) + (0 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) \cdot (0 \times 2^{-1}) \\
 & \quad + (1 \times 2^{-2}) + (1 \times 2^{-3}) + (0 \times 2^{-4}) \\
 & = (0 + 64 + 0 + 16 + 0 + 4 + 2 + 0) \times (0 + 0.25 + 0.125 + 0) = 86.375_{10}
 \end{aligned}$$

1.6.5 Binary-to-Hexa Decimal Conversion

The binary to hexadeciml conversion is explained with the following examples:

Example 1.18: $111101010110_2 = ?_{16}$

$$\begin{array}{ccc} \overbrace{1111} & \overbrace{0101} & \overbrace{0110} \\ F & 5 & 6 \end{array}$$

$$(111101010110)_2 = (F56)_{16}$$

Example 1.19: $1111110000_2 = ?_{16}$

$$\begin{array}{ccc} \overbrace{0011} & \overbrace{1111} & \overbrace{0000} \\ 3 & F & 0 \end{array}$$

$$(1111110000)_2 = (3F0)_{16}$$

Example 1.20: $1110100011010110_2 = ?_{16}$

$$\begin{array}{cccc} \overbrace{1110} & \overbrace{1000} & \overbrace{1101} & \overbrace{0110} \\ E & 8 & D & 6 \end{array}$$

$$= (E8D6)_{16}$$

Example 1.21: $0.10110110_2 = ?_{16}$

$$\begin{array}{cc} \overbrace{1011} & \overbrace{0110} \\ B & 6 \end{array}$$

$$(0.10110110)_2 = (0.B6)_{16}$$

Example 1.22: $(10111011.1111001)_2 = (?)_{16}$

$$\begin{array}{cccc} \overbrace{1011} & \overbrace{1011} & \cdot & \overbrace{1111} \\ B & B & \cdot & F \end{array} \quad \begin{array}{c} \overbrace{0010} \\ 2 \end{array}$$

$$= (B\ B\ \cdot\ F\ 2)_{16}$$

Example 1.23: $(10110001101011.1101101)_2 = ?_{16}$

$$\begin{array}{ccccccc} \overbrace{0010} & \overbrace{1100} & \overbrace{0110} & \overbrace{1011} & \cdot & \overbrace{1101} & \overbrace{1010} \\ 2 & C & 6 & B & \cdot & D & A \end{array}$$

$$= (2C6B\cdot DA)_{16}$$

1.6.6 Binary-to-Octal Conversion

The binary to octal conversion is performed by forming a 3 bit binary group and converting each 3 bit binary to its octal equivalent.

Example 1.24: $(10101111)_2 = ?_8$

$$\begin{array}{c} \overbrace{010} \\ 2 \end{array} \quad \begin{array}{c} \overbrace{101} \\ 5 \end{array} \quad \begin{array}{c} \overbrace{111} \\ 7 \end{array} \quad (10101111)_2 = (257)_8$$

Example 1.25: $(0.0110111)_2 = ?_8$

$$\begin{array}{c} \overbrace{011} \\ 3 \end{array} \quad \begin{array}{c} \overbrace{011} \\ 3 \end{array} \quad \begin{array}{c} \overbrace{100} \\ 1 \end{array} \quad (0.0110111)_2 = (0.331)_8$$

Example 1.26: $(1011011.01101)_2 = ?_8$

$$\begin{array}{c} \overbrace{001} \\ 1 \end{array} \quad \begin{array}{c} \overbrace{011} \\ 3 \end{array} \quad \begin{array}{c} \overbrace{011} \\ 3 \end{array} \cdot \begin{array}{c} \overbrace{011} \\ 3 \end{array} \quad \begin{array}{c} \overbrace{010} \\ 2 \end{array} \quad (1011011.01101)_2 = (133.32)_8$$

Example 1.27: $(1010011.00101)_2 = ?_8$

$$\begin{array}{c} \overbrace{001} \\ 1 \end{array} \quad \begin{array}{c} \overbrace{010} \\ 2 \end{array} \quad \begin{array}{c} \overbrace{011} \\ 3 \end{array} \quad \cdot \quad \begin{array}{c} \overbrace{001} \\ 1 \end{array} \quad \begin{array}{c} \overbrace{010} \\ 2 \end{array} \quad (1010011.00101)_2 = (123.12)_8$$

1.6.7 Octal-to-Binary Conversion

The octal number is converted into binary number by a group of 3 bits

Example 1.28: $(3574)_8 = ?_2$

$$\begin{array}{cccc} 3 & 5 & 7 & 4 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 011 & 101 & 111 & 100 \end{array} \quad (3574)_8 = (11101111100)_2$$

Example 1.29: $(0.7460)_8 = ?_2$

$$\begin{array}{cccc} 7 & 4 & 6 & 0 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 111 & 100 & 110 & 000 \end{array} \quad (0.7460)_8 = (0.111100110)_2$$

Example 1.30: $(34.321)_8 = ?_2$

$$\begin{array}{ccccc} 3 & 4 & . & 3 & 2 & 1 \\ \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow \\ 011 & 100 & . & 011 & 010 & 001 \end{array} \quad (34.321)_8 = (11100.011010001)_2$$

1.6.8 Octal-to-Decimal Conversion

The positional weight for the octal number system and their decimal equivalents are given in **Figure 1.6**.

512	64	8	1	•	0.125	0.015625	0.00195
8^3	8^2	8^1	8^0	•	8^{-1}	8^{-2}	8^{-3}

Fig. 1.6: Positional weight and decimal equivalent of octal number system

Example 1.31: $465_8 = ?_{10}$

$$\begin{aligned}
 & (4 \times 8^2) + (6 \times 8^1) + (5 \times 8^0) \\
 &= (4 \times 64) + (6 \times 8) + (5 \times 1) \\
 &= 256 + 48 + 5 \\
 &= 309 \quad (465)_8 = (309)_{10}
 \end{aligned}$$

Example 1.32: $0.731_8 = ?_{10}$

$$\begin{aligned}
 & (7 \times 8^{-1}) + (3 \times 8^{-2}) + (1 \times 8^{-3}) \\
 &= (7 \times 0.125) + (3 \times 0.015625) + (1 \times 0.00195) \\
 &= 0.923825_{10} \quad (0.731)_8 = (0.923825)_{10}
 \end{aligned}$$

Example 1.33: $326.216_8 = ?_{10}$

$$\begin{aligned}
 & (3 \times 8^2) + (2 \times 8^1) + (6 \times 8^0) + (2 \times 8^{-1}) + (1 \times 8^{-2}) + (6 \times 8^{-3}) \\
 &= (3 \times 64) + (2 \times 8) + (6 \times 1) \cdot \left(2 \times \frac{1}{8}\right) + \left(1 \times \frac{1}{8^2}\right) + \left(6 \times \frac{1}{8^3}\right) \\
 &= 214.27734375_{10}
 \end{aligned}$$

$$(326.216)_8 = 214.27734375_{10}$$

1.6.9 Octal-to-Hexa Decimal Conversion

Example 1.34: $327_8 = ?_{16}$

	3	2	7	
Octal to Binary	↓	↓	↓	
	011	010	111	
	<u>0000</u>	<u>1101</u>	<u>0111</u>	
Binary to Hex	0	D	7	

$$(327)_8 = (D7)_{16}$$

Example 1.35: $615_8 = ?_{16}$

Octal to Binary	6 1 5
	\downarrow \downarrow \downarrow
	110 001 101
Binary to Hex	$\underbrace{0001}_{1}$ $\underbrace{1000}_{8}$ $\underbrace{1101}_{D}$
	$(615)_8 = (18D)_{16}$

Example 1.36: $1024.102_8 = ?_{16}$

Octal to Binary	1 0 2 4 . 1 0 2
	\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow
	001 000 010 100 001 000 010
Binary to Hex	$\underbrace{0010}_{2}$ $\underbrace{0001}_{1}$ $\underbrace{0100}_{4}$. $\underbrace{0010}_{2}$ $\underbrace{0001}_{1}$ $\underbrace{0000}_{0}$
	$(1024.102)_8 = (214.210)_{16}$

1.6.10 Hexadecimal-to-Binary Conversion

The hexadecimal-to-binary conversion is explained with following examples:

Example 1.37: $306_{16} = ?_2$

3 0 6
\downarrow \downarrow \downarrow
0011 0000 0110
$(306)_{16} = (1100000110)_2$

Example 1.38: $9AF.F_{16} = ?_2$

9 A F . F
\downarrow \downarrow \downarrow \downarrow
1001 1010 1111 1111
$(9AF.F)_{16} = (100110101111.1111)_2$

Example 1.39: $7AF4.BB_{16} = ?_2$

7 A F 4 . B B
\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow
0111 1010 1111 0100 . 1011 1011
$(7AF4.BB)_{16} = (111101011110100.10111011)_2$

1.6.11 Hexa Decimal-to-Decimal Conversion

The positional weight for the hexadecimal number system and their decimal equivalents are given in **Figure 1.7**.

4096	256	16	1	•	0.0625	0.0039	0.00024
16^3	16^2	16^1	16^0	•	16^{-1}	16^{-2}	16^{-3}

Fig. 1.7: Positional weight and decimal equivalent of Hex

Example 1.40: $2F59_{16} = ?_{10}$

$$\begin{aligned}
 & (2 \times 16^3) + (F \times 16^2) + (5 \times 16^1) + (9 \times 16^0) \\
 &= (2 \times 4096) + (15 \times 256) + (5 \times 256) + (5 \times 16) + 9 \\
 &= 12121_{10} \quad (2F59)_{16} = (12121)_{10}
 \end{aligned}$$

Example 1.41: $ABCD_{16} = ?_{10}$

$$\begin{aligned}
 & (A \times 16^3) + (B \times 16^2) + (C \times 16^1) + (D \times 16^0) \\
 &= (10 \times 4096) + (11 \times 256) + (12 \times 16) + (13 \times 1) \\
 &= 40960 + 2816 + 192 + 13 \\
 &= 43981_{10} \quad (ABCD)_{16} = (43981)_{10}
 \end{aligned}$$

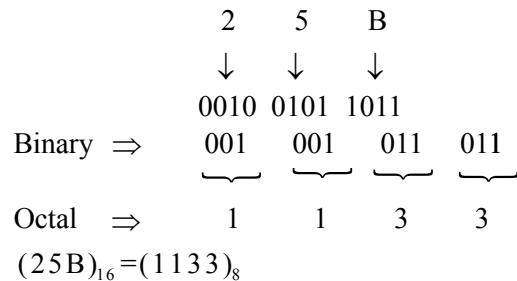
Example 1.42: $F8E6_{16} \cdot 39_{10} = ?_{10}$

$$\begin{aligned}
 & (15 \times 16^3) + (8 \times 16^2) + (14 \times 16^1) + (6 \times 16^0) \cdot (3 \times 16^{-1}) + (9 \times 16^{-2}) \\
 &= (61440 + 2048 + 224 + 6) \cdot (0.1875 + 0.0352) \\
 &= 63718.2227_{10} \quad (F8E6 \cdot 39)_{16} = (63718.2227)_{10}
 \end{aligned}$$

1.6.12 Hexa Decimal-to-Octal Conversion

Convert the given hexadecimal number into its equivalent 4 bit binary number and regroup the bits in 3 bit group. Then 3 bit binary number is converted into octal number.

Example 1.43: $25B_{16} = ?_8$



Example 1.44: A 2 4 6₁₆ = ?₈

$$\begin{array}{ccccccc}
 & A & 2 & 4 & 6 & & \\
 & \downarrow & \downarrow & \downarrow & \downarrow & & \\
 1010 & 0010 & 0100 & 0110 & & & \\
 \text{Binary} \Rightarrow & \underbrace{001} & \underbrace{010} & \underbrace{001} & \underbrace{001} & \underbrace{000} & \underbrace{110} \\
 \text{Octal} \Rightarrow & 1 & 2 & 1 & 1 & 0 & 6 \\
 (A246)₁₆ = (121106)₈
 \end{array}$$

Example 1.45: 5 C 2 . 3 9₁₆ = ?₈

$$\begin{array}{ccccccc}
 & 5 & C & 2 & . & 3 & 9 \\
 & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow \\
 0101 & 1100 & 0010 & & 0011 & 1001 & \\
 \text{Binary} \Rightarrow & \underbrace{010} & \underbrace{111} & \underbrace{000} & \underbrace{010} & \cdot \underbrace{001} & \underbrace{110} \underbrace{010} \\
 \text{Octal} \Rightarrow & 2 & 7 & 0 & 2 & 1 & 6 & 2 \\
 (5C2.39)₁₆ = (2702.162)₈
 \end{array}$$

1.7 COMPLEMENTS

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. There are two types of complements:

- ◆ r's complement
- ◆ (r - 1)'s complement

For binary numbers, r(base) = 2

- ◆ 2's complement
- ◆ 1's complement

For decimal numbers, r(base) = 10

- ◆ 10's complement
- ◆ 9's complement

1.7.1 1's Complement

The 1's complement of a binary number is the number that results when we complement each bit. If the binary number is,

$$A_3 A_2 A_1 A_0 = 1001$$

The 1's complement is

$$\bar{A}_3 \bar{A}_2 \bar{A}_1 \bar{A}_0 = 0110$$

Therefore, the 1's complement of a binary number is formed by changing 1's to 0's and 0's to 1's. The following are some examples:

The 1's complement of 10110001 is 01001110

The 1's complement of 1111 is 0000

1.7.2 2's Complement

The 2's complement is the binary number that results when we add 1 to the 1's complement. i.e.,

2's complement = 1's complement + 1

If the binary number is 1101

1's complement \Rightarrow 0010

1(+)

2's complement \Rightarrow 0011

Some other examples of 2's complements:

Binary Number	1's Complement	2's Complement
1000 0001	0111 1110	0111 1111
1111 1001	0000 0110	0000 0111

1.7.3 1's Complement Subtraction

Subtraction of binary numbers using 1's complement method allows subtraction only by addition. To subtract a smaller number from a large number ($X - Y$), the 1's complement method as follows:

- ◆ Obtain the 1's complement of the smaller number (Y)
- ◆ Add this to the large number (X)
- ◆ Remove the carry and add it to the result. This carry is called 'end-around-carry'.

Example 1.46: $X - Y = 14 - 10 = 4$

$$X = 1110, Y = 1010$$

1's complement of $Y = 0101$

Solution: $X = 1110$

1's complement of $Y = \underline{0101} (+)$

Sum = 1 0011

End-around carry = 1 (+)

$$X - Y = \underline{0100}$$

Example 1.47: $(X - Y) = 9 - 3 = 6$

$$X = 1001$$

1's complement of $Y = \underline{\hspace{2cm}} 1100 (+)$
 Sum = 1 0101

End-around carry = $\underline{\hspace{2cm}} 1 (+)$
 $X - Y = \underline{\hspace{2cm}} 0110$

Subtraction of a large number (X) from a smaller (Y), the 1's complement method as follows:

- ◆ Obtain the 1's complement of the larger number (X).
- ◆ Add this to the smaller number (Y).
- ◆ The answer is the 1's complement of the result and is opposite in sign. There is no carry.

Example 1.48: $Y - X = 10 - 14 = -4$

$$X = 1110$$

1's complement of $X = 0001$

$$Y = 1010$$

Solution: $Y = 1010$

1's complement of $X = \underline{\hspace{2cm}} 0001 (+)$
 $\text{Sum} = \underline{\hspace{2cm}} 1011$

1's complement of result (1011) is 0100 and is opposite sign. i.e., -0100 .

Example 1.49: $Y - X = 3 - 9 = -6$

$$X = 1001$$

1's complement of $X = 0110$

$$Y = 0011$$

Solution: $Y = 0011$

1's complement of $X = \underline{\hspace{2cm}} 0110 (+)$
 $\text{Sum} = \underline{\hspace{2cm}} 1001$

1's complement of result (1001) is 0110 and is opposite sign.

i.e., -0110 .

1.7.4 2's Complement Subtraction

The subtraction of a small number from a large number by the 2's complement method is as follows:

- (i) Determine the 2's complement of the smaller number
- (ii) Add this is to the larger number
- (iii) Omit the carry.

Example 1.50: $X - Y = 14 - 10 = 4$

$$X = 1110$$

$$Y = 1010$$

1's complement of $Y = 0101$

$$\underline{\quad\quad\quad} \quad 1 (+)$$

2's complement of $Y = \underline{0110}$

Solution: $X = 1110$

2's complement of $Y = \underline{0110} (+)$

$$\underline{1 \quad 0100}$$

The carry (1) is discarded. Therefore the result is (0100).

Example 1.51: $X - Y = 1010100 - 1000011$

1's complement of $Y = 0111100$

$$\underline{\quad\quad\quad} \quad 1 (+)$$

2's complement of $Y = \underline{0111101}$

Solution: $X = 1010100$

2's complement of $Y = \underline{0111101} (+)$

$$\underline{1 \quad 0010001}$$

Ans: $X - Y = 0010001$

The subtraction of a larger number from a smaller number is as follows:

- ◆ Determine the 2's complement of the larger number.
- ◆ Add 2's complement to the smaller number.
- ◆ There is no carry. The result is in 2's complement form and is negative.
- ◆ To get the true answer, take 2's complement of the result and change the sign.

Example 1.52: $Y - X = 10 - 14 = -4$

$$Y = 1010$$

$$X = 1110$$

1's complement of $X = 0001$

$$\underline{\quad\quad\quad} \quad 1 (+)$$

2's complement of $X = \underline{0010}$

Solution: $Y = 1010$

2's complement of $X = \underline{0010} (+)$

$$\underline{\quad\quad\quad}$$

$Y - X = - (2's \text{ complement of } 1100)$

1's complement of $1100 = 0011$

$$\begin{array}{r} & 1 \\ & \underline{0100} \\ 2\text{'s complement of } 1100 = & \underline{\underline{0100}} \end{array}$$

Therefore the true answer is -0100

Example 1.53: $(Y - X) = 9 - 10 = -1$

$$Y = 1001$$

$$X = 1010$$

1's complement of $X = 0101$

$$\begin{array}{r} & 1 (+) \\ & \underline{0110} \\ 2\text{'s complement of } X = & \underline{\underline{0110}} \end{array}$$

Solution: $Y = 1001$

$$\begin{array}{r} & 1 (+) \\ & \underline{0110} (+) \\ & \underline{\underline{1111}} \\ (Y - X) = - (2\text{'s complement of } 1111) & \end{array}$$

1's complement of $1111 = 0000$

$$\begin{array}{r} & 1 (+) \\ & \underline{0001} \\ 2\text{'s complement of } 1111 = & \underline{\underline{0001}} \\ Y - X = - & 0001 \end{array}$$

1.7.5 Comparison of 1's and 2's complements

- ❖ The 1's complement can be easily obtained using an inverter. The 2's complement is obtained as adding '1' to the 1's complement.
- ❖ The 2's complement system requires only one arithmetic operation, but the 1's complement system requires two arithmetic operations.
- ❖ The 1's complement is used in logical manipulation for inversion operation; the 2's complement is used only for arithmetic applications.

1.7.6 9's Complement

The 9's complement of a decimal number is found by subtracting each digit in the number from 9. The examples of 9's complement are:

(i) 9's complement of 71 is, 99

$$\begin{array}{r} & 17 (-) \\ & \underline{82} \\ (i) \quad 9\text{'s complement of } 71 \text{ is, } & \underline{\underline{99}} \end{array}$$

(ii) 9's complement of 444 is, 999

$$\begin{array}{r} & 444 (-) \\ & \underline{555} \\ (ii) \quad 9\text{'s complement of } 444 \text{ is, } & \underline{\underline{999}} \end{array}$$

(iii) 9's complement of 1542701 is, 9999999

$$\begin{array}{r} 1542701 \\ - 9999999 \\ \hline 8457298 \end{array}$$

1.7.7 10's Complement

The 10's complement of a decimal number that results add 1 to its 9's complement.

(i) 10's complement of 71 is, 99

$$\begin{array}{r} 17 (-) \\ - 62 \\ \hline 1 (+) \\ \hline 83 \end{array}$$

(ii) 10's complement of 444 is, 999

$$\begin{array}{r} 444 (-) \\ - 555 \\ \hline 1 (+) \\ \hline 556 \end{array}$$

(iii) 10's complement of 1542701 is, 9999999

$$\begin{array}{r} 1542701 (-) \\ - 9999999 \\ \hline 8457298 \\ + 1 (+) \\ \hline 8457299 \end{array}$$

1.7.8 9's Complement Subtraction

The 9's complement subtraction method is same as the 1's complement subtraction method. The 9's complement method for subtraction a smaller number from a larger number is as follows:

- ◆ Determine the 9's complement of the smaller number
- ◆ Add this is to the larger number
- ◆ Remove the carry and add it to the result

Example 1.54: $98 - 18 = 80$

9's complement of 18, $99 - 18 = 81$

$$\begin{array}{r} 98 \\ - 81 (+) \\ \hline \text{Carry } (1) 79 \\ \hline 80 \end{array}$$

Example 1.55: $55 - 10 = 45$

9's complement of 10 is, $99 - 10 = 89$

$$\begin{array}{r} 55 \\ 89 \quad (+) \\ \hline \text{Carry } (1) \ 44 \\ \underline{1} \quad (+) \\ \hline 45 \end{array}$$

Subtraction of a larger number from a smaller one by the 9's complement method is as follows:

- ◆ Determine the 9's complement of the larger number.
- ◆ Add this is to the smaller number.
- ◆ The answer is the 1's complement of the true result and is opposite in sign.

Example 1.56: $15 - 67 = -52$

9's complement of 67 is, $99 - 67 = 32$

$$\begin{array}{r} 15 \\ 32 \quad (+) \\ \hline 47 \end{array}$$

True result = - (1's complement of 47)

i.e., $- (99 - 47) = -52$

Example 1.57: $265 - 347 = -082$

9's complement of 347 is, $999 - 347 = 652$

$$\begin{array}{r} 265 \\ 652 \quad (+) \\ \hline 917 \end{array}$$

True result = - (1's complement of 917)

$$\begin{aligned} &= - (999 - 917) \\ &= - (082) \end{aligned}$$

1.7.9 10's Complement Subtraction

Subtraction of a smaller number from a larger number by the 10's complement method is as follows:

- ◆ Determine the 10's complement of the smaller number.
- ◆ Add this is to the larger number.
- ◆ The carry is discarded.

Example 1.58: $25 - 15 = 10$ 1's complement of 15 = $99 - 15 = 84$ 2's complement of 15 = $84 + 1 = 85$

$$\begin{array}{r} 25 \\ 85 (+) \\ \hline (1) \underline{10} \end{array}$$

Ans: 10**Example 1.59:** $786 - 412 = 374$ 1's complement of 412 $\Rightarrow 999$

$$\begin{array}{r} 412 \\ \hline 587 \\ \hline 1 (+) \end{array}$$

2's complement of 412 $\Rightarrow \underline{\underline{588}}$

$$\begin{array}{r} 786 \\ 588 (+) \\ \hline \text{Carry (1)} \quad \underline{\underline{374}} \end{array}$$

Ans: 374

The 10's complement method for subtraction of a larger number from a smaller number is as follows:

- ◆ Determine the 10's complement of larger number.
- ◆ Add this to the smaller number.
- ◆ The result is in 10's complement form and is negative.
- ◆ To get an answer in true form, take the 10's complement and change the sign.

Example 1.60: $31 - 57 = -26$ 9's complement of 57 $\Rightarrow 99$

$$\begin{array}{r} 57 (-) \\ \hline 42 \\ \hline 1 (+) \end{array}$$

10's complement of 57 $\Rightarrow \underline{\underline{43}}$

$$\begin{array}{r} 31 \\ 43 (+) \\ \hline \underline{\underline{74}} \end{array}$$

Result = $-(10\text{'s complement of } 74)$

9's complement of 74 is 99

$$\begin{array}{r} 74 (-) \\ \hline 25 \end{array}$$

10's complement of 74 is $25 + 1 = 26$. $\therefore 31 - 57 = -26$.

Example 1.61: $265 - 347 = - 082$

9's complement of 347 \Rightarrow 999

$$\begin{array}{r} 347 (-) \\ \hline 652 \end{array}$$

9's complement of 347 \Rightarrow 653

$$\begin{array}{r} 265 \\ 653 (+) \\ \hline 918 \end{array}$$

Result = - (10's complement of 918)

9's complement of 918 \Rightarrow 999

$$\begin{array}{r} 918 (-) \\ \hline 81 \\ 1 (+) \end{array}$$

10's complement of 918 \Rightarrow 82

$$\therefore 265 - 347 = - 82.$$

NOTE: The same procedure is used to find the solution for subtraction with 7's complement, 8's complement, 15's complement and 16's complement.

1.8 SIGNED BINARY NUMBERS

Digital systems must be able to handle both positive and negative numbers. In ordinary arithmetic, a positive number is indicated by a '+' sign and a negative number is indicated by a '-' sign. But in binary numbers, the left most bit (sign bit) denotes the sign.

0 is used for the +ve sign and 1 is used for the -ve sign. Therefore $-001, -111$ are coded as 1001, 1111. These numbers contain a sign bit followed by magnitude bits. Numbers in this form are called signed binary numbers.

When a signed binary number is represented in sign-magnitude form, the left-most bit is the sign bit and the remaining bits are the magnitude bits. For example + 32 is expressed as an 8 bit signed binary number using the sign-magnitude form as,



- 32 is represented as 10100000

Thus in the sign-magnitude form, a negative number has the same magnitude bits as the corresponding positive number, but the sign bit is '1' rather than '0'.

Some other examples are:

(i) $-18 \Rightarrow 10010010$

$+18 \Rightarrow 00010010$

(ii) $-97 \Rightarrow 11100001$

$+97 \Rightarrow 01100001$

The sign-magnitude for the decimal numbers (- 15 to + 15) are represented in **Table 1.2**.

In the 1's complement form of signed binary numbers, a negative number is the 1's complement of the corresponding positive number.

For example, - 8 is represented as 10001000

Signed 1's complement \Rightarrow 11110111 (1's complement of + 8)

In the 2's complement form, a negative number is the 2's complement of the corresponding positive number. Signed 2's complement of - 8 is,

$$\begin{array}{r} 11110111 \\ \hline 1 (+) \\ \hline 11111000 \end{array} \text{ (2's complement of + 8)}$$

Range of signed numbers

For 8 bit numbers,

Unsigned numbers range = 0 to $2^n - 1$

$$= 0 \text{ to } 255$$

Signed numbers range $= -(2^{n-1}) \text{ to } +(2^{n-1} - 1)$
 $= -128 \text{ to } +127$

16 bit signed numbers range = - 32768 to + 32767

Table 1.2: Sign-Magnitude Numbers

<i>Decimal</i>	<i>Sign Magnitude</i>	<i>Decimal</i>	<i>Sign Magnitude</i>
+ 15	01111	- 15	11111
+ 14	01110	- 14	11110
+ 13	01101	- 13	11101
+ 12	01100	- 12	11100
+ 11	01011	- 11	11011
+ 10	01010	- 10	11010
+ 9	01001	- 9	11001
+ 8	01000	- 8	11000
+ 7	00111	- 7	10111
+ 6	00110	- 6	10110
+ 5	00101	- 5	10101
+ 4	00100	- 4	10100
+ 3	00011	- 3	10011
+ 2	00010	- 2	10010
+ 1	00001	- 1	10001
+ 0	00000	- 0	10000

1.9 BINARY ARITHMETIC

1.9.1 Binary Addition

$A + B$	Sum	Carry
0 + 0	0	0
0 + 1	1	0
1 + 0	1	0
1 + 1	0	1

The examples for binary addition are given below:

Example 1.62:

$$\begin{array}{r} 0011 \\ 1010 (+) \\ \hline 1101 \end{array}$$

Example 1.63:

$$\begin{array}{r} 0101 \quad 0111 \\ 0011 \quad 0101 (+) \\ \hline 1000 \quad 1100 \end{array}$$

Example 1.64

$$\begin{array}{r} 57.5 + 0.3125 = 57.8125 \\ 111001.1000 \\ 000000.0101 (+) \\ \hline 111001.1101 \end{array}$$

Example 1.65

$$\begin{array}{r} 31.75 + 19.25 = 51.00 \\ 011111.110 \\ 010011.010 (+) \\ \hline 110011.000 \end{array}$$

Example 1.66

$$\begin{array}{r} 101.01 + 110 + 10.1 = ? \\ 101.01 \\ 110.00 \\ \hline 1011.01 \\ 10.10 \\ \hline 1101.11 \end{array}$$

Example 1.67

$$101101 + 1110 + 110 = ?$$

101101

1110

110

1000001**Note:** $1 + 1 = 10$

$$1 + 1 + 1 = 11$$

$$1 + 1 + 1 + 1 = 100$$

1.9.2 Binary Subtraction

$A - B$	Difference	Borrow
0 – 0	0	0
0 – 1	1	1
1 – 0	1	0
1 – 1	0	0

Example 1.68: 1011

$$\begin{array}{r} 0101 \\ (-) \\ \hline 0110 \end{array}$$

Example 1.69: 111.01

$$\begin{array}{r} 100.10 \\ (-) \\ \hline 010.11 \end{array}$$

Example 1.70: 10101.001

$$\begin{array}{r} 01110.110 \\ (-) \\ \hline 00110.011 \end{array}$$

Note: For binary subtraction, use 1's complement and 2's complement methods also.**1.9.3 Binary Multiplication**

The four basic rules for multiplying bits are as follows:

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

Example 1.71: 110

$$\begin{array}{r} 101 \\ \hline 110 \\ 000 \\ \hline 110 \\ \hline 1110 \end{array}$$

Example 1.72: 110.1×10.1

$$\begin{array}{r} 110.1 \\ \times 10.1 \\ \hline 0000 \\ 1101 \\ \hline 10000.01 \end{array}$$

1.9.4 Binary Division

Example 1.73: $110 \div 11$

$$\begin{array}{r} 10 \\ 11 \overline{)110} \\ -11 \\ \hline 000 \\ \hline \end{array} \quad 110 \div 11 = 10$$

Example 1.74: $110 \div 10$

$$\begin{array}{r} 11 \\ 10 \overline{)110} \\ -10 \\ \hline 10 \\ -10 \\ \hline 00 \\ \hline \end{array} \quad 110 \div 10 = 11$$

1.9.5 Addition with signed numbers

There are 4 cases that can occur when two signed binary numbers are added.

- ◆ Both numbers are +ve
- ◆ +ve number with magnitude larger than –ve number
- ◆ –ve with magnitude larger than +ve number
- ◆ Both numbers are –ve

1. Addition of two positive numbers yields a positive number

$$\begin{array}{r} 6 \quad 00000110 \\ 13 (+) \quad 00001101 \\ \hline 19 \quad \underline{00010011} \end{array}$$

2. Addition of a positive number and a smaller negative number yields a positive number,

$$\begin{array}{r}
 + 13 & 00001101 \\
 - 6 (+) & 11111010 \\
 \hline
 + 7 & (1) \quad \underline{00000111} \\
 \downarrow & \\
 \text{discard carry}
 \end{array}$$

3. Addition of a positive number and a larger negative number yields a negative number in 2's complement

$$\begin{array}{r}
 + 6 & 00000110 \\
 - 13 (+) & 11110011 \\
 \hline
 17 & (1) \quad \underline{11111001} \\
 \downarrow & \\
 \text{discard carry}
 \end{array}$$

The sum is negative and therefore 2's complement form.

4. Addition of two negative numbers yields a negative number in 2's complement

$$\begin{array}{r}
 - 6 & 11111010 \\
 - 13 (+) & 11110011 \\
 \hline
 - 19 & \underline{11101101}
 \end{array}$$

The sum is negative and therefore in 2's complement form.

Example 1.75: Perform each of the following computations using signed, 8 bit words in 1's complement and 2's complement binary arithmetic. (Dec. 2005)

1. $(+95)_{10} + (-63)_{10}$
2. $(+42)_{10} + (-87)_{10}$
3. $(-13)_{10} + (-59)_{10}$
4. $(+38)_{10} + (-38)_{10}$
5. $(-105)_{10} + (-120)_{10}$

1. $(+95)_{10} + (-63)_{10}$.

In signed binary number, the left most bit is the sign bit and the remaining bits are the magnitude bits.

$$\begin{array}{lll}
 (+63)_{10} = & \mathbf{0} & 0111\ 111 \\
 (-63)_{10} = & \underbrace{\mathbf{1}}_{\text{sign}} & \underbrace{01111111}_{\text{magnitude}}
 \end{array}$$

1's complement of $(-63)_{10} = 1\ 1000000$

2's complement of $(-63)_{10} = 1\ 1000001$

1's complement arithmetic

$$(+95)_{10} + (-63)_{10}$$

$$\begin{array}{rcl} (+95)_{10} & \Rightarrow & 0101\ 1111 \\ (-63)_{10} & \Rightarrow & 1100\ 0000 \\ & & \boxed{1}0001\ 1111 \\ & & \quad +\ 1 \\ & & \hline 0010\ 0000 = +32 \end{array}$$

2's complement arithmetic

$$(+95) + (-63)$$

$$\begin{array}{rcl} 0101\ 1111 \\ 1100\ 0001 \\ \hline 0010\ 0000 = +32 \end{array}$$

$$2. \ (+42)_{10} + (-87)_{10}$$

$$\begin{array}{l} (+87)_{10} = 0\ 1010111 \\ (-87)_{10} = 1\ 1010111 \end{array}$$

$$1\text{'s complement of } (-)_{10} = 1\ 0101000$$

$$2\text{'s complement of } (-7)_{10} = 1\ 0101001$$

1's complement arithmetic

$$(+42)_{10} + (-87)_{10}$$

$$\begin{array}{rcl} 0010\ 1010 \\ 1010\ 1000 \\ \hline 1101\ 0010 \end{array}$$

$$1\text{'s complement of result is } 1\ 0101101 = -45$$

2's complement arithmetic

$$\begin{array}{rcl} 00101010 \\ 10101001 \\ \hline 11010011 = -45 \end{array}$$

Since the two's complement of 1101 0011 is $(0010\ 1100 + 1 = 0010\ 1101)$ $(+45)_{10}$, 1101 0011 is $(-45)_{10}$.

$$3. \ (-13)_{10} + (-59)_{10},$$

$$\begin{array}{lll} (+13) \Rightarrow & 0 & 000\ 1101 \\ (-13) \Rightarrow & 1 & 000\ 1101 \\ (+59) \Rightarrow & 0 & 0111\ 011 \\ (-59) \Rightarrow & 1 & 0111011 \end{array}$$

1's complement arithmetic

$$(-13)_{10} + (-59)_{10}$$

Sign

$$\begin{array}{rcl} \text{1's complement of } (-13)_{10} = & \begin{array}{r} 1 \\ 111\ 0010 \end{array} \\ \text{1's complement of } (-59)_{10} = & \begin{array}{r} 1 \\ 1000100 \\ \hline 1 \end{array} \\ & \begin{array}{r} 1 \\ 0110110 \\ \hline +1 \\ \hline 1 \end{array} \\ & \underline{01100111} \end{array}$$

1's complement of result is 1 1001000 = -72

2's complement arithmetic

$$\text{2's complement of } (-13) = \begin{array}{r} 1 \\ 111\ 0011 \end{array}$$

$$\text{2's complement of } (-59) = \begin{array}{r} 1 \\ 100\ 0101 \\ \hline 1 \end{array}$$

$$\underline{011\ 1000} = -72$$

Since 2's complement of result is

$$01001000 = +72, 10111000 = -72.$$

4. $(+38)_{10} + (-38)_{10}$

$$(+38) = \begin{array}{r} 0 \\ 0100110 \end{array}$$

$$(-38) = \begin{array}{r} 1 \\ 0100110 \end{array}$$

$$\text{1's complement of } (-38) = \begin{array}{r} 1 \\ 1011001 \end{array}$$

$$\text{2's complement of } (-38) = \begin{array}{r} 1 \\ 1011010 \end{array}$$

1's complement arithmetic

$$\begin{array}{r} 0 \\ 0100110 \\ \hline 1 \\ 1011001 \\ \hline 1 \\ 1111111 \end{array}$$

$$\text{1's complement of result is } 1\ 0000000 = -0$$

2's complement arithmetic

$$\begin{array}{r} 0 \\ 0100110 \\ \hline 1 \\ 1011010 \\ \hline 0 \\ 0000000 = 0 \end{array}$$

5. $(-105)_{10} + (-120)_{10}$

$$(+105) = \begin{array}{r} 0 \\ 1101001 \end{array}$$

$$(-105) = \begin{array}{r} 1 \\ 1101001 \end{array}$$

$$(+120) = \begin{array}{r} 0 \\ 1111000 \end{array}$$

$$(-120) = \begin{array}{r} 1 \\ 1111000 \end{array}$$

1's complement arithmetic

$$(-105) + (-120)$$

$$\text{1's complement of } (-105) = \begin{array}{r} 1 \\ 0010110 \end{array}$$

$$\text{1's complement of } (-120) = \begin{array}{r} 1 \\ 0000111 \end{array}$$

$$\begin{array}{r} 1 \\ 0 \\ \hline 0011101 \\ +1 \\ \hline 01001110 \end{array} = 30$$

2's complement arithmetic

$$\text{2's complement of } (-105) = \begin{array}{r} 1 \\ 0010111 \end{array}$$

$$\text{2's complement of } (-120) = \begin{array}{r} 1 \\ 0001000 \end{array}$$

$$\begin{array}{r} 0 \\ 0011111 \\ \hline 0011111 = 31 \end{array}$$

Error occurs due to overflow.

1.10 OTHER NUMBER SYSTEMS

Table 1.3 shows the some positional number system and their possible symbols.

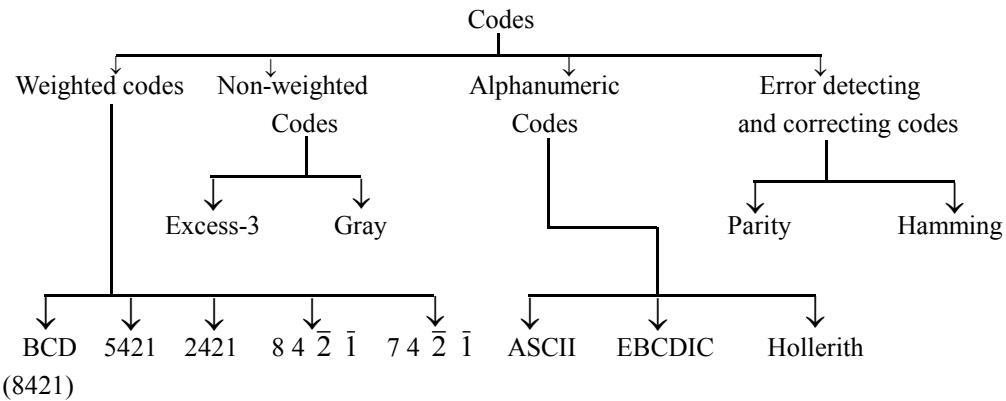
Table 1.3: Number Systems

Number System	Base	Possible Symbols
Binary	2	0, 1
Ternary	3	0, 1, 2
Quarternary	4	0, 1, 2, 3
Quinary	5	0, 1, 2, 3, 4
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Duodecimal	12	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

1.11 BINARY CODES

The digital data is represented, stored and transmitted as groups of binary bits. The group of bits is called as binary code. The binary code represent numbers, alphabets, special characters and control functions. The codes are classified as,

- ◆ Weighted codes
- ◆ Non-weighted codes
- ◆ Error detecting and correcting codes
- ◆ Alphanumeric codes



1.11.1 Weighted Codes

Weighted binary codes obey their positional weighting principles. Each digit position of a number represents a specific weight. The bits are multiplied by the weights and the sum of these weighted bits give the equivalent decimal value.

8421 code is the binary-coded-decimal (BCD) code. The other 4 bit weighted binary codes are: 5421, 2421, 8 4 $\bar{2}$ $\bar{1}$, 74 $\bar{2}$ $\bar{1}$ etc. The examples for 5 bit code are 84621, 51111 and 63210 code. The example for 7 bit code is biquinary (5043210) code.

1. BCD (8421) Code

The binary-coded-decimal (BCD) uses the binary number system to specify the decimal numbers 0 to 9. It has 4 bits. It is called 8 – 4 – 2 – 1 code, i.e., bit 3 has weight $8(2^3)$, bit 2 has weight $4(2^2)$, bit 1 has weight $2(2^1)$ and bit 0 has weight $1(2^0)$. **Table 1.4** shows the 8421 BCD code used to represent the decimal digits.

Table 1.4: BCD Code

<i>Decimal Digit</i>	<i>BCD Code</i>
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
10	0001 0000
11	0001 0001
12	0001 0010

2. Other 4 bit codes: Table 1.5 shows the 4 bit binary codes for the decimal digits 0 to 9.

Table 1.5: Binary Codes

Decimal	5 4 2 1	2 4 2 1	8 4 $\bar{2} \bar{1}$	7 4 $\bar{2} \bar{1}$
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1	0 1 1 1	0 1 1 1
2	0 0 1 0	0 0 1 0	0 1 1 0	0 1 1 0
3	0 0 1 1	0 0 1 1	0 1 0 1	0 1 0 1
4	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0
5	1 0 0 0	1 0 1 1	1 0 1 1	1 0 1 0
6	1 0 0 1	1 1 0 0	1 0 1 0	1 0 0 1
7	1 0 1 0	1 1 0 1	1 0 0 1	1 0 0 0
8	1 0 1 1	1 1 1 0	1 0 0 0	1 1 1 1
9	1 1 0 0	1 1 1 1	1 1 1 1	1 1 1 0

- Representation of decimal digit 7 in 5421 code:

$$1010 \Rightarrow (1 \times 5) + (0 \times 4) + (1 \times 2) + (0 \times 1) = 7$$

- Representation of decimal digit 5 in 2421 code:

$$1011 \Rightarrow (1 \times 2) + (0 \times 4) + (1 \times 2) + (1 \times 1) = 5$$

- Representation of decimal digit 2 in 8 4 $\bar{2} \bar{1}$ code:

$$0110 \Rightarrow (0 \times 8) + (1 \times 4) + (1 \times -2) + (0 \times -1) = 4 - 2 = 5$$

- Representation of decimal digit 1 in 7 4 $\bar{2} \bar{1}$ code:

$$0111 \Rightarrow (0 \times 7) + (1 \times 4) + (1 \times -2) + (1 \times -1) = 4 - 3 = 1$$

The some other weighted 4 bit binary codes are:

3 3 2 1 Code

4 2 2 1 Code

5 2 2 1 Code

5 3 1 1 Code

6 3 1 1 Code

7 4 2 1 Code

1.11.2 Non Weighted Codes

Non-weighted codes are codes that are not positionally weighted, i.e, each position within a binary number is not assigned a fixed value. The non-weighted codes are:

- Excess-3 (XS – 3) Code
 - Gray Code
- (i) **Excess-3 Codes:** The excess-3 code is a modified form of binary number. It is obtained by simply adding ‘3’ to a decimal number. For example, to encode the decimal number ‘4’ into an excess 3 code, first add 3. i.e., $4 + 3 = 7$. Then it is encoded into binary code 0111. Table 1.6

shows excess-3 codes to represent decimal numbers. XS – 3 code is also called as reflective code.

Table 1.6: Excess-3 Codes

Decimal	BCD Code	XS – 3 Code
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100
10	0001 0000	0100 0011
11	0001 0001	0100 0100
12	0001 0010	0100 0101

- (ii) **Gray Codes:** In gray code, numbers differ from any adjacent number by a single bit. For example, in going from decimal 3 to 4, the Gray-code number changes from 0010 to 0110. Every number differs by only 1 bit from the preceding number. It is also called as unit-distance code or reflected code. **Table 1.7** shows the gray code representation.

Table 1.7: Gray Codes

Decimal	Binary Code	Gray Code
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
10	1 0 1 0	1 1 1 1
11	1 0 1 1	1 1 1 0
12	1 1 0 0	1 0 1 0
13	1 1 0 1	1 0 1 1
14	1 1 1 0	1 0 0 1
15	1 1 1 1	1 0 0 0

(iii) **Binary-to-Gray Code Conversion:** The binary-to-gray code conversion is achieved using following rules:

- The most significant bit (MSB) in the Gray code is the same as the corresponding MSB in the binary number.
- Going from left to right, perform an EX-OR operation between the adjacent pair of binary code bits to get the next Gray code bit.

Example 1.76: Convert the binary number 1001 to Gray code,

$$\begin{array}{ccccccc} \text{Binary : } & 1 & \xrightarrow{\oplus} & 0 & \xrightarrow{\oplus} & 0 & \xrightarrow{\oplus} 1 \\ & \downarrow & & \downarrow & & \downarrow & \\ \text{Gray : } & 1 & & 1 & & 0 & \\ \text{Gray Code : } & 1101 & & & & & \end{array}$$

Example 1.77: Convert the binary $(10110)_2$ to its gray code.

$$\begin{array}{ccccccc} \text{Binary : } & 1 & \xrightarrow{\oplus} & 0 & \xrightarrow{\oplus} & 1 & \xrightarrow{\oplus} 1 \xrightarrow{\oplus} 0 \\ & \downarrow & & \downarrow & & \downarrow & \downarrow \\ \text{Gray : } & 1 & & 1 & & 1 & 0 \\ \text{Gray Code : } & 11101 & & & & & \end{array}$$

Let the binary number is represented as $B_1, B_2, B_3, \dots, B_N$ and gray code is $G_1, G_2, G_3, \dots, G_N$ then

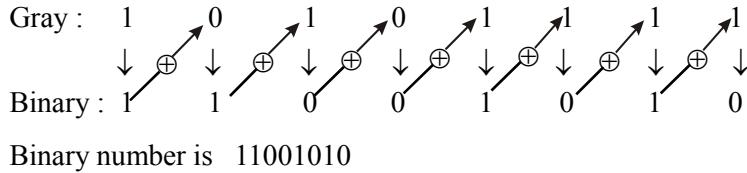
$$\begin{aligned} G_1 &= B_1 \\ G_2 &= B_1 \oplus B_2 \\ G_3 &= B_2 \oplus B_3 \\ &\vdots \\ G_N &= B_{N-1} \oplus B_N \end{aligned}$$

(iv) **Gray-to-Binary Code Conversion:** The gray-to-binary code conversion is achieved using following rules:

- The MSB in the binary code is the same as the corresponding bit in the gray code.
- To obtain the next binary digit, perform an EX-OR operation between the bit just written down and the next gray code bit.

Example 1.78: Convert the Gray code 1000 to binary

$$\begin{array}{ccccccc} \text{Gray : } & 1 & & 0 & & 0 & & 0 \\ & \downarrow & & \downarrow & & \downarrow & & \\ \text{Binary : } & 1 & \nearrow \oplus & 1 & \nearrow \oplus & 1 & \nearrow \oplus & 1 \\ & & & & & & & \\ \text{Binary Number : } & 1 & 1 & 1 & 1 & & & \end{array}$$

Example 1.79: Convert the gray code 10101111 to binary**1.12 ERROR DETECTION CODES****1.12.1 Parity Bit**

In any electronic system involving the transfer of data (in the form of binary digits) then data transmission errors are possible. Any external noise introduced in the physical communication medium may change some of the bits from 0 to 1 or 1 to 0. The purpose of an error-detection code is to detect such bit-reversal errors. The method of **Parity** is widely used as a method of error detection.

A parity bit is an extra bit included with a message to make the total number of 1's transmitted either odd or even.

Even Parity: The value of the parity bit is set such that the total number of 1's in the data word is **even**.

Example : 11001 which has an odd number of 1's. The new total group is thus 110011.

11110 which has an even number of 1's. The new total group is thus 111100.

Odd Parity: The value of the parity bit is set such that the total number of 1's in the data word is **odd**.

Example: 11001 which has an odd number of 1's. The new total group is thus 110010.

11110 which has an even number of 1's. The new total group is thus 111101.

A message of 4 bits and a parity bit are shown in **Table 1.8**.

Error-Detection

Let an even parity bit is generated in the sending end for each message transmission. The message, together with the parity bit is transmitted to its destination. The parity of the received data is checked in the receiving end. If the parity of the received information is not even, it means that atleast one bit has changed value during the transmission. This method detects 1, 3 or any odd combination of errors in each message that is transmitted. An even combination of errors is undetected.

1.12.2 2-out of-5 Code

2-out of-5 code is used for error detection in communications work. It utilizes five bits to represent the ten decimal digits, so it is a form of BCD code. Each code word has exactly two 1's, a convention that facilitates decoding and provides for better error detection than the single-parity-bit method. If more or less than two 1's appear, an error is detected.

Table 1.8: Parity Bit

ODD Parity		EVEN Parity	
Message	P	Message	P
0000	1	0000	0
0001	0	0001	1
0010	0	0010	1
0011	1	0011	0
0100	0	0100	1
0101	1	0101	0
0110	1	0110	0
0111	0	0111	1
1000	0	1000	1
1001	1	1001	0
1010	1	1010	0
1011	0	1011	1
1100	1	1100	0
1101	0	1101	1
1110	0	1110	1
1111	1	1111	0

1.12.3 63210 Code

This is a BCD code. Like 2-out of-5 code, it is also characterized by having exactly two 1's in each 5 bit groups.

1.12.4 50 - 43210 Code

This code is also called as **Biquinary** (two – five) code. It is used in counters and is composed of a 2 bit group and a 5 bit group, each with a single 1. Its weights are 50 – 43210. The 2 bit group, having weights of five and zero, indicates whether the number represented is less than, equal to, or greater than 5. The 5 bit group indicates the count above or below 5.

During transmission of signals from one location to another, an error may occur. One or more bits may change value. A circuit in the receiving side can detect the presence of more or less than two 1's and if the received combination of bits does not agree with the allowable combination, an error is detected.

1.12.5 Ring Counter Code

This code has ten bits, one for each decimal digit, and a single 1 makes error detection possible. It is easy to decode but wastes bit and requires more circuitry to implement than the 4 bit or 5 bit codes. Its weights are 9876543210. It is used in shift registers and ring counters.

Table 1.9 shows these error detection codes.

Table 1.9: Error Detection Codes

Decimal	2-out of-5	63210	50 – 43210	9876543210
0	00011	00110	01 – 00001	0000000001
1	00101	00011	01 – 00010	0000000010
2	00110	00101	01 – 00100	0000000100
3	01001	01001	01 – 01000	0000001000
4	01010	01010	01 – 10000	0000010000
5	01100	01100	10 – 00001	0000100000
6	10001	10001	10 – 00010	0001000000
7	10010	10010	10 – 00100	0010000000
8	10100	10100	10 – 01000	0100000000
9	11000	11000	10 – 10000	1000000000

1.13 ALPHANUMERIC CODES

Alphanumeric codes are used to represent numbers, letters and other special features using binary bits. The alphanumeric codes are encoding the following:

- Decimal digits (0 – 9)
 - Alphabetic characters (A to Z and a to z)
 - Mathematical symbols (like, +, -, =, <, >)
 - Special control characters (like, ESC, NUL, ACK)
- The alphanumeric codes are,
- ♦ ASCII Code,
 - ♦ EBCDIC Code,
 - ♦ Hollerith Code.

1.13.1 ASCII Code

ASCII [American Standard Code for Information Interchange] is a 7 bit code, which is used to represent numbers, letters, characters and other special computer control functions. ASCII is pronounced as “as-kee”.

ASCII is 7 bit code, therefore it has $2^7 = 128$ characters.

Upper case alphabets	=	26
Lower case alphabets	=	26
Decimal digits (0 – 9)	=	10
Special symbols	=	33
Control characters	=	33
Total	=	128

The ASCII characters, ASCII values in binary, decimal and hexadecimal are shown in **Appendix 1**.

1.13.2 EBCDIC Code

EBCDIC [Extended Binary Coded Decimal Interchange Code] is an 8-bit code, pronounced as “eb-see-disk”. It differs from ASCII code only in its code grouping for the different alphanumeric characters.

EBCDIC has $2^8 = 256$ characters, in which 117 are unassigned. The remaining 139 characters are as follows:

Upper case alphabets	= 26
Lower case alphabets	= 26
Decimal digits (0 – 9)	= 10
Special symbols	= 27
Control characters	<u>= 50</u>
Total	<u>= 139</u>

The control characters are placed in between 0000 0000 and 0011 1111 and the alphanumeric codes and symbols are placed in between 0100 0000 and 1111 1111.

1.13.3 Hollerith Code

The Hollerith Code is the alphanumeric code used in punched cards. Each card has 80 columns and 12 rows. The rows are numbered 0 through 9, 11 and 12. The columns are numbered 1 through 80, each one containing one character. Each character is uniquely identified by the rows punched in that column.

Thus far, we can express a decimal 8 as follows:

Decimal	8
Binary	1000
Octal	10
Hexadecimal	8
BCD	1000
XS 3	1011
Gray	1100
Biquinary	10 01000
ASCII	0011 1000
EBCDIC	1111 1000
Hollerith	8

Example 1.80:

Consider a decimal number 14 and write the equivalent for this in Binary, BCD, 2421, Gray and Excess 3 codes. **(Dec. 2005)**

Solution:

Binary	1110
BCD	0001 0100
2421	0001 0100
Gray	1001
XS 3	0001 0111

1.14 BOOLEAN POSTULATES AND LAWS

1.14.1 Introduction

In 1854 George Boole introduced a systematic treatment of logic and developed for this purpose an algebraic system now called Boolean Algebra.

In 1938 C.E. Shannon introduced a two-valued Boolean Algebra called Switching Algebra, in which he demonstrated that the properties of bistable electrical switching circuits can be represented by this algebra.

For the formal definition of Boolean Algebra, we shall employ the postulates formulated by E.V. Huntington in 1904.

1.14.2 Postulates or Axioms

The Postulates or Axioms of Boolean Algebra are a set of logical expressions that we accept without proof and upon which we can build a set of useful theorem. Actually the axioms are nothing more than the definitions of 3 basic logic operations: AND, OR and INVERT.

$$\text{Postulate } 1 \rightarrow 0 \cdot 0 = 0$$

$$\text{Postulate } 6 \rightarrow 0 + 1 = 1$$

$$\text{Postulate } 2 \rightarrow 0 \cdot 1 = 0$$

$$\text{Postulate } 7 \rightarrow 1 + 0 = 1$$

$$\text{Postulate } 3 \rightarrow 1 \cdot 0 = 0$$

$$\text{Postulate } 8 \rightarrow 1 + 1 = 1$$

$$\text{Postulate } 4 \rightarrow 1 \cdot 1 = 1$$

$$\text{Postulate } 9 \rightarrow \bar{1} = 0$$

$$\text{Postulate } 5 \rightarrow 0 + 0 = 0$$

$$\text{Postulate } 10 \rightarrow \bar{0} = 1$$

1.14.3 Theorems of Boolean Algebra

- ◆ The Commutative Properties
- ◆ The Associative Properties
- ◆ The Idempotent Properties
- ◆ The Identity Properties
- ◆ The Null Properties
- ◆ The Distributive Properties
- ◆ The Negation Properties

- ◆ The Double Negation Properties
- ◆ The Absorption Properties and
De Morgan's Theorems

(i) Commutative Properties

Theorem 1a : $AB = BA$

1b : $A + B = B + A$

Theorem 1a: $AB = BA$

“A AND B” is the same as ‘B AND A’ ; in effect, it makes no difference which input of an AND gate is connected to A and which is connected to B. The truth tables are identical.

A	B	AB	=	B	A	BA
0	0	0		0	0	0
0	1	0		0	1	0
1	0	0		1	0	0
1	1	1		1	1	1

Table 1.10(a)

Theorem 1b : $A + B = B + A$

“A OR B” is same as “B OR A”

A	B	$A + B$	=	B	A	$B + A$
0	0	0		0	0	0
0	1	1		0	1	1
1	0	1		1	0	1
1	1	1		1	1	1

Table 1.10(b)

The commulative properties can be extended to any number of variables:

$$A + B + C = B + A + C \quad ABCD = BACD$$

$$A + C = C + A \quad BADC = ABDC$$

$$B + A + C = B + C + A$$

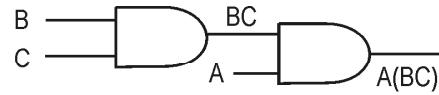
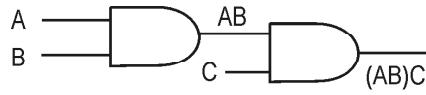
(ii) The Associative Properties

Theorem 2a: $(AB)C = A(BC)$

2b: $(A + B) + C = A + (B + C)$

Theorem 2a : $(AB)C = A(BC)$

A AND B ANDed with C is same as A ANDed with B AND C



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

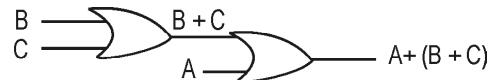
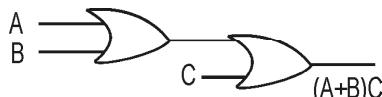
A	B	C	BC	$A(BC)$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Fig. 1.8: Associative Law

Note that $(AB)C = A(BC) = ABC$

Theorem 2b: $(A + B) + C = A + (B + C)$

A OR B ORed with C is same as A ORed with B OR C



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	0	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

Fig. 1.9: Associative Law

Note that, $(A + B) + C = A + (B + C) = A + B + C$

The associative properties can be extended to any number variables:

$$A(BCD) = (AB)(CD) = (ABC)D = ABCD$$

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

(iii) The Idempotent Properties

Theorem 3a : $AA = A$

Theorem 3b : $A + A = A$

$$AA = A$$



$$A + A = A$$



If $A = 0$, then $AA = 0 \cdot 0 = 0 = A$

If $A = 1$, then $AA = 1 \cdot 1 = 1 = A$

If $A = 0$, then $A + A = 0 + 0 = 0 = A$

If $A = 1$, then $A + A = 1 + 1 = 1 = A$

Note: $AAA = A$

$$A + A + A = A$$

(iv) Identity Properties

Theorem 4a : $A \cdot 1 = A$

Theorem 4b : $A + 1 = 1$

$$A \cdot 1 = A$$



$$A + 1 = 1$$



If $A = 0$, then $A \cdot 1 = 0 \cdot 1 = 0 = A$

If $A = 1$, then $A \cdot 1 = 1 \cdot 1 = 1 = A$

If $A = 0$, then $A + 1 = 0 + 1 = 1 = 1$

If $A = 1$, then $A + 1 = 1 + 1 = 1 = 1$

Note: $1 \cdot A = A$

$$1 + A = 1$$

$$(AB + C) \cdot 1 = AB + C$$

$$AB + CD + 1 = 1$$

(v) The Null Properties**Theorem 5a:** $A \cdot 0 = 0$ **Theorem 5b:** $A + 0 = A$

$$A \cdot 0 = 0$$

If $A = 0$, then $A \cdot 0 = 0 \cdot 0 = 0$ If $A = 1$, then $A \cdot 0 = 1 \cdot 0 = 0$ 

$$A + 0 = A$$

If $A = 0$, then $A + 0 = 0 + 0 = 0 = A$ If $A = 1$, then $A + 0 = 1 + 0 = 1 = A$ **Note:**

$$(AB+CD) \cdot 0 = 0$$

$$ABC + D + 0 = ABC + D$$

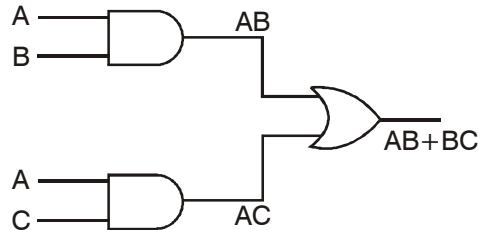
(vi) The Distributive Property**Theorem 6:** $A(B+C) = AB + AC$ 

Fig. 1.10: Distributive Law

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

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

A	B	C	AB	AC	$AB + AC$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	1	1

$$AB + AC = AB + AC$$

$$AB(C+DE) = ABC + ABD E$$

$$\begin{aligned}(A+B)(C+D) &= (A+B)C + (A+B)D \\ &= AC + BC + AD + BD\end{aligned}$$

Note: The distributive property is often used in reverse called as **factoring**.

$$AB + AC = A(B+C)$$

$$XYW + XYZ = XY(W+Z)$$

(vii) The Negative Properties

Theorem 7a : $A\bar{A} = 0$

Theorem 7b : $A + \bar{A} = 1$



$$\text{If } A=0, 0 \cdot 1 = 0$$

$$\text{If } A=1, 1 \cdot 0 = 0$$



$$\text{If } A=0, 0 + 1 = 1$$

$$\text{If } A=1, 1 + 0 = 1$$

Note: $ABC + AB\bar{C} = 1$

(viii) The Double Negation Property

Theorem 8 : $\overline{\overline{A}} = A$



$$\text{If } A=0, \overline{A}=1, \overline{\overline{A}}=0=A$$

$$\text{If } A=1, \overline{A}=0, \overline{\overline{A}}=1=A$$

Note: $A + BC = A + \overline{B}\overline{C}$

(ix) The Absorption Properties

Theorem 9a: $A + AB = A$

Theorem 9b: $A(A + B) = A$

Theorem 9c: $A + \overline{A}B = A + B$

9a) $A + AB = A$

$$A(1+B) = A \cdot 1 = A$$

Example

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

9b) $A(A + B) = A$

$$AA + AB = A + AB = A$$

$$XY(X\bar{Y} + WYZ) = X\bar{Y}$$

9c) $A + \bar{A}B = A + B$

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

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

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

$$X + Y + (\bar{X} + \bar{Y})Z$$

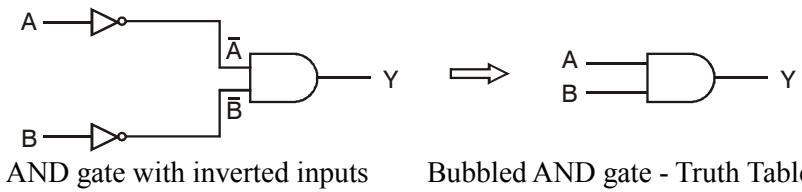
$$= A + (1 \cdot B)$$

$$= X + Y + Z$$

$$= A + B$$

$$= X + Y + Z$$

Bubbled AND Gate

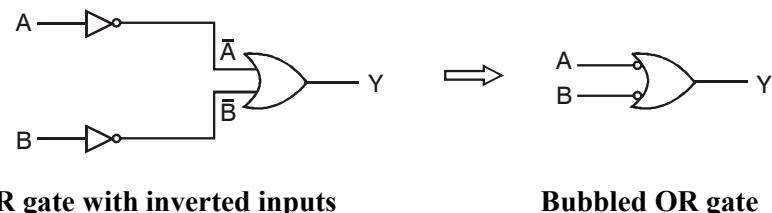


$$Y = \bar{A}\bar{B}$$

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

Fig. 1.11: Bubbled AND gate

Bubbled OR Gate



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

Bubbled OR gate – Truth Table

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

Fig. 1.12 : Bubbled OR gate

1.15 DE MORGAN'S THEOREMS

Theorem (1) : $\overline{AB} = \overline{A} + \overline{B}$

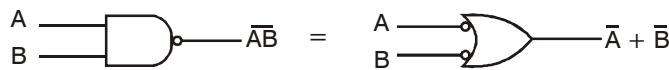
Theorem (2) : $\overline{A+B} = \overline{A}\overline{B}$

De Morgan's First Theorem

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

“The complement of a product equals the sum of the complements.”

A NAND gate performs the same operation as a bubbled OR gate.



NAND

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

Bubbled OR

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

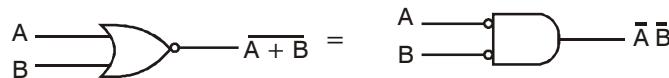
Fig. 1.13: De Morgan's Theorem 1

De Morgan's Second Theorem

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

“The complement of sum equals the product of the complements”.

A NOR gate performs the same operation as Bubbled AND gate



NOR

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

Bubbled AND

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

Fig. 1.14: De Morgan's Theorem 2

Note: $A \cdot B \neq \overline{A} \cdot \overline{B}$; $A + B \neq \overline{A} + \overline{B}$

De Morgan's two theorems can be regarded as a single theorem by observing the following rule:

“Change the logic operation covered by the inversion bar, remove the inversion bar, and complement each variable that was originally covered by the bar.”

$$(i) \quad \overline{ABC} = \overline{A} + \overline{B} + \overline{C} \text{ and } \overline{A+B+C} = \overline{A} \cdot \overline{B} \cdot \overline{C}$$

$$(ii) \quad \overline{\overline{AB}} = \overline{\overline{A}} + \overline{\overline{B}} = A + B$$

$$(iii) \quad \overline{A \cdot \overline{B} \cdot \overline{C}} = \overline{A} + \overline{\overline{B}} + \overline{\overline{C}} = \overline{A} + B + C$$

$$(iv) \quad \overline{A \cdot \overline{B} + C} = (\overline{A} + \overline{\overline{B}}) \cdot \overline{C} = (\overline{A} + B) \cdot \overline{C}$$

$$(v) \quad \overline{(A + \overline{B}) \cdot \overline{CD} + E} = \overline{[(\overline{A} + \overline{\overline{B}}) \cdot \overline{CD}]} \cdot \overline{E}$$

$$= \overline{[(\overline{A} + \overline{\overline{B}}) + \overline{\overline{CD}}]} \cdot \overline{E}$$

$$= \overline{[(A + \overline{B}) + \overline{\overline{C}} + \overline{D}]} \cdot \overline{E}$$

$$= \overline{[A + \overline{B} + C + \overline{D}]} \cdot \overline{E}$$

$$(vi) \quad \overline{ABC + \overline{BC}} = (\overline{ABC}) \cdot \overline{\overline{BC}}$$

$$= (\overline{ABC}) (BC)$$

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

$$= \overline{ABC} + \overline{BBC} + BCC\overline{C}$$

$$= \overline{ABC} + 0 + 0$$

$$= \overline{ABC}$$

The Boolean theorems are given in **Table 1.11**.

TABLE 1.11: Boolean Theorems

1. Commutative	1a	$AB = BA$	1b	$A + B = B + A$
2. Associative	2a	$(AB)C = A(BC)$	2b	$(A + B) + C = A + (B + C)$
3. Idempotent	3a	$AA = A$	3b	$A + A = A$
4. Identity	4a	$A \cdot 1 = A$	4b	$A + 1 = 1$
5. Null	5a	$A \cdot 0 = 0$	5b	$A + 0 = A$
6. Distributive	6	$A(B + C) = AB + AC$		
7. Negation	7a	$A\bar{A} = 0$	7b	$A + \bar{A} = 1$
8. Double Negation	8	$\bar{\bar{A}} = A$		
9. Absorption	9a	$A + AB = A$	9b	$A(A + B) = A$
	9c	$A + \bar{A}B = A + B$		
10. De Morgan		$\bar{AB} = \bar{A} + \bar{B}$		$\bar{A+B} = \bar{A}\bar{B}$

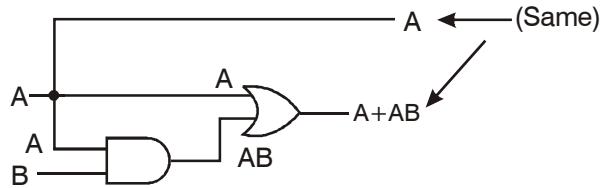
1.16 BOOLEAN RULES FOR SIMPLIFICATION

Boolean algebra finds its most practical use in the simplification of logic circuits. To translate a logic circuit's function into symbolic (Boolean) form, and apply certain algebraic rules to the resulting equation to reduce the number of terms and/or arithmetic operations, the simplified equation may be translated back into circuit form a logic circuit performing the same function with fewer components. If equivalent function may be achieved with fewer component, the result will be increased reliability and decreased cost of manufacture.

To this end, there are several rules of Boolean algebra presented in this section for use in reducing expressions to their simplest forms. The identities and properties already reviewed in this chapter are very useful in Boolean simplification, and for the most part bear similarity to many identities and properties of "normal" algebra. However, the rules shown in this section are all unique to Boolean mathematics.

Rule 1:

$$A + AB = A$$

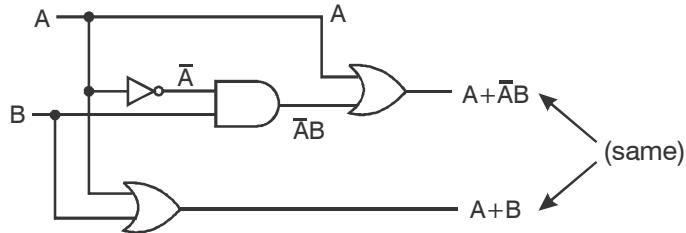
Fig. 1.15: Logic diagram for $A + AB = A$

This rule may be proven symbolically by factoring an “ A ” out of the two terms, then applying the rules of $A + 1 = 1$ and $1A = A$ to achieve the final result:

$$\begin{array}{c}
 A + AB \\
 \downarrow \\
 A(1 + B) \\
 \downarrow \\
 A(1) \\
 \downarrow \\
 A
 \end{array}
 \quad \begin{array}{l}
 \text{Factoring } A \text{ out of both terms} \\
 \text{Applying identity } B + 1 = 1 \\
 \text{Applying identity } 1A = A
 \end{array}$$

Rule 2: (Dec. 2005)

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

Fig. 1.16: Logic diagram for $A + \bar{A}B = A + B$

$$\begin{array}{l}
 A + \bar{A}B \\
 \downarrow \\
 A + AB + \bar{A}B \\
 \downarrow \\
 A + B(A + \bar{A}) \\
 \downarrow \\
 A + B(1) \\
 \downarrow \\
 A + B
 \end{array}
 \quad \begin{array}{l}
 \text{Applying the previous rule to expand } A \text{ term } A + AB = A \\
 \text{Factoring B out of 2nd and 3rd terms} \\
 \text{Applying negation } A + \bar{A} = 1 \\
 \text{Applying identity } 1B = B
 \end{array}$$

Rule 3:

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

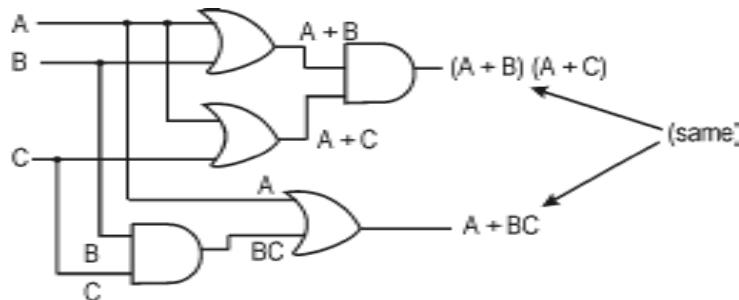


Fig. 1.17 : Logic diagram $(A + B)(A + C) = A + BC$

And, the corresponding proof:

$$\begin{array}{l}
 (A + B)(A + C) \\
 \downarrow \\
 AA + AC + AB + BC \\
 \downarrow \\
 A + AC + AB + BC \\
 \downarrow \\
 A + AB + BC \\
 \downarrow \\
 A + BC
 \end{array}
 \quad \begin{array}{l}
 \text{Distributing terms} \\
 \text{Applying identity } AA = A \\
 \text{Applying rule } A + AB = A \text{ to the } A + AC \text{ term} \\
 \text{Applying rule } A + AB = A \text{ to the } A + AB \text{ term}
 \end{array}$$

Example 1.81: Simplify the expression $AB + BC(B + C)$

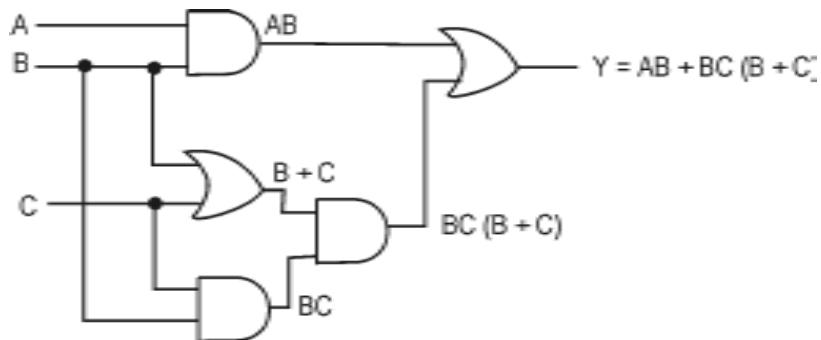


Fig. 1.18: Logic diagram

$$\begin{array}{c}
 AB + BC(B + C) \\
 \downarrow \qquad \qquad \qquad \text{Distributing terms} \\
 AB + BBC + BCC \\
 \downarrow \qquad \qquad \qquad \text{Applying identity } AA = A \text{ to 2nd and 3rd terms} \\
 AB + BC + BC \\
 \downarrow \qquad \qquad \qquad \text{Applying identity } A + A = A \text{ to 2nd and 3rd terms} \\
 AB + BC \\
 \downarrow \qquad \qquad \qquad \text{Factoring } B \text{ out of terms} \\
 B(A + C)
 \end{array}$$

The final expression, $B(A + C)$, is much simpler than the original, yet performs the same function.

The truth tables for these two expressions should be identical.

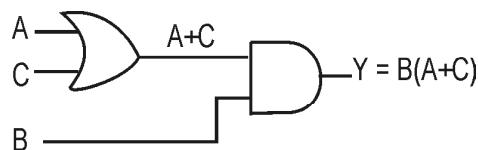


Fig. 1.19 : Simplified function diagram

1.17 MINIMIZATION OF BOOLEAN EXPRESSIONS

Example 1.82:
$$\begin{aligned} W &= ABC + CAB + BAC \\ &= ABC + ABC + ABC && \text{(Theorem 1a)} \\ &= ABC + ABC && \text{(Theorem 3b)} \\ &= ABC \end{aligned}$$

Example 1.83:
$$\begin{aligned} X &= ABC + A B \bar{C} \\ &= AB(C + \bar{C}) && \text{(Theorem 6)} \\ &= AB \cdot 1 && \text{(Theorem 7b)} \\ &= AB && \text{(Theorem 4a)} \end{aligned}$$

Example 1.84:
$$\begin{aligned} W &= X(\bar{X}YZ + \bar{X}Y\bar{Z}) \\ &= X\bar{X}YZ + X\bar{X}Y\bar{Z} \\ &= 0 \cdot YZ + 0 \cdot Y\bar{Z} \\ &= 0 + 0 \\ &= 0 \end{aligned}$$

Example 1.85:
$$\begin{aligned} Z &= A\bar{B}\bar{C} + A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C \\ &= A\bar{B}(\bar{C} + C) + \bar{A}B C + \bar{A}\bar{B}C && \text{(Theorem 6)} \\ &= A\bar{B} \cdot 1 + \bar{A}C(B + \bar{B}) && \text{(Theorem 7b)} \\ &= A\bar{B} \cdot 1 + \bar{A}C \cdot 1 \\ &= A\bar{B} + \bar{A}C && \text{(Theorem 4a)} \end{aligned}$$

Example 1.86:
$$\begin{aligned} W &= (X_1 + X_2)(\bar{X}_1 + X_1 X_2) + (\bar{X}_2 + X_1 \bar{X}_2) \\ &= X_1 \bar{X}_1 + X_1 X_1 X_2 + X_2 \bar{X}_1 + X_2 X_1 X_2 + \bar{X}_2 + X_1 \bar{X}_2 && \text{(Theorem 6)} \\ &= 0 + X_1 X_2 + X_2 \bar{X}_1 + X_1 X_1 + \bar{X}_2 + X_1 \bar{X}_2 && \text{(Theorem 7a, 3a)} \\ &= X_2 X_1 + X_2 \bar{X}_1 + X_1 \bar{X}_2 + X_1 X_2 + \bar{X}_2 \\ &= X_2(X_1 + \bar{X}_1) + X_1(X_2 + \bar{X}_2) + \bar{X}_2 \\ &= X_2 \cdot 1 + X_1 \cdot 1 + \bar{X}_2 && \text{(Theorem 7b)} \\ &= X_2 + X_1 + \bar{X}_2 \end{aligned}$$

$$= X_1 + (X_2 + \bar{X}_2) \quad (\text{Theorem 7b})$$

$$= X_1 + 1 \quad (\text{Theorem 4b})$$

$$= 1$$

Example 1.87: $A = WXY(W\bar{X} + W\bar{Y}) + W\bar{X}Y(\bar{W}\bar{X} + X\bar{Y})$

$$\begin{aligned} &= WXYW\bar{X} + WXYW\bar{Y} + W\bar{X}Y\bar{W}\bar{X} + W\bar{X}YX\bar{Y} \\ &= WYWX\bar{X} + WXWY\bar{Y} + W\bar{W}\bar{X}Y\bar{X} + W\bar{X}XY\bar{Y} \quad (\text{Theorem 7a}) \\ &= WYW \cdot 0 + WXW \cdot 0 + 0 \cdot \bar{X}Y\bar{X} + W\bar{X}X \cdot 0 \quad (\text{Theorem 5a}) \\ &= 0 + 0 + 0 + 0 \quad (\text{Theorem 3b}) \\ &= 0 \end{aligned}$$

Example 1.88: $Y = ABC + AB + A$

$$\begin{aligned} &= A(BC + B + 1) \quad (\text{Theorem 4b}) \\ &= A \cdot 1 \quad (\text{Theorem 4a}) \\ &= A \end{aligned}$$

Example 1.89: $W = X\bar{Y} + \bar{Y}ZX$

$$\begin{aligned} &= X\bar{Y} + X\bar{Y}Z \quad (\text{Theorem 1a}) \\ &= X\bar{Y} \quad (\text{Theorem 9a}) \end{aligned}$$

Example 1.90: $W = \bar{A} + \bar{B}\bar{C}$

$$\begin{aligned} &= \bar{\bar{\bar{A}}} \bar{\bar{\bar{B}}} \bar{\bar{\bar{C}}} \quad (\text{De Morgan's Theorem 10b}) \\ &= ABC \end{aligned}$$

Example 1.91: $W = \overline{AB}(\bar{C} + \bar{D})$

$$\begin{aligned} &= \bar{\bar{\bar{A}}} \bar{\bar{\bar{B}}} + (\bar{C} + \bar{D}) \quad (\text{Theorem 10a}) \\ &= \bar{\bar{\bar{A}}} + \bar{\bar{\bar{B}}} + \bar{\bar{\bar{C}}} \bar{\bar{\bar{D}}} \quad (\text{Theorem 10a, 10b}) \\ &= A + B + CD \end{aligned}$$

Example 1.92: $W = \overline{A(C + D)} + \bar{C}(A + \bar{B})$

$$\begin{aligned} &= \bar{\bar{\bar{A}}} + (\bar{C} + \bar{D}) + \bar{\bar{\bar{C}}} (A + \bar{B}) \quad (\text{Theorem 10a, 10b}) \\ &= A + \bar{C} \bar{D} + C + \bar{A} \bar{B} \\ &= A + \bar{C}D + C + \bar{A}B \quad (\text{Theorem 8}) \end{aligned}$$

$$\begin{aligned}
 &= A + \overline{AB} + C + \overline{CD} \\
 &= A + B + C + D
 \end{aligned}
 \quad \begin{array}{l} (\text{Theorem } 1b) \\ (\text{Theorem } 9c) \end{array}$$

Example 1.93: $W = \overline{(\overline{A} + \overline{B})(\overline{C} + AB)}$

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

Example 1.94: $Y = A + AB + ABC + ABCD$

$$\begin{aligned}
 &= A(1 + B + BC + BCD) \\
 &= A \cdot 1 = A
 \end{aligned}$$

Example 1.95: $W = A(A + AB)(A + ABC)(A + ABCD)$

$$\begin{aligned}
 &= (A \cdot (A))[(A(1 + BC))][A(1 + BCD)] \\
 &= AA(A \cdot 1)(A \cdot 1) \\
 &= AAA = A
 \end{aligned}$$

Example 1.96: $(A_1 \overline{A}_2 A_3)(\overline{A_1 \overline{A}_2}) + A_2 A_3$

$$\begin{aligned}
 &= (\overline{A_1 \overline{A}_2 A_3}) + (\overline{A_1 + \overline{A}_2}) + \overline{A_2 A_3} \\
 &= (\overline{A_1} A_2 \overline{A}_3)(A_1 + \overline{A}_2) + \overline{A_2 A_3} \\
 &= \overline{A_1} A_1 A_2 \overline{A}_3 + \overline{A_1} A_2 \overline{A}_2 \overline{A}_3 + \overline{A_2} A_3 \\
 &= 0 + 0 + \overline{A_2} A_3 = \overline{A_2} A_3
 \end{aligned}$$

Example 1.97: $Y = (\overline{A} + B)(A + B)$

$$\begin{aligned}
 &= \overline{A}A + \overline{A}B + BA + BB \\
 &= \overline{A}B + AB + B \\
 &= (\overline{A} + A)B + B \\
 &= 1 \cdot B + B = B + B = B.
 \end{aligned}$$

Example 1.98: $Y = \overline{ABC} + \overline{AB}\overline{C} + A\overline{B}\overline{C} + AB\overline{C}$

$$\begin{aligned} &= (\overline{A}\overline{B} + \overline{A}B + A\overline{B} + AB)\overline{C} \\ &= (\overline{A}(\overline{B} + B) + A(\overline{B} + B))\overline{C} \\ &= (\overline{A} \cdot 1 + A \cdot 1)\overline{C} \\ &= (\overline{A} + A)\overline{C} \\ &= 1 \cdot \overline{C} = \overline{C}. \end{aligned}$$

Example 1.99: $Y = AB + A(B + C) + B(B + C)$

$$\begin{aligned} &= AB + AB + AC + BB + BC \\ &= AB + AB + AC + B + BC && (BB = B) \\ &= AB + AC + B + BC && (AB + AB = AB) \\ &= AB + AC + B && (B + BC = B) \\ &= AC + B && (B + BA = B) \end{aligned}$$

Example 1.100: $Y = (\overline{A}\overline{B}(C + BD) + \overline{A}\overline{B})C$

$$\begin{aligned} &= (\overline{A}\overline{B}C + \overline{A}\overline{B}BD + \overline{A}\overline{B})C \\ &= (\overline{A}\overline{B}C + A \cdot 0 \cdot D + \overline{A}\overline{B})C && (\overline{B}B = 0) \\ &= (\overline{A}\overline{B}C + 0 + \overline{A}\overline{B})C && (A \cdot 0 \cdot D = 0) \\ &= (\overline{A}\overline{B}C + \overline{A}\overline{B})C \\ &= \overline{A}\overline{B}CC + \overline{A}\overline{B}C \\ &= \overline{A}\overline{B}C + \overline{A}\overline{B}C && (CC = C) \\ &= \overline{B}C(A + \overline{A}) = \overline{B}C \cdot 1 && (A + \overline{A} = 1) \\ &= \overline{B}C \end{aligned}$$

Example 1.101: $Y = \overline{ABC} + \overline{AB}\overline{C} + \overline{A}\overline{B}\overline{C} + A\overline{B}C + ABC$

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

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

Example 1.102: $Y = \overline{AB + AC} + \overline{ABC}$

$$\begin{aligned}
 &= (\overline{AB})(\overline{AC}) + \overline{ABC} \\
 &= (\overline{A} + \overline{B})(\overline{A} + \overline{C}) + \overline{ABC} \\
 &= \overline{AA} + \overline{AC} + \overline{AB} + \overline{BC} + \overline{ABC} \\
 &= \overline{A} + \overline{AC} + \overline{AB} + \overline{BC} + \overline{ABC} \quad (\overline{AA} = \overline{A}) \\
 &= \overline{A} + \overline{AC} + \overline{AB} + \overline{BC} \quad (\overline{AB} + \overline{ABC} = \overline{AB}) \\
 &= \overline{A} + \overline{AB} + \overline{BC} \quad (\overline{A} + \overline{AC} = \overline{A}(1 + \overline{C}) = \overline{A}) \\
 &= \overline{A} + \overline{BC} \quad (\overline{A} + \overline{AB} = \overline{A}(1 + \overline{B}) = \overline{A})
 \end{aligned}$$

Example 1.103: $Y = AB + \overline{AC} + A\bar{B}C(AB + C)$

$$\begin{aligned}
 &= AB + \overline{AC} + A\bar{B}C \cdot AB + A\bar{B}CC \\
 &= AB + \overline{AC} + A\bar{B}CC \quad (AACB\bar{B} = AAC \cdot 0 = 0) \\
 &= AB + \overline{AC} + A\bar{B}C \quad (C \cdot C = C) \\
 &= AB + \overline{A} + \overline{C} + A\bar{B}C \\
 &= AB + \overline{A} + \overline{C} + \overline{BC} \quad (A + \overline{AB} = A + B) \\
 &= \overline{A} + AB + \overline{C} + C\bar{B} \\
 &= \overline{A} + B + \overline{C} + \overline{B} \\
 &= \overline{A} + \overline{C} + 1 \quad (B + \overline{B} = 1) \\
 &= 1
 \end{aligned}$$

Example 1.104: $Y = (\overline{A} + B)(A + B)$

$$\begin{aligned}
 &= \overline{AA} + \overline{AB} + AB + BB \\
 &= \overline{AB} + AB + BB \quad (A\overline{A} = 0) \\
 &= \overline{AB} + AB + B \quad (BB = B) \\
 &= (\overline{A} + A + 1)B \quad (\overline{A} + A + 1 = 1) \\
 &= B
 \end{aligned}$$

Example 1.105:
$$\begin{aligned} Y &= AB + A(B+C) + B(B+C) \\ &= AB + AB + AC + BB + BC \\ &= AB + AB + AC + B + BC && (BB = B) \\ &= AB + AC + B + BC && (AB + AB = AB) \\ &= AB + AC + B && (B + BC = B) \\ &= AB + B + AC \\ &= B + AC && (AB + B = B) \end{aligned}$$

Example 1.106:
$$\begin{aligned} Y &= \overline{AB + AC} + \overline{AB}C \\ &= (\overline{AB}) \cdot (\overline{AC}) + \overline{AB}C \\ &= (\overline{A} + \overline{B}) + (\overline{A} + \overline{C}) + \overline{AB}C \\ &= \overline{A}\overline{A} + \overline{A}\overline{C} + \overline{A}\overline{B} + \overline{B}\overline{C} + \overline{A}\overline{B}C \\ &= \overline{A} + \overline{B}\overline{C} + \overline{A}\overline{C} + \overline{A}\overline{B} + \overline{A}\overline{B}C && (\overline{A}\overline{A} + \overline{A}) \\ &= \overline{A} + \overline{B}\overline{C} + \overline{A}\overline{C} + \overline{A}\overline{B} && (\overline{A}\overline{B} + \overline{A}\overline{B}C = \overline{A}\overline{B}(1+C) = \overline{A}\overline{B}) \\ &= \overline{A} + \overline{B}\overline{C} && (\overline{A} + \overline{A}\overline{C} = \overline{A}(1+C) = \overline{A}) \\ &= \overline{A} + \overline{B}\overline{C} && (\overline{A} + \overline{A}\overline{B} = \overline{A}(1+\overline{B}) = \overline{A}) \end{aligned}$$

Example 1.107:
$$\begin{aligned} Y &= \overline{\overline{ABC} \cdot D} \\ &= \overline{ABC} + \overline{D} \\ &= (\overline{AB})C + \overline{D} \\ &= (\overline{A} + \overline{B})C + \overline{D} \\ &= \overline{AC} + \overline{BC} + \overline{D} \end{aligned}$$

Example 1.108: Simplify the following Boolean expression to a minimum number of literals:

(i) $\overline{AC} + ABC + A\overline{C}$ (Dec 2011)

$$\begin{aligned} &= \overline{A} + \overline{C} + ABC + A\overline{C} && (\text{Theorem 10}) \\ &= (\overline{A} + A\overline{C}) + (\overline{C} + ABC) && (\text{Theorem 9C}) \\ &= (\overline{A} + \overline{C}) + (\overline{C} + AB) \\ &= (\overline{A} + AB) + \overline{C} + \overline{C} \\ &= \overline{A} + B + \overline{C} + \overline{C} \end{aligned}$$

$$= \bar{A} + 1 + \bar{B}$$

$$= 1 + \bar{B} = 1$$

(ii) $XYZ + \bar{X}Y + XY\bar{Z}$

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

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

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

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

$$= y(1)$$

$$= y$$

(iii) $XY + YZ + X\bar{Y}\bar{Z}$

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

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

$$= xy + yz + xz$$

$$= xy + yz + zx$$

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

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

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

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

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

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

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

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

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

1.18 DUALITY

The dual is formed by replacing

AND with OR

OR with AND

0 with 1

1 with 0 in a Boolean expression.

The variables and components are left unchanged. This rule for forming the dual as,

$$[f(X_1, X_2, \dots, X_N, 0, 1, +, \cdot)]^D = f(X_1, X_2, \dots, X_N, 1, 0, \cdot, +)$$

If $F = A B \bar{C} + \bar{A} \bar{B} C$, then the dual expression is $F^D = (A + B + \bar{C})(\bar{A} + \bar{B} + C)$

Table 1.12 lists the postulates and theorems related to duality theorem.

TABLE 1.12: Duality Theorem

Expression	Dual
$X + 0 = X$	$X \cdot 1 = X$
$X + \bar{X} = 1$	$X \cdot \bar{X} = 0$
$X + X = X$	$X \cdot X = X$
$X + 1 = 1$	$X \cdot 0 = 0$
$X + Y = Y + X$	$XY = YX$
$X + (Y + Z) = (X + Y) + Z$	$X(YZ) = (XY)Z$
$X(Y + Z) = XY + XZ$	$X + YZ = (X + Y)(X + Z)$
$(\bar{X} + \bar{Y}) = \bar{X} \bar{Y}$	$(\bar{XY}) = \bar{X} + \bar{Y}$
$X + XY = X$	$X(X + Y) = X$

Proof:

- | | |
|---|--|
| <p>1. $X + X = X$</p> $\begin{aligned} X + X &= (X + X) \cdot 1 \\ &= (X + X)(X + \bar{X}) \\ &= X + X\bar{X} \\ &= X + 0 \\ &= X \end{aligned}$ | $\begin{aligned} X \cdot X &= X \\ X \cdot X &= XX + 0 \\ &= XX + X\bar{X} \\ &= X(X + \bar{X}) \\ &= X \cdot 1 \\ &= X \end{aligned}$ |
| <p>2. $X + 1 = 1$</p> $\begin{aligned} X + 1 &= 1 \cdot (X + 1) \\ &= (X + \bar{X})(X + 1) \\ &= X + \bar{X} \cdot 1 \\ &= X + \bar{X} \\ &= 1 \end{aligned}$ | $X \cdot 0 = 0$ |
| <p>3. $\overline{ABC + DEF} = (\bar{A} + \bar{B} + \bar{C})(\bar{D} + \bar{E} + \bar{F})$</p> $\begin{aligned} \overline{ABC + DEF} &= \overline{ABC} \cdot \overline{DEF} \\ &= (\bar{A} + \bar{B} + \bar{C})(\bar{D} + \bar{E} + \bar{F}) \end{aligned}$ | |

1.19 BOOLEAN FUNCTIONS

A Boolean function is an expression formed with binary variables, the two binary operators OR and AND, the unary operator NOT, parentheses and an equal sign. For a given value of the variables, the function can be either 0 or 1.

Consider, for example, the Boolean function

$$F_1 = xy\bar{z}$$

$F_1 = 1$ if $x=1, y=1$ and $\bar{z}=1$

$F_1 = 0$; otherwise

Boolean function also be represented in a Truth Table. To represent a function in a truth table, we need a list of the 2^n combinations of 1's and 0's of the ' n ' binary variables and a column showing the combinations for which the function is equal to 1 or 0.

TABLE 1.13 : Boolean Functions

X	Y	Z	$F_1 = XY\bar{Z}$	$F_2 = X + \bar{Y}Z$	$F_3 = \bar{X}\bar{Y}Z + \bar{X}YZ + X\bar{Y}$	$F_4 = X\bar{Y} + \bar{X}Z$
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

When $X = 1, Y=1, Z=1$

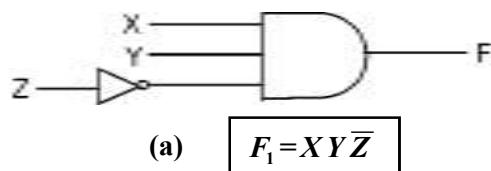
$$F_1 = 1 \cdot 1 \cdot 0 = 0$$

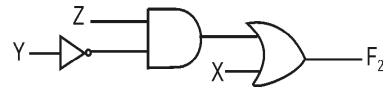
$$F_2 = 1 + 0 \cdot 1 = 1$$

$$F_3 = 0 \cdot 0 \cdot 1 + 0 \cdot 1 \cdot 1 + 1 \cdot 0 = 0$$

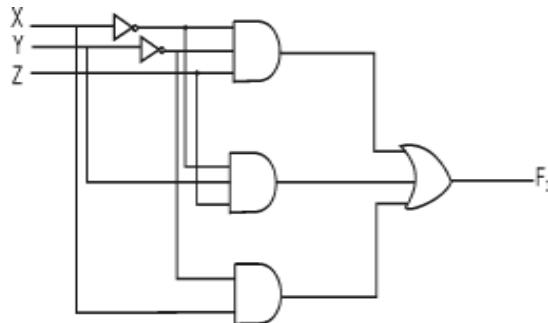
$$F_4 = 1 \cdot 0 + 0 \cdot 1 = 0$$

A Boolean function may be transformed from an algebraic expression into a logic diagram composed of AND, OR and NOT gates

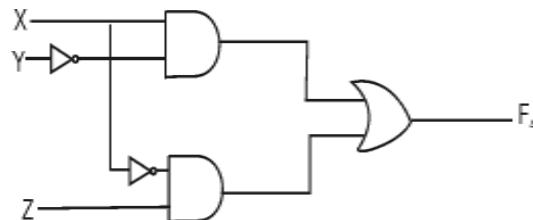




(b)
$$F_2 = X + \bar{Y}Z$$



(c)
$$F_3 = \bar{X}\bar{Y}Z + \bar{X}YZ + X\bar{Y}$$



(d)
$$F_4 = X\bar{Y} + \bar{X}Z$$

Fig. 1.20 : Implementation of Boolean functions with gates

To implement a Boolean function with a less number of gates we have to minimize literals and the number of terms. Usually, literals (Boolean variables in complemented or uncomplemented form) and terms are arranged in one of the two standard forms of switching equations:

- ◆ Sum of Product form (SOP)
- ◆ Product of Sum form (POS)

1.20 PRODUCT-OF-SUMS METHODS

The logical product of those fundamental sums that produce output 0's in the truth table. The corresponding logic circuit is an OR-AND circuit or the equivalent NOR-NOR circuit.

TABLE 1.14 : POS table

A	B	C	Y
0	0	0	$0 \rightarrow A + B + C$
0	0	1	1
0	1	0	1
0	1	1	$0 \rightarrow A + \bar{B} + \bar{C}$
1	0	0	1
1	0	1	1
1	1	0	$0 \rightarrow \bar{A} + \bar{B} + C$
1	1	1	1

$Y=0$, when $A = 0, B = 1$ and $C = 1$. So this particular combination makes the output of an OR gate equal to 0, when $\bar{B}=0$ and $\bar{C}=0$. Thus $A + \bar{B} + \bar{C}$ is on sum term. Similarly other two sum terms are $A + B + C$ and $\bar{A} + \bar{B} + C$. Thus the standard product of sum form is,

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

Figure 1.21 Shows the OR-AND logic circuit and **Figure 1.22** shows the NOR-NOR logic circuit for this expression.

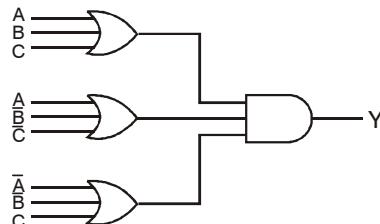


Fig. 1.21: OR-AND circuit

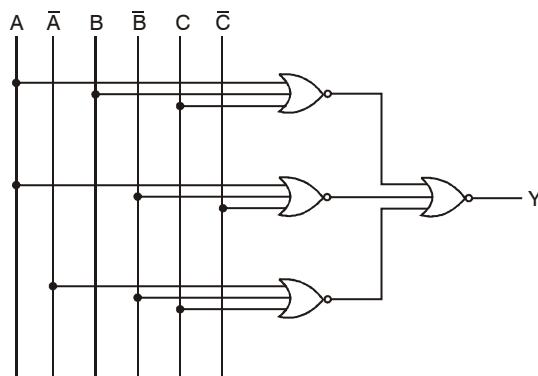


Fig. 1.22: NOR-NOR circuit

1.21 SUM-OF-PRODUCTS METHOD

The logical sum of those fundamental products that produce output 1's in the truth table. The corresponding logic circuit is an AND-OR circuit or the equivalent NAND-NAND circuit.

For example, $F = A\bar{B}C + AB\bar{C} + \bar{A}B\bar{C}$, is a standard sum of products of A , B and C are the only variables pertaining to the logic. Note that this expression is not in simplest form because we can write,

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

This simplified expression is a sum of products; but not a **standard** sum of products.

The logic expression corresponding to a given truth table can be written in a standard sum-of-products form of writing one product term for each input combination that produces an output of 1. These product terms are ORed together to create the standard sum of products.

TABLE 1.15 : SOP table

<i>A</i>	<i>B</i>	<i>C</i>	<i>F</i>	
0	0	0	0	
0	0	1	1	$\leftarrow \bar{A}\bar{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$\leftarrow A\bar{B}\bar{C}$
1	0	1	1	$\leftarrow A\bar{B}C$
1	1	0	0	
1	1	1	0	

We note that F is 1 when $A = 0$, $B = 0$ and $C = 1$, so this particular combination makes the output of an AND gate equal to 1 when \bar{A} and \bar{B} roman and C are all equal to 1. Thus $\bar{A}\bar{B}C$ is one product term. Similarly other two product terms are $A\bar{B}\bar{C}$ and $A\bar{B}C$. Thus the standard sum-of-products form is,

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

Consider the truth table given in **Table 1.16**.

TABLE 1.16: Design Truth Table

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

$$\rightarrow \bar{A}BC$$

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

$$\rightarrow AB\bar{C}$$

$$\rightarrow ABC$$

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

Figure 1.23 shows the AND-OR logic circuit and **Figure 1.24** shows the NAND-NAND circuit for the expression.

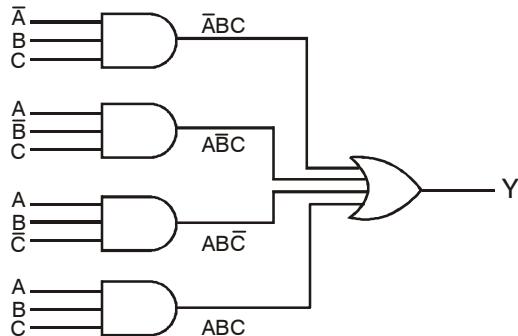


Fig. 1.23: AND-OR circuit

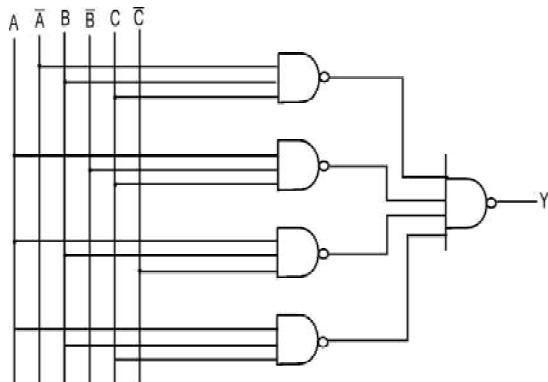


Fig. 1.24: NAND-NAND circuit

Example 1.109: Suppose a truth table has a low output for the first three input conditions: 000, 001 and 010. If all other outputs are high, what is the product-of-sum (POS) form?

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

Example 1.110: Suppose a 3 variable truth table has a high output for these input conditions: 000, 010, 100 and 110. What is the sum-of-product (SOP) form?

Solution: $Y = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$

Example 1.111: Convert the Boolean expressions to SOP form.

(a) $AB + B(CD + EF) = AB + BCD + BEF$

(b) $(A + B)(B + C + D) = AB + AC + AD + BB + BC + BD$

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

1.22 MINTERMS

The ‘n’ variables forming an AND term, with each variable being primed or unprimed, provide 2^n possible combinations, called Minterms or Standard Products.

Consider two binary variables x and y combined with an AND operation. Since each variable may appear in either form, there are 4 possible combinations:

$$\bar{x}\bar{y}, \bar{x}y, x\bar{y} \text{ and } xy$$

Each of these four AND terms represents one of the distinct areas in the Venn diagram and is called a Minterm.

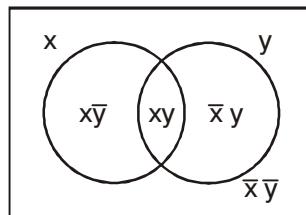


Fig. 1.25 : Venn Diagram for two variables

The 2^n difference minterms may be determined by a method similar to one shown in **Table 1.17** for 3 variables. The binary numbers from 0 to $2^n - 1$ are listed under the ‘n’ variables. Each Minterm

is obtained from an AND term of the ‘ n ’ variables, with each variable being primed, if the corresponding bit of the binary number is a ‘0’ and unprimed if a ‘1’.

Symbol for Minterm $\Rightarrow M_j$

TABLE 1.17 : Minterms and Maxterms

X Y Z	Minterms		Maxterms	
	Term	Symbol	Term	Symbol
0 0 0	$\bar{x}\bar{y}\bar{z}$	m_0	$x+y+z$	M_0
0 0 1	$\bar{x}\bar{y}z$	m_1	$x+y+\bar{z}$	M_1
0 1 0	$\bar{x}y\bar{z}$	m_2	$x+\bar{y}+z$	M_2
0 1 1	$\bar{x}yz$	m_3	$x+\bar{y}+\bar{z}$	M_3
1 0 0	$x\bar{y}\bar{z}$	m_4	$\bar{x}+y+z$	M_4
1 0 1	$x\bar{y}z$	m_5	$\bar{x}+y+\bar{z}$	M_5
1 1 0	$x\bar{y}\bar{z}$	m_6	$\bar{x}+\bar{y}+z$	M_6
1 1 1	xyz	m_7	$\bar{x}+\bar{y}+\bar{z}$	M_7

1.23 MAXTERMS

The ‘ n ’ variables forming an OR term, with each variable being primed or unprimed, provide in 2^n possible combinations, called Maxterms or Standard sums.

The eight maxterms for 3 variables, together with their symbolic designation are listed in Table. Each maxterm is obtained from an OR term of the ‘ n ’ variables, with each variable being unprimed if the corresponding bit is a 0 and primed if a 1.

Each Maxterm is the complement of its corresponding Minterm and Vice versa.

A Boolean function may be expressed algebraically from truth table (**Table 1.18**) by forming a minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms for example the function f_1 in the table is determined by expressing the combinations 001, 100 and 111 as $\bar{x}\bar{y}z$, $x\bar{y}\bar{z}$ and xyz .

$$f_1 = \bar{x}\bar{y}z + x\bar{y}\bar{z} + xyz = m_1 + m_4 + m_7 = 1$$

$$f_2 = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz = m_3 + m_5 + m_6 + m_7$$

“Any Boolean function can be expressed as a sum of Minterms (by ‘SUM’ is meant the ORing of terms)”.

The complement of f_1 is

$$\bar{f}_1 = (\bar{x}\bar{y}\bar{z} + \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}z + xy\bar{z})$$

TABLE 1.18 : Truth Table for f_1 and f_2

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The complement of \bar{f}_1 gives the function f_1 ,

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

$$\begin{aligned} f_1 &= (x+y+z)(x+\bar{y}+z) + (x+\bar{y}+\bar{z}) + (\bar{x}+y+\bar{z}) + (\bar{x}+\bar{y}+z) \\ &= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6 \end{aligned}$$

Similarly, it is possible to read the expression for f_2 from the table,

$$\begin{aligned} f_2 &= (x+y+z)(x+y+\bar{z})(x+\bar{y}+z) + (\bar{x}+y+z) \\ &= M_0 \cdot M_1 \cdot M_2 \cdot M_4 \end{aligned}$$

“Any Boolean function can be expressed as a product of Maxterms (by ‘product’ is the meant of ANDing of terms).”

1.24 CANONICAL FORM

Boolean functions expressed as a sum of minterms ($\sum m$) or product of maxterms ($\prod M$) are said to be in canonical forms.

1.24.1 Sum of Minterms

The sum of minterms of a Boolean function is obtained in two ways:

- ◆ from truth table
- ◆ from algebraic expression.

The following example clarifies these procedure:

Example 1.112: Express the Boolean function $F = A + \bar{B}C$ in a sum of minterms.

Method 1: The truth table for the function $F = A + \bar{B}C$ is shown in **Table 1.19**.

TABLE 1.19 : Truth table for $F = A + \bar{B}C$

A	B	C	\bar{B}	$\bar{B}C$	$F = A + \bar{B}C$
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	0	1

The combination of variables that produces '1' in the function are the minterms and then taking the OR of all these minterms is the sum of minterms.

$$\begin{aligned} F(A, B, C) &= m_1 + m_4 + m_5 + m_6 + m_7 \\ &= \sum m(1, 4, 5, 6, 7) \end{aligned}$$

Method 2: Explaining the expression into a sum of AND terms. Each term is then inspected to see if it contains all the variables. If it misses one or more variables, it is ANDed with an expression such as $(x + \bar{x})$, where x is one of the missing variables.

The Boolean function,

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

The function has three variables. The first term A is missing two variables and the second term $\bar{B}C$ is missing one variable. Therefore, the first term

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

This is still missing one variable, therefore

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

The second term,

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

Combining all terms, we obtain,

$$\begin{aligned} F &= A + \bar{B}C \\ &= ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + \bar{A}\bar{B}C \\ &= ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + \bar{A}\bar{B}C \\ &= \bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

Canonical form, $F(A, B, C) = \sum m(1, 4, 5, 6, 7)$

1.24.2 Product of Maxterms

The product of maxterms of a Boolean function is obtained

- ◆ from truth table
- ◆ from expression

Example 1.113: Express the Boolean function $F = XY + \bar{X}Z$ in a product of maxterm form.

Method 1:

TABLE 1.20 : Truth table for $F = XY + \bar{X}Z$

X	Y	Z	XY	$\bar{X}Z$	$F = XY + \bar{X}Z$
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	0	1

Form a maxterm for each combination of the variables that produces a 0 in the function and then form the AND of all those maxterms. Therefore the product of Maxterms,

$$F = M_0 \cdot M_2 \cdot M_4 \cdot M_5$$

$$F = (X, Y, Z) = \pi M(0, 2, 4, 5)$$

or

$$= \pi(0, 2, 4, 5)$$

Method 2: Expanding the expression into a product of OR terms using the distributive law, $X + YZ = (X + Y)(X + Z)$. Each term is then inspected to see if it contains all the variables. If it misses one or more variables, it is ORed with an expression such as $(x\bar{x})$, where x is one of the missing variables.

$$\begin{aligned} F &= XY + \bar{X}Z \\ &= (XY + \bar{X})(XY + Z) \\ &= (X + \bar{X})(Y + \bar{X})(X + Z)(Y + Z) \\ &= (\bar{X} + X)(X + Z)(Y + Z) \end{aligned}$$

The function has 3 variables: X , Y and Z . Each OR term is missing one variable; therefore,

$$\text{I term, } (\bar{X} + Y) = \bar{X} + Y + Z\bar{Z} = (\bar{X} + Y + Z)(\bar{X} + Y + \bar{Z})$$

$$\text{II term, } (X + Y) = X + Y + Y\bar{Y} = (X + Y + Z)(X + \bar{Y} + Z)$$

$$\text{III term, } (Y + Z) = Y + Z + X\bar{X} = (X + Y + Z)(\bar{X} + Y + Z)$$

Combining all the terms and removing those that appear more than once, we finally obtain:

$$\begin{aligned} F &= (X+Y+Z)(X+\bar{Y}+Z)(\bar{X}+Y+Z)(\bar{X}+Y+\bar{Z}) \\ &= M_0 \cdot M_2 \cdot M_4 \cdot M_5 \\ &= \pi M(0, 2, 4, 5) \\ F(X, Y, Z) &= \pi(0, 2, 4, 5) \end{aligned}$$

Example 1.114: Express the Boolean function

$$F = (A+\bar{B}+C)(\bar{B}+C+\bar{D})(A+\bar{B}+\bar{C}+D) \text{ in canonical POS form.}$$

Solution: The first term is missing variable 'D'

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

The second term is missing variable 'A'

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

The third term is already in standard form.

$$\text{The canonical form, } F = (A+\bar{B}+C+D)(A+\bar{B}+C+\bar{D})$$

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

Example 1.115: Convert the Boolean function into canonical SOP form,

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

Solution: First term is missing variable 'D'.

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

Second term is missing two variables C and D.

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

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

Third term is already in standard form,

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

Example 1.116: Obtain the canonical sum of product (SOP) from the function,

$$F = A + B$$

Soluton: $F = A + B$

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

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

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

Example 1.117: Obtain the canonical SOP of the function $F = AB + ACD$.

Solution: First term is missing two variables C and D

$$AB = AB(C + \bar{C})(D + \bar{D})$$

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

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

Second term is missing one variable ‘B’

$$ACD = ACD(B + \bar{B})$$

$$= ABCD + A\bar{B}CD$$

Canonical form,

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

Example 1.118: Obtain the canonical SOP of the function,

$$F = A + BC$$

$$\text{Solution: } F = A(B + \bar{B})(C + \bar{C}) + BC(A + \bar{A})$$

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

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

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

This result can be checked with the truth table.

TABLE 1.21 : Truth Table for $F = A + BC$

A	B	C	BC	$F = A + BC$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$\bar{A}BC$

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

$A\bar{B}C$

$AB\bar{C}$

ABC

$$\begin{aligned} F &= \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + AB\overline{C} + ABC \\ &= \sum m(3, 4, 5, 6, 7) \end{aligned}$$

Example 1.119: Obtain the canonical POS form $F = (A + \overline{B})(B + C)$

Solution: The first term has a missing variable ‘C’

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

The second term has a missing variable ‘A’

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

∴ Canonical form is $F = (A + \overline{B} + C)(A + \overline{B} + \overline{C})(A + B + C)(\overline{A} + B + C)$

Example 1.120: Express the function $F = A + \overline{B}C$ in

(a) Canonical SOP and (b) Canonical POS form

Solution: Canonical SOP form:

$$\begin{aligned} F &= A + \overline{B}C \\ &= A(B + \overline{B})(C + \overline{C}) + \overline{B}C(A + \overline{A}) \\ &= (AB + A\overline{B})(C + \overline{C}) + A\overline{B}C + \overline{A}\overline{B}C \\ &= ABC + AB\overline{C} + A\overline{B}C + A\overline{B}\overline{C} + A\overline{B}C + \overline{A}\overline{B}C \\ &= ABC + AB\overline{C} + A\overline{B}C + A\overline{B}\overline{C} + \overline{A}\overline{B}C \\ &= m_7 + m_6 + m_5 + m_4 + m_1 \\ F &= \sum(1, 4, 5, 6, 7) \end{aligned}$$

Canonical POS form:

$$\begin{aligned} F &= A + \overline{B}C \\ &= (A + \overline{B})(A + C) \\ &= (A + \overline{B} + C\overline{C})(A + C + B\overline{B}) \\ &= (A + \overline{B} + C)(A + \overline{B} + \overline{C})(A + B + C)(A + \overline{B} + C) \\ &= (A + \overline{B} + C)(A + \overline{B} + \overline{C})(A + B + C) \\ &= M_2 \cdot M_3 \cdot M_0 \\ F &= \pi(0, 2, 3) \end{aligned}$$

1.25 CONVERSION BETWEEN CANONICAL FORMS

The binary values of the product terms in given canonical SOP expression are not present in the equivalent canonical POS expression. Therefore to convert from canonical SOP to canonical POS, the following steps are taken:

Step 1: Evaluate each product term in the SOP expression. i.e., determine the binary numbers that represent the product terms.

Step 2: Determine all of the binary numbers not included in the evaluation of step 1.

Step 3: Write the equivalent sum term for each binary number from step 2 and express in POS form.

Using a similar procedure, we can convert POS to SOP form.

Example 1.121: Convert the following SOP expression to an equivalent POS expression:

$$\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC$$

Solution:

Step 1: The evaluation is: 000 + 010 + 011 + 101 + 111

Step 2: Since there are 3 variables, $2^3 = 8$ possible combinations are possible. The SOP expression contains 5 of these combinations, so the POS must contain other 3 combinations, which are 001, 100 and 110.

Step 3: POS expression is, $(A+B+\bar{C})(\bar{A}+B+C)(\bar{A}+\bar{B}+C)$

Method 2: The conversion between canonical forms can be done by another method:

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function. This is because the original function is expressed by those minterms that make the function equal to 1, whereas its complement as a 1 for those minterms that the function is a 0. As an example consider the function,

$$\begin{aligned} F(A, B, C) &= \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC \\ &= \sum(0, 2, 3, 5, 7) \end{aligned}$$

This has a complement that can be expressed as

$$\bar{F} = \sum(1, 4, 6) = m_1 + m_4 + m_6$$

The complement of F , \bar{F} is obtained by De Morgan's theorem,

$$\begin{aligned} \bar{F} &= (\overline{m_1 + m_4 + m_6}) \\ &= \bar{m}_1 \cdot \bar{m}_4 \cdot \bar{m}_6 \\ &= M_1 \cdot M_4 \cdot M_6 \\ &= \pi M(1, 4, 6) \\ &= \pi(1, 4, 6) \end{aligned}$$

$$\bar{m}_j = M_j$$

Example 1.122: Convert the canonical SOP into canonical POS $F(A, B, C) = \sum(1, 3, 6, 7)$.

Solution:

$$\begin{aligned} F &= \sum(1, 3, 6, 7) \\ \overline{F} &= \sum(0, 2, 4, 5) \\ &= m_0 + m_2 + m_4 + m_5 \end{aligned}$$

By DeMorgan's theorem,

$$\begin{aligned} F &= \overline{m_0 + m_2 + m_4 + m_5} \\ &= \overline{m_0} \cdot \overline{m_2} \cdot \overline{m_4} \cdot \overline{m_5} \\ &= M_0 \cdot M_2 \cdot M_4 \cdot M_5 \\ &= \pi(0, 2, 4, 5) \end{aligned}$$

Using this complementary relationship, find logical function in terms of maxterms. For example, for a 4 variable if

$$F = \sum(0, 2, 4, 6, 8, 10, 12, 14)$$

then the complement is $\overline{F} = \pi M(1, 3, 5, 7, 9, 11, 13, 15)$.

1.26 KARNAUGH MAPS

A Karnaugh map is a graphical representation of a truth table that can be used to reduce a logic circuit to its simplest terms. The size of the Karnaugh map depends on the amount of inputs that are listed in the truth table. An example of a three variable Karnaugh Map is shown below:

	$\overline{A}\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
C	$\overline{A}\overline{B}C$	$\overline{A}BC$	ABC	$A\overline{B}C$
\overline{C}	$\overline{A}\overline{B}\overline{C}$	$\overline{A}B\overline{C}$	$A\overline{B}\overline{C}$	$A\overline{B}C$

The terms within a Karnaugh Map are obtained by combining the row and column boolean expression that are shown at the top and left margins of the Karnaugh Map. Each combined term within the Karnaugh Map corresponds to a single line of inputs in a truth table. Terms that have a line over them corresponds to a low or zero input. Terms without any marking corresponds to a high or 1 input.

1.26.1 Constructing a Karnaugh Map

(1) Two variable maps:

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

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

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

(2) Three variable maps:

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

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

(3) Four variable maps:

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

$\bar{A}\bar{B}$	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
0	1	0	0	0
0	0	1	1	1
0	0	0	0	1
0	0	0	0	0

Pairs, Quads and Octets

Pairs

- * Pair eliminates are variable and its complements.
- * Pair of 1's horizontally and vertically adjacent.

$\bar{A}\bar{B}$	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$	
$\bar{A}\bar{B}$	0	0	0	0	
$\bar{A}B$	0	0	0	0	$Y = ABCD + ABC\bar{D}$
AB	0	0	1	1	$= ABC(D + \bar{D})$
$A\bar{B}$	0	0	0	0	$= ABC$

	$\bar{C}\bar{D}$	$\bar{C}D$	$C\bar{D}$	CD
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	0	0	0
$A\bar{B}$	0	0	0	1
AB	0	0	0	1

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

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

$$= AC\bar{D}$$

Quads: A quad is a group of four 1's that are horizontally or vertically adjacent

	$\bar{C}\bar{D}$	$\bar{C}D$	$C\bar{D}$	CD	$\bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + ABCD + ABC\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0	$= ABC(D + \bar{D}) + AB\bar{C}(D + \bar{D})$
$\bar{A}B$	0	0	0	0	$= ABC + AB\bar{C}$
$A\bar{B}$	1	1	1	1	$= AB(C + \bar{C})$
AB	0	0	0	0	$= AB$

	$\bar{C}\bar{D}$	$\bar{C}D$	$C\bar{D}$	CD	$ABCD + ABC\bar{D} + A\bar{B}CD + A\bar{B}C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0	$= ABC(D + \bar{D}) + A\bar{B}C(D + \bar{D})$
$\bar{A}B$	0	0	0	0	$= AC(B + \bar{B})$
$A\bar{B}$	0	0	1	1	$= AC$
AB	0	0	1	1	

Octet: An octet is a group of eight 1's.

An octet eliminates three variables and their complements.

	$\bar{C}\bar{D}$	$\bar{C}D$	$C\bar{D}$	CD	$Y = AB\bar{C}\bar{D} + A\bar{B}\bar{C}D + ABCD + ABC\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + A\bar{B}CD + A\bar{B}C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0	$= ABC(D + \bar{D}) + AB\bar{C}(D + \bar{D}) +$
$\bar{A}B$	0	0	0	0	$A\bar{B}\bar{C}(D + \bar{D}) + A\bar{B}C(D + \bar{D}) +$
$A\bar{B}$	1	1	1	1	$AB(C + \bar{C}) + A\bar{B}(C + \bar{C})$
AB	1	1	1	1	$= AB + A\bar{B} = A(B + \bar{B}) = A$

1.26.2 Karnaugh Map Simplifications

Encircle octets first, the quads second and the pairs last.

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$	
$\bar{A}\bar{B}$	0	1	1		$Y = A\bar{C} + C\bar{D} + \bar{A}\bar{B}D$
$\bar{A}B$	0	0	0	1	
AB	1	1	0	1	
$A\bar{B}$	1	1	0	1	

Overlapping Groups: It is possible to use the same 1 more than once.

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$	
$\bar{A}\bar{B}$	0	0	0	0	$Y = A + B\bar{C}D$
$\bar{A}B$	0	1	0	0	
AB	1	1	1	1	
$A\bar{B}$	1	1	1	1	

It is valid to encircle the 1's as shown below. But the isolated 1 results in a more complicated equation.

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$	
$\bar{A}\bar{B}$	0	0	0	0	$Y = A + \bar{A}B\bar{C}D$
$\bar{A}B$	0	1	0	0	
AB	1	1	1	1	
$A\bar{B}$	1	1	1	1	

Rolling the map

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$	
$\bar{A}\bar{B}$	0	0	0	0	$Y = B\bar{D}$
$\bar{A}B$	1	0	0	1	
AB	1	0	0	1	
$A\bar{B}$	0	0	0	0	

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$	
$\bar{A}\bar{B}$	0	0	0	0	$Y = B\bar{D}$
$\bar{A}B$	1	0	0	1	
AB	1	0	0	1	
$A\bar{B}$	0	0	0	0	

Visualize the picking up the karnaugh map and rolling it so that the left side touches the right side. By doing so, the two pairs can be realised as Quad.

\therefore The quad has the equation,

$$Y = B\bar{D}$$

Proof: To show whether the rolling is valid or not.

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

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

Rolling and Overlapping: It is possible to overlap and roll the map to get large groups.

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1 1	0	0	
$\bar{A}B$	1 1	0	1	
AB	1 1	0	1	
$A\bar{B}$	1 1	0	0	

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1 1	0	0	
$\bar{A}B$	1 1	0	1	
AB	1 1	0	1	
$A\bar{B}$	1 1	0	0	

$$Y = \bar{C} + BCD$$

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1 1	0	1	
$\bar{A}B$	1 1	0	1	
AB	1 1	0	0	
$A\bar{B}$	1 1	0	1	

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1 1	0	1	
$\bar{A}B$	1 1	0	1	
AB	1 1	0	0	
$A\bar{B}$	1 1	0	1	

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

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

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1 1	0	1	
$\bar{A}B$	1 1	0	1	
AB	1 1	0	0	
$A\bar{B}$	1 1	0	1	

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

Eliminating Redundant Groups

Redundant group is a group whose 1's are already used by other groups. The redundant group is eliminated as shown in **Figure 1.26 (b)**.

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	1	0
$\bar{A}B$	1	1	1	0
AB	0	1	1	1
$A\bar{B}$	0	1	0	0

Fig. 1.26 (a)

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	1	0
$\bar{A}B$	1	1	1	0
AB	0	1	1	1
$A\bar{B}$	0	1	0	0

Fig. 1.26 (b)

1.26.3 Don't Care Condition

In some digital system, certain output conditions never occur during normal operation. Therefore corresponding output never appears. Since the output never appears it is indicated by an 'X' in the truth table.

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	0	1	0
AB	X	X	X	X
$A\bar{B}$	0	0	X	X

$$Y = BCD$$

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	0	0	0
AB	X	X	X	X
$A\bar{B}$	X	X	X	X

$$Y = AD$$

Solution:

- Given the truth table, draw a Karnaugh map with other 0's, 1's and don't cares.
- Enclose the actual 1's on the Karnaugh map in the largest groups you can find by treating the don't cares as 1's.
- After the actual 1's have been included in groups, disregard the remaining don't cares by visualizing them as 0's.

Example:

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	0	0	0
$\bar{A}B$	0	0	0	0
AB	X	X	X	X
$A\bar{B}$	X	X	X	X

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

Here don't cares are of no help. The best way is, encircle the isolated 1, while treating don't cares as 0's.

1.26.4 Reducing Karnaugh Maps

The rules for reducing Karnaugh Maps are as follows:

- ❖ All of the 1's in the Karnaugh Map are called minterms.
- ❖ The 1's can be reduced in groups of 2, 4 and 8.
- ❖ Minterms that are next to each other horizontally or vertically, can be grouped together.
- ❖ Minterms that have been grouped in a Karnaugh Map, can be reduced to the boolean terms that are common with all the terms in the group.
- ❖ Minterms that cannot be grouped together, cannot be reduced.
- ❖ Use a minterm for grouping more than once.
- ❖ All 1's must be accounted for.

Example 1.123: Determine the Karnaugh Map and reduced boolean equation for the truth table shown in **Table 1.22**.

TABLE 1.22 : Truth Table

C	B	A	OUTPUT
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Without reduction the Boolean Equation for the above truth table is:

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

Each minterm corresponds to an instance in the truth table when the output is high. The Karnaugh Map for the above truth table, with the allowed groupings are shown below:

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

The map shows two groupings that cover each minterm. Each of these groupings will reduce to one term.

The two terms that are grouped together $\overline{A}\overline{B}C$ reduces to $\overline{A}\overline{B}\overline{C}$. This is because \overline{A} and \overline{B} are common to both terms.

The four terms $\overline{A}\overline{B}\overline{C}$ and $\overline{A}B\overline{C}$ and $AB\overline{C}$ and $A\overline{B}\overline{C}$ that are grouped together reduces to \overline{C} . This is because \overline{C} is the only input common to all four terms.

Therefore the boolean equation

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

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

1.26.5 Simplification of Sum of Product Expression

The procedure to simplify the SOP expression using K-map as follows:

- ❖ Plot the K-map and enter the 1's in those cells corresponding to the combinations for which function value is 1.
- ❖ Check the K-map for adjacent 1's and encircle those 1's which are not adjacent to any other 1's.
- ❖ Check for those 1's which are adjacent to only one other 1 and encircle such pairs.
- ❖ A group must contain either 1,2,4,8 or 16 ones (1's), which are all powers of two.
- ❖ Combine any pairs necessary to include any 1's that have not yet been grouped.
- ❖ Form the simplified expression by summing product terms of all the groups.

The minterms for variable and standard product terms and represented by 2 varibale K map, 3 variable K-map and 4 variable K-map are shown in **Figurre 1.27**.

	0	1
0	0	1
1	2	3

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

(a) Two Variable Map

	0	1
00	0	1
01	2	3
01	6	7
10	4	5

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

(b) Three Variable Map

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

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

(c) 4 Variable Map

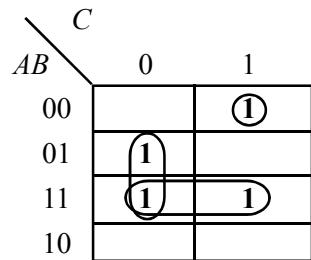
Fig.1.27: Representation of functions in the Map

Example 1.124: Simplify the following SOP expression on a Karnaugh Map.

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

Solution: The expression is evaluated as follows:

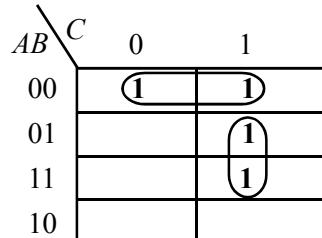
$$\begin{array}{cccc} \bar{A}\bar{B}C + \bar{A}B\bar{C} + AB\bar{C} + ABC \\ 001 & 010 & 110 & 111 \end{array}$$



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

Example 1.125: Simplify the SOP by using K-map

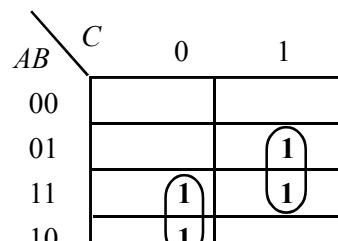
$$Y = \sum m (0,1,3,7)$$



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

Example 1.126: Simplify the expression $Y = m_3 + m_4 + m_6 + m_7$

Solution:



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

Example 1.127: Simplify the Boolean expression using K map $F = \overline{A}C + \overline{A}B + A\overline{B}C + BC$.

Solution: The given expression is not a standard SOP form. First convert this non-standard SOP to standard SOP and then simplify the expression using K-map.

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

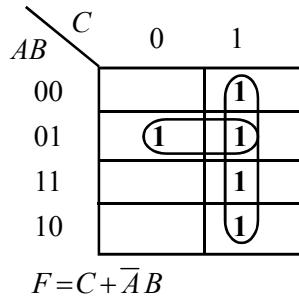
$$\overline{A}B = \overline{A}B(C + \overline{C}) = \overline{A}BC + \overline{A}B\overline{C}$$

$$BC = BC(A + \overline{A}) = ABC + \overline{A}BC$$

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

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

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



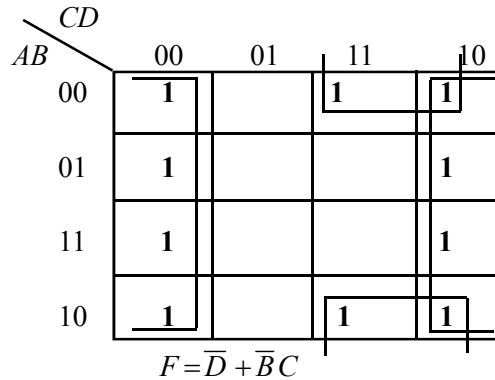
$$F = C + \overline{A}B$$

Example 1.128: Use a K map to minimize the following SOP expression:

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

$$\overline{A}\overline{B}CD + A\overline{B}CD + ABC\overline{D} + A\overline{B}C\overline{D} + \overline{A}\overline{B}CD + \overline{A}BC\overline{D}$$

Solution:

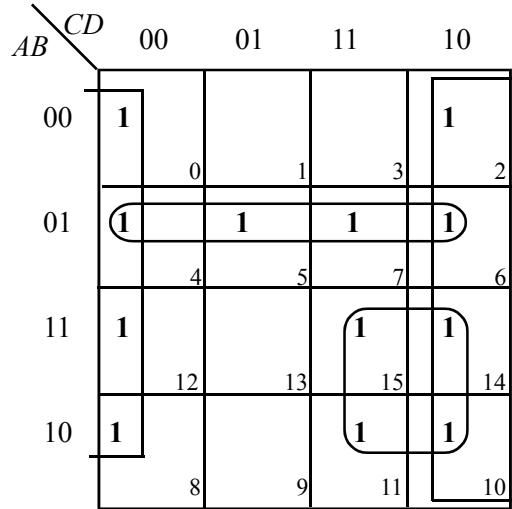


$$F = \overline{D} + \overline{B}C$$

Example 1.129: Simplify the expression

$$F = \sum m(0, 2, 4, 5, 6, 7, 8, 10, 11, 12, 14, 15)$$

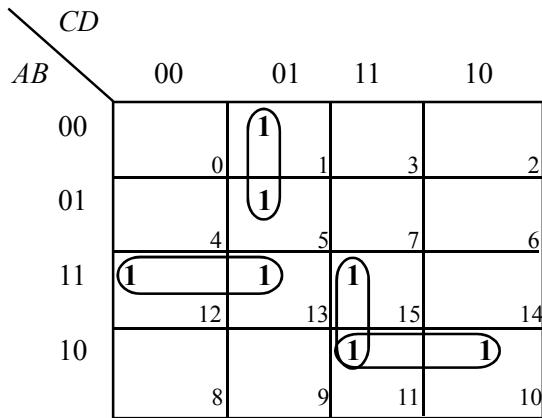
Solution:



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

Example 1.130: Simplify the expression using K-map

$$F = m_1 + m_5 + m_{10} + m_{11} + m_{12} + m_{13} + m_{15}$$



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

Example 1.131: Simplify the following SOP expression on a K-map

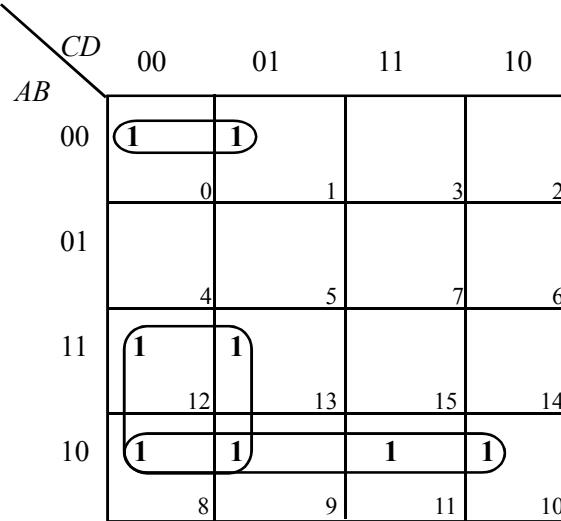
$$\overline{B}\overline{C} + A\overline{B} + AB\overline{C} + A\overline{B}C\overline{D} + \overline{A}\overline{B}\overline{C}D + A\overline{B}CD$$

Solution: The SOP expression is obviously not in standard form because each product term does not have 4 variables. The first and second term are both missing 2 variables, the third term is missing one variable and the rest of the terms are standard. First expand the terms by including all combinations of the missing variables numerically as follows:

$\bar{B}\bar{C}$	$+ A\bar{B}$	$+ AB\bar{C}$	$+ A\bar{B}C\bar{D}$	$+ \bar{A}\bar{B}\bar{C}D$	$+ A\bar{B}CD$
		1 1 0 0	1 0 1 0	0 0 0 1	1 0 1 1
0 0 0 0	1 0 0 0				
	1 1 0 1				
0 0 0 1	1 0 0 1				
1 0 0 0	1 0 1 0				
1 0 0 1	1 0 1 1				

Repeated terms are cancelled under the rule $A + A = A$, therefore,

$$\begin{aligned}
 F &= \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D \\
 &\quad + A\bar{B}CD + A\bar{B}C\bar{D} + AB\bar{C}\bar{D} + AB\bar{C}D \\
 &= \sum(0, 1, 8, 9, 10, 11, 12, 13)
 \end{aligned}$$



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

Example 1.132: Simplify using K-map

$$F(A, B, C, D) = \sum m(7, 8, 9) + d(10, 11, 12, 13, 14, 15)$$

Solution:

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

$$F = A + BCD$$

Note: Without don't cares, $F = A\bar{B}C + \bar{A}BCD$

With don't cares, $F = A + BCD$

Therefore, it is clear that, the advantage of using don't care terms is to get the simplest expression.

Example 1.133: Simplify using K-map

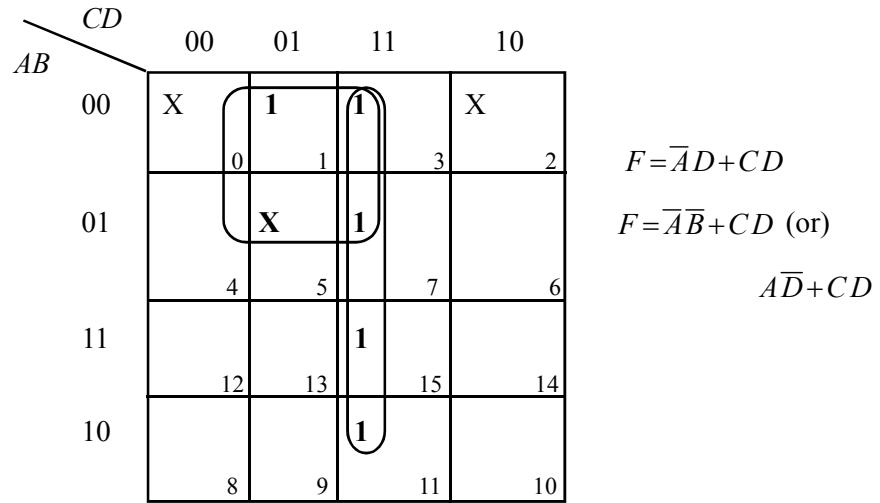
$$F(A, B, C, D) = \sum m(1, 3, 7, 11, 15) + d(0, 2, 5)$$

Solution:

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

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

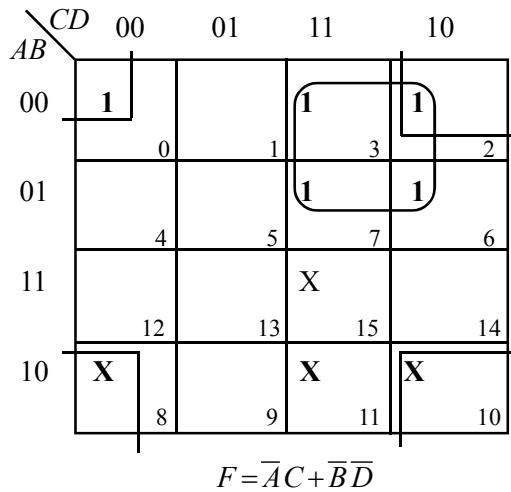
(or)



Example 1.134: Using the K-Map method, simplify the following Boolean function

$$F = \sum m(0, 2, 3, 6, 7) + \sum d(8, 10, 11, 15) \quad (\text{April 2005})$$

Solution:



Example: Simplify the following Boolean function F using Karnaugh map method.

Example 1.135: $F(A, B, C, D) = \sum(1, 4, 5, 6, 12, 14, 15)$

(Dec 2011)

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

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

Example 1.136: $F(A, B, C, D) = \sum(0, 1, 2, 4, 5, 7, 11, 15)$

(Dec 2011)

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

$$F(A, B, C, D) = \overline{AC} + \overline{ABD} + ACD + \overline{ABD}$$

Example 1.137: $F(A, B, C, D) = \sum(2, 3, 10, 11, 12, 13, 14, 15)$

(Dec 2011)

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

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

Example 1.138: $F(A, B, C, D) = \sum(0, 2, 4, 5, 6, 7, 8, 10, 13, 15)$

(Dec 2011)

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

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

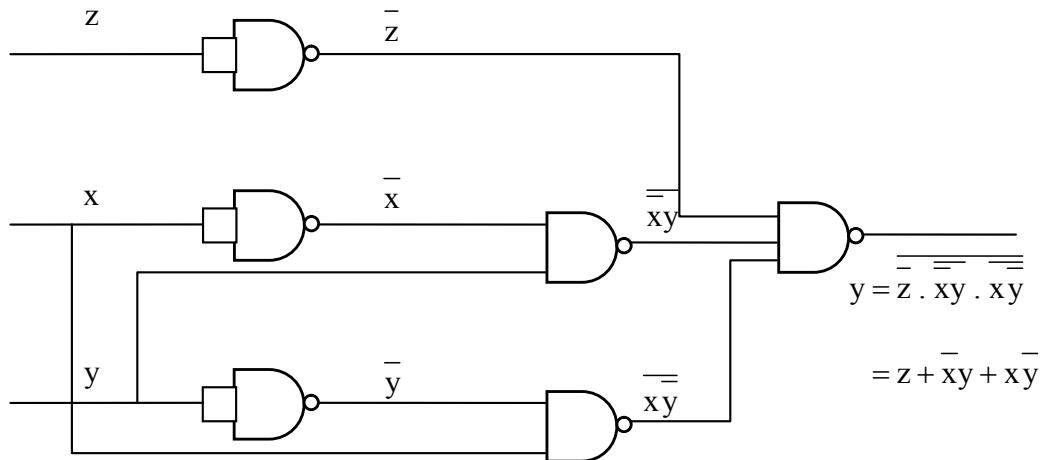
Example 1.139: Implement the switching function.

(May 2012)

$$F(x, y, z) = \sum m(1, 2, 3, 4, 5, 7) \text{ with NAND gates.}$$

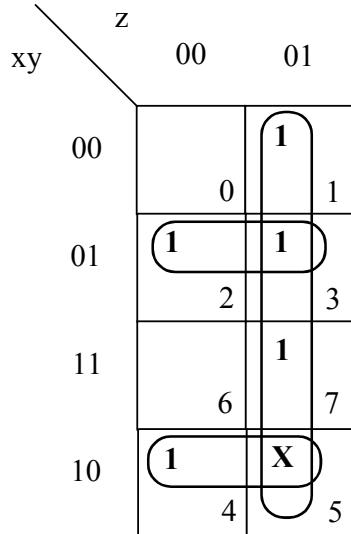
xy	z	0	1
00	0	1	
01	1	1	
11	4	5	
10	12	13	
	8	9	

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



Example 1.140: $f(A, B, C) = \sum m(0, 1, 3, 7) + \sum d(2, 5)$

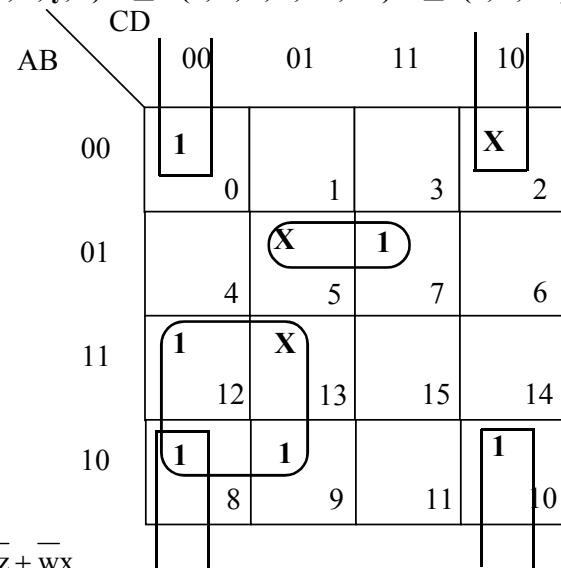
(May 2013)



$$f = C + \overline{AB}$$

Example 1.141: $F(w, x, y, z) = \sum m(0, 7, 8, 9, 10, 12) + \sum d(2, 5, 13)$

(May 2013)



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

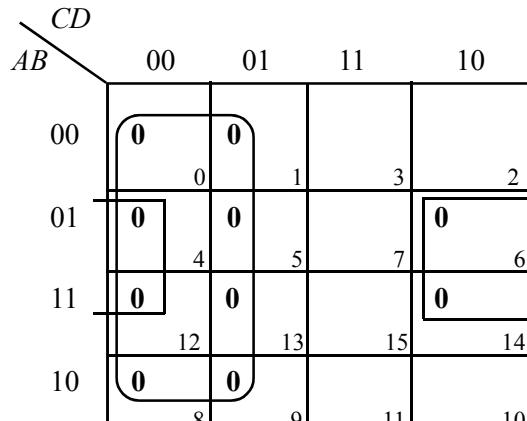
1.26.6 Simplification of Product of Sum expression

To simplify a POS expression, for each maxterm in the expression in, a '0' has to be entered in the corresponding cells and groups must be formed with '0' cells, instead of 1 cells to get the minterm (SOP) expression. The simplified term corresponding to each group can be obtained by the OR operation of the variables that are same for all cells of that group. Here, a variable corresponding to '0' has to be represented in an uncomplemented form.

Example 1.142: Simplify the POS expression using K map method.

$$F = \pi(0, 1, 4, 5, 6, 8, 9, 12, 13, 14)$$

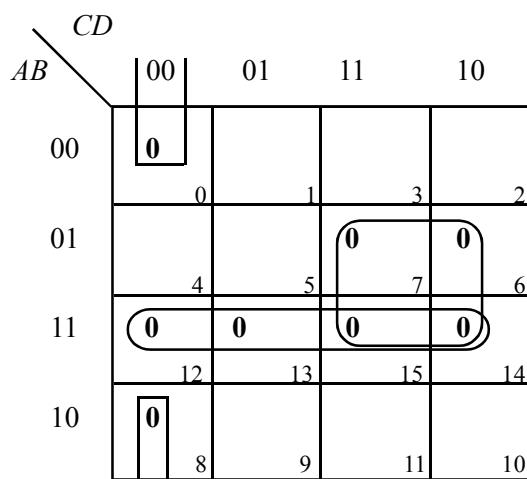
Solution:



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

Example 1.143: Simplify the POS expression

$$F = \pi(0, 6, 7, 8, 12, 13, 14, 15)$$



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

Example 1.144: Simplify the expression

$$F(A, B, C, D) = \pi M(4, 5, 6, 7, 8, 12) \cdot d(1, 2, 3, 9, 11, 14)$$

Solution:

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

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

Example 1.145: Simplify the POS expression,

$$F = \pi M(0, 3, 4, 7, 8, 10, 12, 14) \cdot d(2, 6)$$

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

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

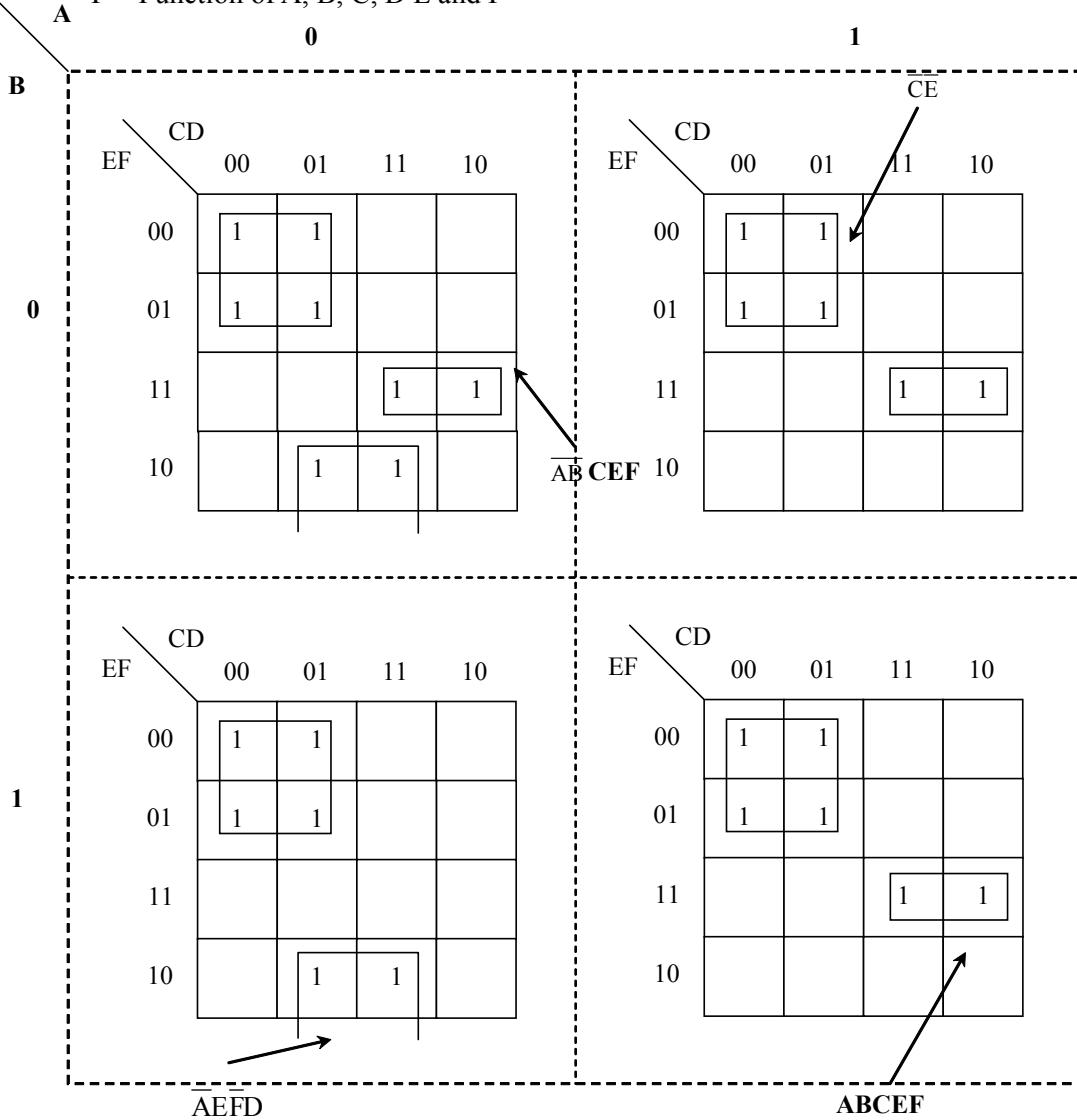
Example 1.146: Determine the minterm sum of product form of the switching function.

$$F = \sum (0, 1, 4, 5, 6, 11, 14, 15, 16, 17, 20, 22, 30, 32, 33, 36, 37, 48, 49, 52, 53, 59, 63) \quad (\text{Dec. 2010})$$

Solution:

Fill up the K-map with the variable given

F = Function of A, B, C, D E and F



Four 1's in each box form a group of 16 bits and their reduced function is \bar{CE} . Therefore

$$F = \sum (0, 1, 4, 5, 6, 11, 14, 15, 16, 17, 20, 22, 30, 32, 33, 36, 37, 48, 49, 52, 53, 59, 63)$$

$$F(A, B, C, D, E, F) = \bar{CE} + \bar{AE}\bar{FD} + \bar{AB}\bar{CEF} + ABCEF$$

Example 1.147: Minimize the following expression using Karnaugh map.

$$Y = A'B'C'D' + A'B'C'D + ABC'D' + AB'C'D + A'B'CD'$$

(May 2011)

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

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

Example 1.148: Simplify the following Boolean function F using Karnaugh map method.

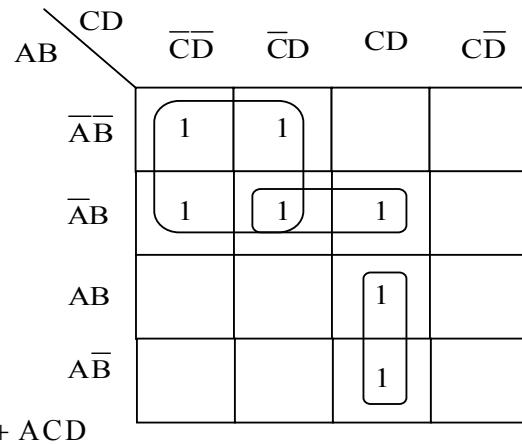
(i) $F(A, B, C, D) = \sum(1, 4, 5, 6, 12, 14)$

(Dec 2010)

		CD	\overline{CD}	$\overline{C}D$	CD	$C\overline{D}$
		AB	\overline{AB}	\overline{AB}	\overline{AB}	AB
\overline{AB}	\overline{AB}			1		
			1	1		1
AB	AB		1			1

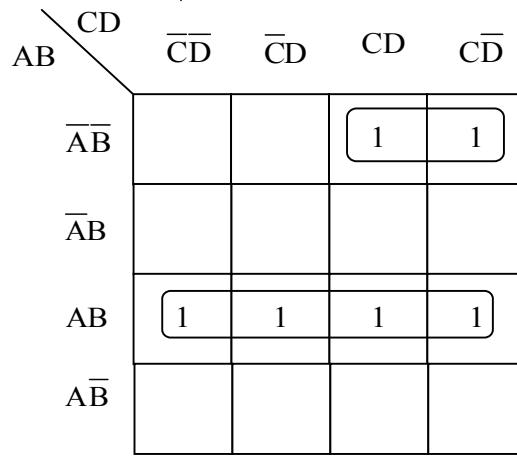
$$F = B\overline{C}\overline{D} + \overline{A}\overline{C}D + BCD$$

(ii) $F(A, B, C, D) = \sum(0, 1, 4, 5, 7, 11, 15)$



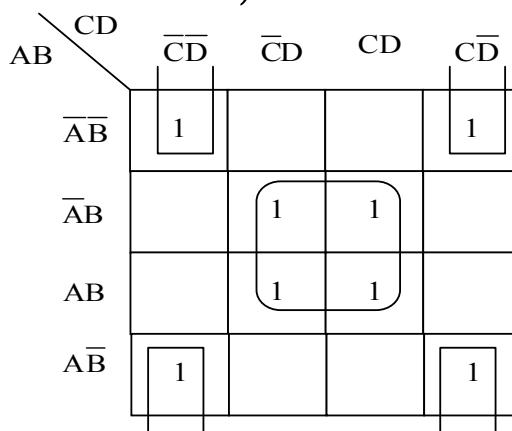
$$F = \bar{A}\bar{C} + \bar{A}BD + ACD$$

(iii) $F(A, B, C, D) = \sum(2, 3, 12, 13, 14, 15)$



$$F = AB + \bar{A}BC$$

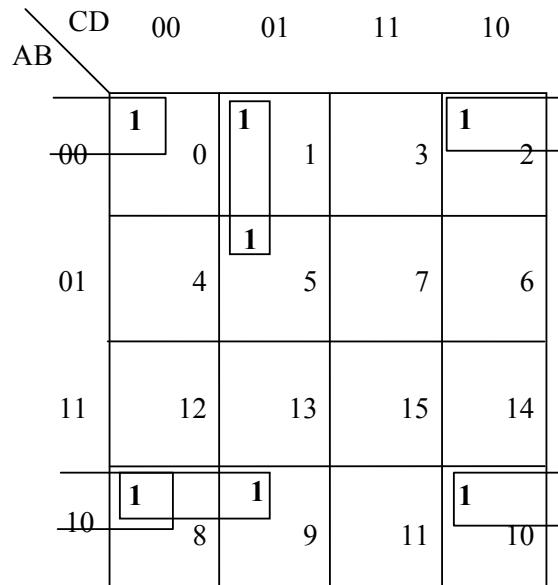
(iv) $F(A, B, C, D) = \sum(0, 2, 5, 7, 8, 10, 13, 15)$



$$F = BD + \bar{B}\bar{D}$$

Example 1.149: Simplify $F(A, B, C, D) = \sum (0, 1, 2, 5, 8, 9, 10)$ in sum of products and product of sums using K-map. **(Dec 2012)**

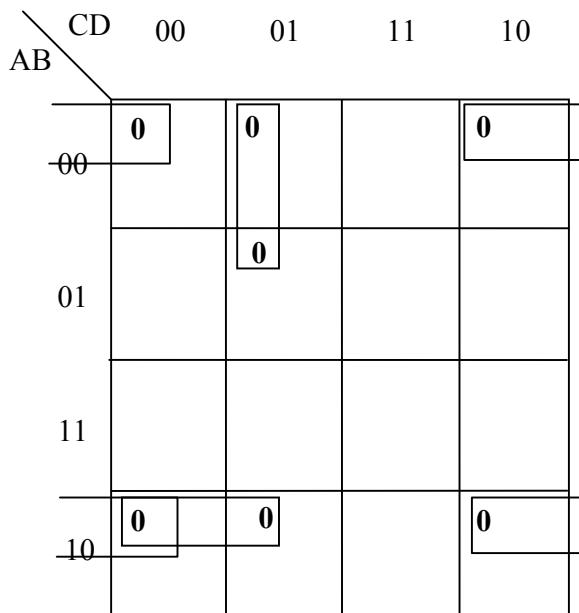
Solution:



SOP:

$$F = \overline{BD} + \overline{ACD} + A\overline{BC}$$

POS:



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

1.26.7 FIVE VARIABLE MAP

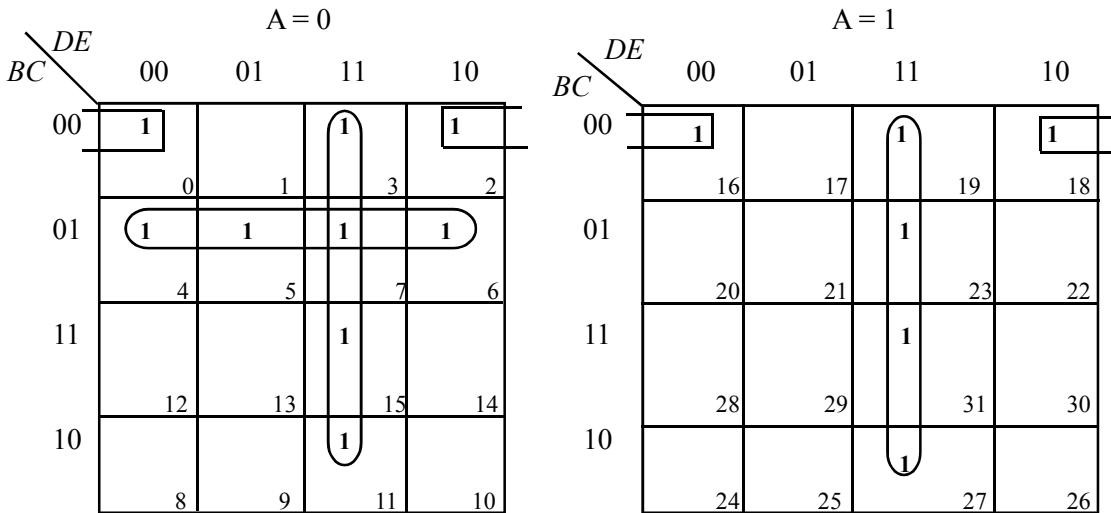
The five-variable map consists of 2 four-variable maps with A, B, C, D and E. Variable A distinguishes between the two maps, as indicated on the top of the diagram as A = 0 and A = 1. Minterms 0–15 belong with A = 0 and minterms 16–31 with A = 1.

		A = 0						A = 1				
		BC	00	01	11	10	BC	00	01	11	10	
DE		00	0	1	3	2	DE	00	16	17	19	18
		01	4	5	7	6		01	20	21	23	22
		11	12	13	15	14		11	28	29	31	30
		10	8	9	11	10		10	24	25	27	26

Example 1.150

Simplify the Boolean expression:

$$F(A, B, C, D, E) = \Sigma(0, 2, 3, 4, 5, 6, 7, 11, 15, 16, 18, 19, 23, 27, 31)$$



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

In DE and $\bar{B} \bar{C} \bar{E}$ terms, A is not included because the adjacent squares belong to both $A = 0$ and $A = 1$.

In $\bar{A} \bar{B} C$ term, it is necessary to include \bar{A} because all the squares are associated with $A = 0$.

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

Example 1.151: Simplify the Boolean function

$$F(A, B, C, D, E) = \Sigma(0, 1, 4, 5, 16, 17, 21, 25, 29)$$

		A = 0			
		DE	B	C	D
BC	DE	00	01	11	10
		1 0 1 4 12 8	1 1 5 13 11	3 7 15 11	2 6 14 10

		A = 1			
		DE	B	C	D
BC	DE	00	01	11	10
		1 16 20 28 24	1 1 21 1 25	17 19 23 29 27	18 22 30 31 26

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

Example 1.152: Find the minimal sum of product form for the following switching function:

$$f(x_1, x_2, x_3, x_4, x_5) = \Sigma m(2, 3, 6, 7, 11, 12, 13, 14, 15, 23, 28, 29, 30, 31) \quad (\text{May 2006})$$

		x ₁ = 0					
		x ₄ x ₅	x ₂ x ₃	00	01	11	10
				0 1 1 4 12 8		1 3 1 5 13 9	2 6 14 10 11
						1 1 1 1 1 1	1 1 1 1 1 1

		x ₁ = 1					
		x ₄ x ₅	x ₂ x ₃	00	01	11	10
				16 20 1 1 24		17 21 1 29	19 23 1 31 27
						1 1 1 1 1	1 1 1 1 1

$$F = x_2x_3 + \overline{x}_1x_4x_5 + \overline{x}_1\overline{x}_2x_4 + x_1x_3x_4x_5$$

Example 1.153: Find the minimal sum of product expression for the following switching function:

$$f(x_1, x_2, x_3, x_4, x_5) = \sum m(1, 2, 3, 6, 8, 9, 14, 17, 24, 25, 26, 27, 30, 31) + \sum d(4, 5)$$

(May 2006)

		$x_1 = 0$			
		00	11	10	
x_2x_3	x_4x_5	00	1	1	1
		0	1	3	2
01	X	X			
		4	5	7	6
11					
		12	13	15	14
10		1	1		
		8	9	11	10

		$x_1 = 1$			
		00	01	11	10
x_2x_3	x_4x_5	00	1		
		16	17	19	18
01		20	21	23	22
11		28	29	31	30
		1	1	1	1
10		1	1	1	1
		24	25	27	26

$$F = x_2 \bar{x}_3 \bar{x}_4 + \bar{x}_3 \bar{x}_4 x_5 + x_1 x_2 x_4 + \bar{x}_1 \bar{x}_2 \bar{x}_3 x_4 + \bar{x}_1 x_3 x_4 \bar{x}_5$$

1.27 TABULATION METHOD (QUINE-McCLUSKEY METHOD)

The K map method of minimization of logic functions is convenient as long as the number of variables does not exceed 4 or 5. When the number of variable increases, K map become very difficult. To avoid this difficult, Quine-McCluskey or Tabulation method can be used. This method is developed by Quine and McCluskey. The procedure for simplification of Boolean function by Quine- McCluskey method is as follows:

- ❖ Each minterm should be expressed by its binary representation.
- ❖ Arrange the minterms based on the number of 1's
- ❖ Compare each binary number from one group to other and if they differ only one bit position, put dash (-) mark and copy the remaining term. Please tick (✓) mark after each comparison.
- ❖ Apply the same process described in step 3 for the resultant column and these cycles have to be continued until no new list can be found (i.e., no further elimination of literals)
- ❖ List the unchecked (unticked) implicant and form prime implicant chart.

Prime implicant chart

- ❖ The prime implicants should be represented in rows and each minterm of the function in a column.

- ❖ The cross (X) mark should be placed in each row to show the comparison of minterms that make the prime implicants.
- ❖ Search for single X column and select prime implicants corresponding to that dot by putting the star (*) mark in front of it.
- ❖ Prime implicants that cover minterms with a single cross in their column are called essential prime implicants.
- ❖ Write the simplified expression using prime implicants.

Example 1.154: Simplify the Boolean function by using tabulation method

$$F(A,B,C,D) = \sum m(0,2,3,6,7,8,10,12,13)$$

Solution: The minterms are represented in the binary form as shown in **Table 1.23(a)**

Table 1.23(a) Binary representation of minterms

Minterm	Binary equivalent
0	0 0 0 0
2	0 0 1 0
3	0 0 1 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
10	1 0 1 0
12	1 1 0 0
13	1 1 0 1

The above binary representation are grouped into a number of sections in terms of the number of 1's as shown in **Table 1.23(b)**

Table 1.23(b) Group of minterms for different number of 1's

Number of 1's	Minterm	A	B	C	D
0	0	0	0	0	✓
1	2	0	0	1	0
	8	1	0	0	0
2	3	0	0	1	1
	6	0	1	1	0
	10	1	0	1	0
	12	1	1	0	0
3	7	0	1	1	1
	13	1	1	0	1

Any two numbers in these groups which differ from each other by only one variable can be chosen and combined, to get 2-cell combination as shown in **Table 1.23(c)**.

Table 1.23(c) 2-cell combination

Combination	A	B	C	D	
(0,2)	0	0	–	0	✓
(0,8)	–	0	0	0	✓
(2,3)	0	0	1	–	✓
(2,6)	0	–	1	0	✓
(2,10)	–	0	1	0	✓
(8,10)	1	0	–	0	✓
(8,12)	1	–	0	0	
(3,7)	0	–	1	1	✓
(6,7)	0	1	1	–	✓
(12,13)	1	1	0	–	

Table 1.23(d) 4-cell combination

Combination	A	B	C	D
(0,2,8,10)	–	0	–	0
(2,3,6,7)	0	–	1	–

From the 2-cell combinations, any variable and a dash (–) in the same position can be combined to form 4-cell combination as shown in **Table 1.23(d)**.

The cells (0,2) and (8,10) from the same 4-cell combination as the cells (0,8) and (2,10). The order in which the cells are placed in a combination does not have any effect. Thus the (0,2,8,10) combination may be given as (0,8,2,10)

$$\text{i.e., } (0,2,8,10) = (0,8,2,10)$$

$$(2,3,6,7) = (2,6,3,7)$$

Using Table 1.28(c) and (d) the prime implicants table can be as shown in **Table 1.23(e)**.

Table 1.23(e) Prime Implicant Table

Prime Implicants	Minterms								
	0	2	3	6	7	8	10	12	13
(8, 12)						✗		✗	
(12,13)*								✗	✗
(0,2,8,10)*	✗	✗	✗			✗	✗		
(2,3,6,7)*	✓		✓	✓	✓				✓

The columns having only one cross (X) mark correspond to essential prime implicants. A tick mark put against every column which has only one cross mark. A star (*) mark is placed against every essential prime implicant. The sum of the prime implicants gives the function in its minimal SOP form. Therefore, $F = (1\ 1\ 0\ -) + (-\ 0\ -\ 0) + (0\ -\ 1\ -)$

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

Example 1.155: Find the minimal SOP for the given function using Quine-McCluskey method

$$F = \sum m(0, 1, 2, 8, 10, 11, 14, 15)$$

(April 2005)

Solution:

Binary representation of minterms

Minterms	Binary equivalent
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
8	1 0 0 0
10	1 0 1 0
11	1 0 1 1
14	1 1 1 0
15	1 1 1 1

Group of minterms for different number of 1's:

Number of 1's	Minterm	A	B	C	D
0	0	0	0	0	✓
1	1	0	0	0	✓
	2	0	0	1	0
	8	1	0	0	0
2	10	1	0	1	0
3	11	1	0	1	1
	14	1	1	1	0
4	15	1	1	1	1

2 cell combination		4 cell combination								
Combination	A	B	C	D	Combination	A	B	C	D	
(0,1)	0	0	0	-	(0,2,8,10)	-	0	-	0	
(0,2)	0	0	-	0	✓	(10,11,14,15)	1	-	1	-
(0,8)	-	0	0	0	✓					
(2,10)	-	0	1	0	✓					
(8,10)	1	0	-	0	✓					
(10,11)	1	0	1	-	✓					
(10,14)	1	-	1	0	✓					
(11,15)	1	-	1	1	✓					
(14,15)	1	1	1	-	✓					

Here $(0,2,8,10) = (0,8,2,10)$, $(10,11,14,15) = (10,14,11,15)$

Prime Implicants Table

Prime Implicants	Minterms							
	0	1	2	8	10	11	14	15
$(0,1)^*$	x	x						
$(0,2,8,10)^*$	x		x	x	x			
$(10,11,14,15)^*$			x	x	x	x	x	x
	✓	✓	✓			✓	✓	✓

$$F = [(0\ 0\ 0\ -) + (-\ 0\ -\ 0) + (1\ -\ 1\ -)]$$

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

Example 1.156: Simplify the following function using tabulation method

$$F(A,B,C,D) = \sum m(1,2,3,5,9,12,14,15) + \sum d(4,8,11) \quad (\text{Dec 2005})$$

Solution: The don't care conditions are used to find the prime implicants; but it is not compulsory to include don't care term in the final expression.

Binary representation of minterms

Minterms	Binary equivalent
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
5	0 1 0 1
9	1 0 0 1
12	1 1 0 0
14	1 1 1 0
15	1 1 1 1
d4	0 1 0 0
d8	1 0 0 0
d11	1 0 1 1

Group of minterms for difference number of 1's:

<i>Number of 1's</i>	<i>Minterm</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
1	1	0	0	0	1 ✓
	2	0	0	1	0 ✓
	d4	0	1	0	0 ✓
	d8	1	0	0	0 ✓
2	3	0	0	1	1 ✓
	5	0	1	0	1 ✓
	9	1	0	0	1 ✓
	12	1	1	0	0 ✓
3	d11	1	0	1	1 ✓
	14	1	1	1	0 ✓
4	15	1	1	1	1 ✓

2-cell combinations				
Combination	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
(1,3)	0	0	—	1 ✓
(1,5)	0	—	0	1
(1,9)	—	0	0	1 ✓
(2,3)	0	0	1	—
(4,5)	0	1	0	—
(4,12)	—	1	0	0
(8,9)	1	0	0	—
(3,11)	—	0	1	1 ✓
(9,11)	1	0	—	1 ✓
(12,14)	1	1	—	0
(11,15)	1	—	1	1
(14,15)	1	1	1	—

4-cell combinations				
Combination	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
(1,3,9,11)	—	0	—	1

Prime Implicant Table

Prime implicant	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
(1,5)	0	–	0	1
(2,3)	0	0	1	–
(4,5)	0	1	0	–
(4,12)	–	1	0	0
(8,9)	1	0	0	–
(8,12)	1	–	0	0
(12,14)	1	1	–	0
(11,15)	1	–	1	1
(14,15)	1	1	1	–
(1,3,9,11)	–	0	–	1

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

Prime Implicants	m_1	m_2	m_3	d_4	m_5	d_8	m_9	d_{11}	m_{12}	m_{14}	m_{15}
(1,5)*	×				×						
(2,3)*		×	×								
(4,5)				×	×						
(4,12)				×							
(8,9)						×	×				
(8,12)						×				×	
(12,14)*								×			×
(11,15)								×			×
(14,15)*										×	×
(1,3,9,11)*	×		×			×	×				
		√									

- ❖ Only m_2 column has single cross (X) mark and hence the prime implicant corresponding to it (2,3) is included in the final expression.
- ❖ m_1 column has 2 cross marks. We can include the prime implicants which has more minterms - (1,3,9,11).
- ❖ Columns d_4 , d_8 and d_{11} are don't cares.
- ❖ m_5 is not included yet, therefore prime implicants (1,5) is included in the final expression.
- ❖ m_{12} is not included yet, therefore prime implicant (12,14) is included.
- ❖ m_{15} also can be included in the final expression by including prime implicant (14,15)

The final expression is,

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

Example 1.157: Simplify the given function

$$F = (A, B, C, D) = \sum m(2, 3, 7, 9, 11, 13) + \sum d(1, 10, 15)$$

Solution: The don't cares are treated like required minterms when finding the prime implicants.

Minterm	Binary Representation
d1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
7	0 1 1 1
9	1 0 0 1
d10	1 0 1 0
11	1 0 1 1
13	1 1 0 1
d15	1 1 1 1

Group of minterms for different number of 1's:

Number of 1's	Minterm	A B C D
1	1	0 0 0 1 ✓
	2	0 0 1 0 ✓
2	3	0 0 1 1 ✓
	9	1 0 0 1 ✓
3	10	1 0 1 0 ✓
	7	0 1 1 1 ✓
	11	1 0 1 1 ✓
	13	1 1 0 1 ✓
4	15	1 1 1 1 ✓

2-cell combination					4-cell combination				
Combination	A	B	C	D	Combination	A	B	C	D
(1,3)	0	0	–	1	✓				
(1,9)	–	0	0	1	✓				
(2,3)	0	0	1	–	✓				
(2,10)	–	0	1	0	✓				
(3,7)	0	–	1	1	✓				
(3,11)	–	0	1	1	✓				
(9,11)	1	0	–	1	✓				
(9,13)	1	–	0	1	✓				
(10,11)	1	0	1	–	✓				
(7,15)	–	1	1	1	✓				
(11,15)	1	–	1	1	✓				
(13,15)	1	1	–	1	✓				

The don't care columns are omitted when forming prime implicants table

Prime implicants	2	3	7	9	11	13
(1,3,9,11)		×		×	×	
(2,3,10,11)*	×	×			×	
(3,7,11,15)*		×	×		×	
(9,11,13,15)*			×	×	×	×
	✓		✓			✓

$$\begin{aligned}
 F &= (\neg 0 1 \neg) + (\neg \neg 1 1) + (1 \neg \neg 1) \\
 &= \overline{B} C + CD + AD
 \end{aligned}$$

Example 1.158: Simplify the following 5 variable expression using Mccluskey method.

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

(Dec 2010)

Solution:

	(1)	(2)	(3)
0	0000	0, 1 (1)	0, 1, 8, 9
1	0001	0, 8 (8)	
8	1000	1, 9(8)	
9	01001	8, 9(1)	
24	11000	8, 24(16)	
11	1011	9, 11(2)	
15	01111	11, 15(4)	
29	11101	15, 31(16)	
30	11110	30, 31(1)	
31	11111		

Prime implicant table

	0	1	9	15	24	29	30
0, 1, 8, 9	✓	✓	✓				
8, 24			✓			✓	
9, 11			✓				
11, 15				✓			
15, 31					✓		
29, 31							
30, 31							✓
	✓	✓			✓	✓	✓

Answer:

$$A'C'D' + BC'D'E' + A'BDE + ABCE + ABCD$$

Or

$$A'C'D' + BC'D'E' + BCDE + ABCE + ABCD$$

Example 1.159: Minimize the expression using Quine McCluskey (Tabulation) method

$$Y' = A'B'C'D' + A'BC'D + ABC'D' + ABC'D + AB'C'D + A'B'CD' \quad (\text{May 2012})$$

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

Given, $F(A, B, C, D) = \{0, 2, 5, 9, 12, 13\}$

Min term	Binary equivalent				
0	0	0	0	0	0
2	0	0	1	0	0
5	0	1	0	1	1
9	1	0	0	1	1
12	1	1	0	0	0
13	1	1	0	1	1

Number of 1's	Minterms	A	B	C	D
0	0	0	0	0	0
1	2	0	0	1	0
2	5 9 12	0 1 1	1 0 1	0 0 0	1 1 0
3	13	1	1	0	1

2 – cell combination

Combination	A	B	C	D
(0, 2)	0	0	–	0
(5, 13)	–	1	0	1
(9, 13)	1	–	0	1
(12, 13)	1	1	0	–

Prime implicants	0	2	5	9	12	13
(0, 2)	×	×				
(5, 13)			×			×
(9, 13)				×		×
(12, 13)					×	×
	✓	✓	✓	✓	✓	

$$F = A'B'D' + BC'D + AC'D + ABC'$$

Example 1.160: Simplify the Boolean function using Quine McCluskey method:

$$F(A, B, C, D, E, F) = \sum m(0, 5, 7, 8, 9, 12, 13, 23, 24, 25, 28, 29, 37, 40, 42, 44, 46, 55, 56, 57, 60, 61) \quad (\text{May 2013})$$

Solution:

Minterms	Binary equivalent					
0	0	0	0	0	0	0
5	0	0	0	1	0	1
7	0	0	0	1	1	1
8	0	0	1	0	0	0
9	0	0	1	0	0	1
12	0	0	1	1	0	0
13	0	0	1	1	0	1
23	0	1	0	1	1	1
24	0	1	1	0	0	0
25	0	1	1	0	0	1
28	0	1	1	1	0	0
29	0	1	1	1	0	1
37	1	0	0	1	0	1
40	1	0	1	0	0	0
42	1	0	1	0	1	0
44	1	0	1	1	0	0
46	1	0	1	1	1	0
55	1	1	0	1	1	1
56	1	1	1	0	0	0
57	1	1	1	0	0	1
60	1	1	1	1	0	0
61	1	1	1	1	0	1

Group of minterms for different number of 1's

Number of 1's	Minterms	A	B	C	D	E	F
0	0	0	0	0	0	0	0
1	8	0	0	1	0	0	0
2	5 9	0	0	0	1	0	1

	12	0	0	1	1	0	0
	24	0	1	1	0	0	0
	40	1	0	1	0	0	0
3	7	0	0	0	1	1	1
	13	0	0	1	1	0	1
	25	0	1	1	0	0	1
	28	0	1	1	1	0	0
	37	1	0	0	1	0	1
	42	1	0	1	0	1	0
	44	1	0	1	1	0	0
4	56	1	1	1	0	0	0
	23	0	1	0	1	1	1
	29	0	1	1	1	0	1
	46	1	0	1	1	1	0
	57	1	1	1	0	0	1
5	60	1	1	1	1	0	0
	55	1	1	0	1	1	1
	61	1	1	1	1	0	1

2-Cell Combination

Combination	A	B	C	D	E	F
(0, 8)	0	0	–	0	0	0
(8, 9)	0	0	1	0	0	–
(8, 12)	0	0	1	–	0	0
(8, 24)	0	–	1	0	0	0
(8, 40)	–	0	1	0	0	0
(5, 7)	0	0	0	1	–	1
(5, 13)	0	0	–	1	0	1
(5, 37)	–	0	0	1	0	1
(9, 13)	0	0	1	–	0	1
(9, 25)	0	–	1	0	0	1
(12, 13)	0	0	1	1	0	–
(12, 28)	0	–	1	1	0	0

(24, 25)	0	1	1	0	0	-
(24, 28)	0	1	1	-	0	0
(24, 56)	-	1	1	0	0	0
(40, 42)	1	0	1	0	-	0
(40, 44)	1	0	1	-	0	0
(40, 56)	1	-	1	0	0	0
(7, 23)	0	-	0	1	1	1
(13, 29)	0	-	1	1	0	1
(25, 29)	0	1	1	-	0	1
(25, 57)	-	1	1	0	0	1
(28, 29)	0	1	1	1	0	-
(28, 60)	-	1	1	1	0	0
(42, 46)	1	0	1	-	1	0
(44, 46)	1	0	1	1	-	0
(44, 60)	1	1	1	-	0	0
(56, 57)	1	1	1	0	0	-
(56, 60)	1	1	1	-	0	0
(23, 55)	-	1	0	1	1	1
(29, 61)	-	1	1	1	0	1
(57, 61)	1	1	1	-	0	1
(60, 61)	1	1	1	1	0	-

4 Cell Combination

Combination	A	B	C	D	E	F
(9, 13, 25, 29)	0	0	-	-	0	1
(12, 13, 28, 29)	0	-	1	1	0	-
(24, 25, 28, 29)	0	1	1	-	0	-
(24, 25, 56, 57)	-	1	1	0	0	-
(24, 28, 44, 60)	-	1	1	-	0	0
(24, 28, 56, 60)	-	1	1	-	0	0
(40, 42, 44, 46)	1	0	1	-	-	0
(25, 29, 57, 61)	-	1	1	-	0	1
(28, 29, 60, 61)	-	1	1	1	0	-

Prime Implicant Table

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

$$F(A, B, C, D, E, F) = (00 - 000) + (-00101) + (101- -0)$$

$$F(A, B, C, D, E, F) = \overline{ABDEF} + \overline{BCDEF} + A\overline{BCF}$$

1.28 BASIC LOGIC GATES

There are three basic logic gates each of which performs a basic logic function, they are called NOT, AND and OR. All other logic functions can ultimately be derived from combinations of these three. For each of three basic logic gates a summary is given including the **logic symbol**, the corresponding **truth table** and the **Boolean expression**.

1.28.1 The NOT gate

The NOT gate is unique in that it only has one input. The logic symbol of NOT gate is shown in **Figure 1.28**.

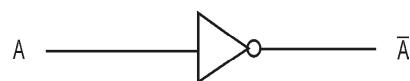


Fig. 1.28: Logic Symbol

The input to the NOT gate **A** is **inverted** i.e., the binary input state of 0 gives an output of 1 and the binary input state of 1 gives an output of 0.

\bar{A} is known as “NOT A” or alternatively as the **complement** of **A**.

The truth table for the NOT gate appears as below:

TRUTH TABLE

A	A-bar
0	1
1	0

1.28.2 The AND gate

The AND gates has two or more inputs. The output from the AND gate is 1 if and only if all of the inputs are 1, otherwise the output from the gate is 0. The AND gate is drawn as shown in **Figure 1.29**.

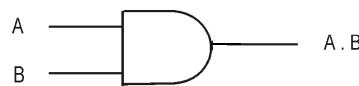


Fig. 1.29 : Logic Symbol

The output from the AND gate is written as **A . B**

The truth table for a two-input AND gate is given below:

TRUTH TABLE

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

1.28.3 The OR gate

The OR gate has two or more inputs. The output from the OR gate is 1 if any of the inputs is 1. The gate output is 0 if and only if all inputs are 0. The OR gate is drawn as shown in **Figure 1.30**.

The output from the OR gate is written as $A + B$.

The truth table for a two-input OR gate is given below:

TRUTH TABLE		
A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1

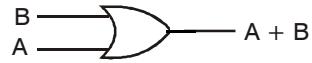


Fig. 1.30 : Logic Symbol

1.29 OTHER LOGIC GATES

The three basic logic gates can be combined to provide more complex logical functions. Four important logical functions are described here, namely NAND, NOR, XOR and XNOR. In each case a summary is given including the **logic symbol** for that function, the corresponding **truth table** and the **Boolean expression**.

1.29.1 The NAND gate

The NAND gate has two or more inputs. The output from the NAND gate is 0 if and only if all of the inputs are 1 otherwise the output is 1. Therefore the output from the NAND gate is the NOT of A AND B (also known as the **complement** or **inversion** of $A \cdot B$). The NAND gate is shown in **Figure 1.31** where the small circle immediately to the right of the gate on the output line is known as an **invert bubble**.

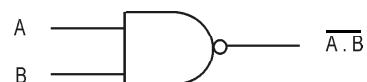


Fig. 1.31: Logic Symbol

The output from the NAND gate is written as $\overline{A \cdot B}$. The Boolean expression $\overline{A \cdot B}$ reads as “A NAND B”. The truth table for a two-input NAND gate is given below:

TRUTH TABLE

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

1.29.2 The NOR gate

The NOR gate has two or more inputs. The output from the NOR gate is 1 if and only if all of the inputs are 0, otherwise the output is 0. This output behaviour is the NOT of A OR B. The NOR gate is drawn as shown in **Figure 1.32**.

The output from the NOR gate is written as $A+B$ which reads “A NOR B”.

The truth table for a two-input NOR gate is given below:

TRUTH TABLE

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

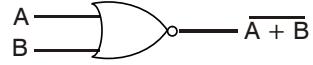


Fig. 1.32: NOR-Logic Symbol

1.29.3 The Exclusive-OR (XOR) gate

The exclusive-OR or XOR gate has two or more inputs. For a two-input XOR the output is similar to that from the OR gate except it is 0 when the both inputs are 1. This cannot be extended to XOR gates comprising 3 or more inputs however.

In general, an XOR gate gives an output value of 1 when there are an odd number of 1's on the inputs to the gate. The truth table for a 3-input XOR gate below illustrates this point.

The XOR gate is drawn as shown in **Figure 1.33**.

The output from the XOR gate is written as $A \oplus B$ which reads “A XOR B”.



Fig. 1.33 : XOR-Logic Symbol

The truth table for a two-input XOR gate looks like

TRUTH TABLE

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

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

For a 3-input XOR gate with inputs A , B and C the truth table is given by

A	B	C	$A \oplus B \oplus C$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

1.29.4 The Exclusive-NOR (XNOR) gate

The exclusive-NOR or XNOR gate has two or more inputs. The output is equivalent to inverting the output from the exclusive-OR gate described above. Therefore an equivalent circuit would comprise an XOR gate, the output of which feeds into the input of a NOT gate.

In general, an XNOR gate gives an output value of 1 when there are an even number of 1's on the inputs to the gate. The truth table for a 3-input XNOR gate below illustrates this point.

The XNOR gate is drawn using the same symbol as the XOR gate with an invert bubble on the output line as is illustrated in **Figure 1.34**.

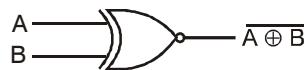


Fig. 1.34 : Logic Symbol

The output from the XNOR gate is written as $\overline{A \oplus B}$ which reads “A XNOR B”.

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

The truth table for a two-input XNOR gate looks like

TRUTH TABLE

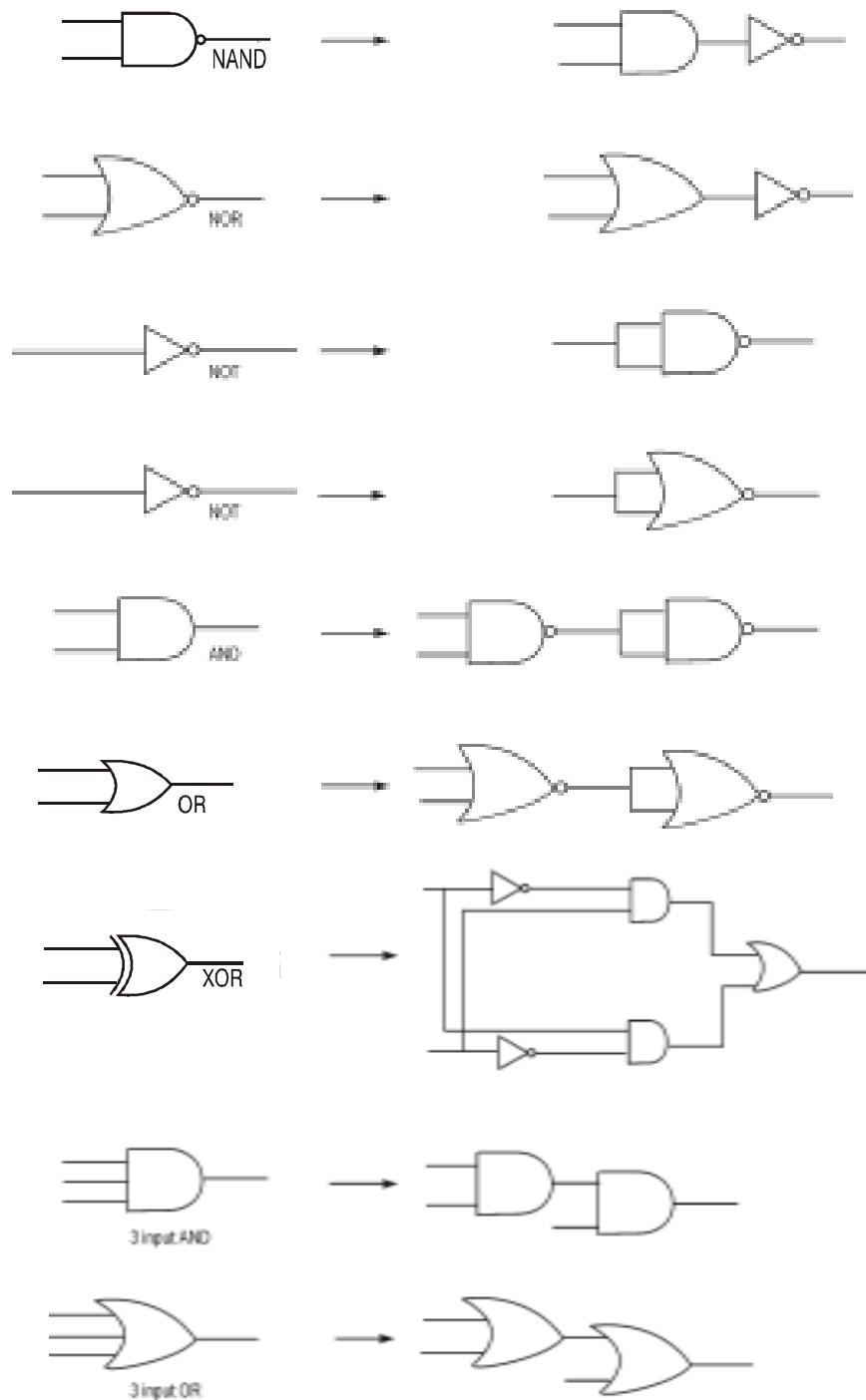
A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

For a 3-input XNOR gate with inputs A , B and C the truth table is given by

A	B	C	$\overline{A \oplus B \oplus C}$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

1.30 GATE CONVERSIONS

Any logic gate can be replaced by sets of other interconnected logic gates. Therefore any digital design can be implemented with a small number of logic gate types. The gate conversion circuits are shown in **Figure 1.35**.

**Fig. 1.35 : Gate Conversions**

1.31 DIGITAL ICs

1.31.1 14 Pin DIP

The 14 Pin DIP (Dual-in-line package) was one of the first types of Integrated Circuits (IC's) developed. The term dual-in-line package comes from the two parallel sets of pins that are situated across each other on the IC. The typical layout of a 14 pin DIP is shown in **Figure 1.36**.

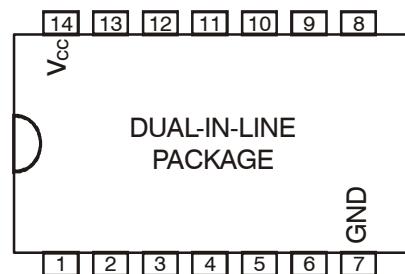


Fig. 1.36 : 14 Pin Dip

The pins are normally numbered counterclockwise with pin # 1 falling under the notch. For most IC's pin 7 is GND and pin 14 is Vcc. In order to utilize this IC, pin 14 must be connected to a supply voltage (usually 5V) and pin 7 must be grounded. Then the various gates within the IC may be used for analysis.

1.31.2 74X04 Hex Inverter

The 74X04 is a hex inverter. The 04 is the number that determines the chip type. It is a hex inverter because it contains 6 inverters on a single IC. The X is in place of specific features that the IC may contain. The most commonly listed special feature is the 74LS04, where LS stands for Low Power Shottky Transistors. The pin-out for the 74X04 is shown in **Figure 1.37**.

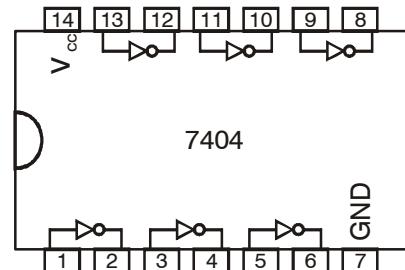


Fig. 1.37 : 7404 IC

1.31.3 74X08 Quad AND Gate

The 74X08 contains 4 AND gates. The 08 is the number that determines the chip type. It is called a quad AND gate because it contains 4 AND gates on a single IC. The pinout for the 74X08 is shown in **Figure 1.38**.

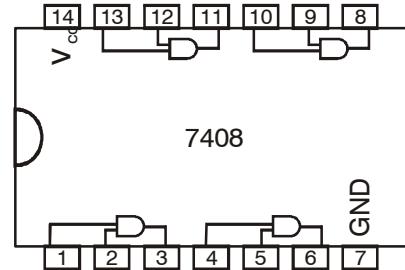


Fig. 1.38 : IC 7408

1.31.4 74X32 Quad OR Gate

The 74X32 contains 4 OR gates. The 32 is the number that determines the chip type. It is called a quad OR gate because it contains 4 OR gates on a single IC. The pinout for the 74X32 is shown in **Figure 1.39**.

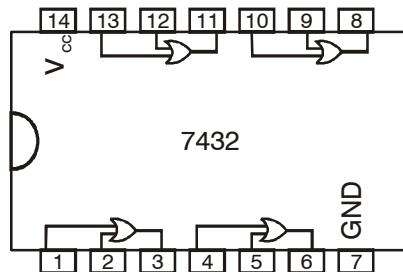


Fig. 1.39 : IC 7432

1.31.5 74X00 Quad NAND Gate

The 74X00 contains 4 NAND gates. The 00 is the number that determines the chip type. It is called a quad NAND gate because it contains 4 NAND gates on a single IC. The pin-out for the 74X00 is shown in **Figure 1.40**.

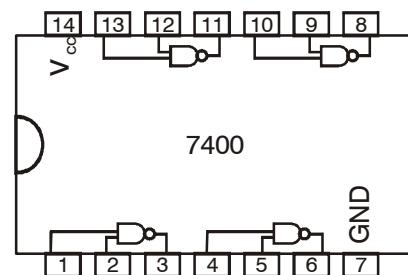


Fig. 1.40 : IC 7400

1.32 UNIVERSAL GATES

1.32.1 NAND GATE AS A UNIVERSAL GATE

The NAND gate is said to be a universal gate because any digital system can be implemented with it. The implementation of AND, OR and NOT operations with NAND gates is shown in **Figure 1.41** and EX-OR operations with NAND gates is shown in **Figure 1.42**.

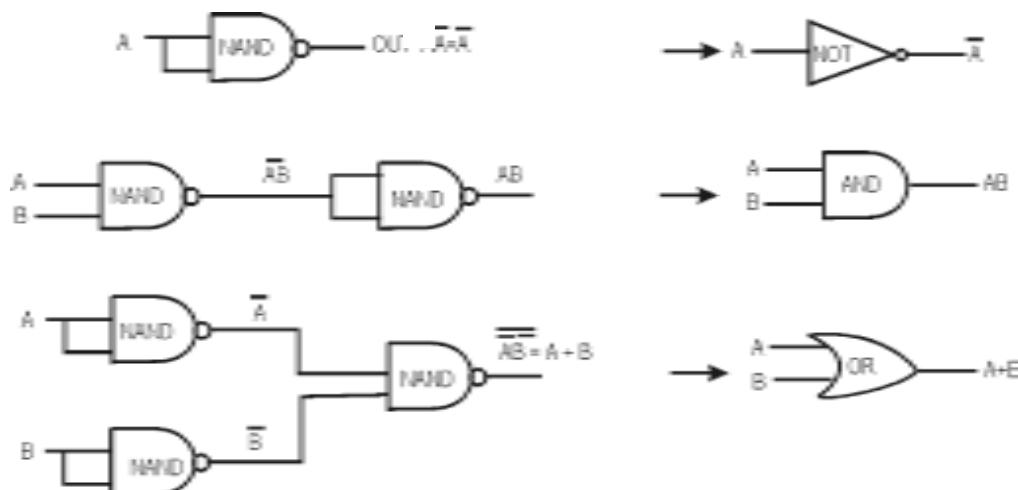


Fig. 1.41 : Implementation of basic gates using NAND

Exclusive OR with NAND

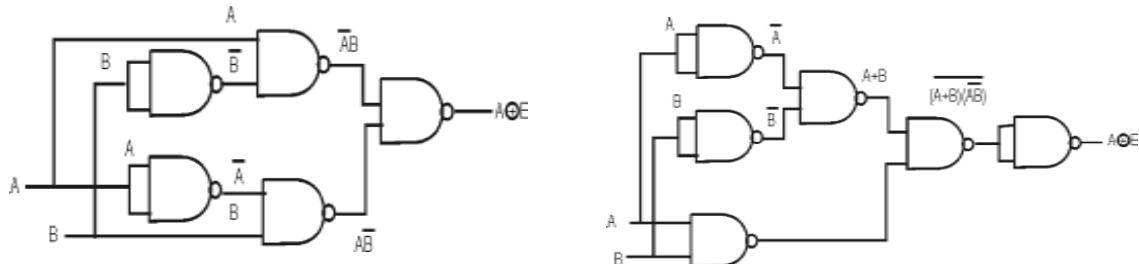


Fig. 1.42 : Implementation of EX-OR gate

1.32.2 NOR GATE AS A UNIVERSAL GATE

The NOR gate is called a universal gate because combinations of it can be used to accomplish all the basic functions. The implementation of basic gates using NOR gate is shown in **Figure 1.43**.

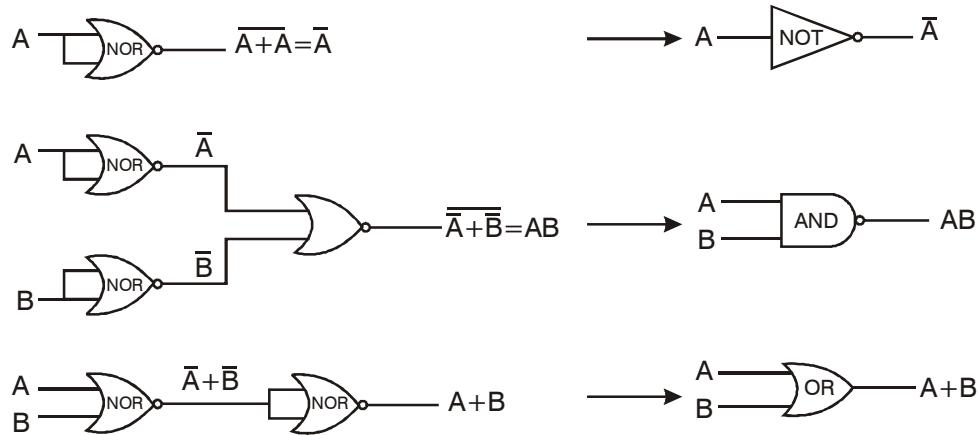


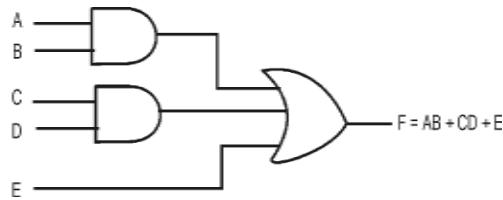
Fig. 1.43 : Implementation of Basic gates using NOR

1.33 IMPLEMENTATION OF LOGIC FUNCTIONS USING GATES

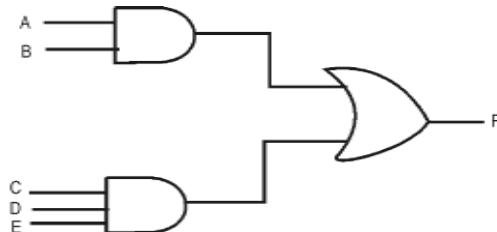
If the operation of the circuit is defined by a logic function, a logic circuit can be implemented directly from that function. For example, suppose we need a circuit that is defined by $X = ABC$. Then we immediately know that all that is needed is a 3 input AND gate. If we need a circuit that is defined by $X = A + \bar{B}$, we will use a two input OR gate with an inverter on one of the inputs (\bar{B}). The same reasoning used for these examples for these simple cases can be extended to more complex logic circuits.

Example 1.161: Draw the logic circuit using gate for the given function: $F = AB + CD + E$

Solution

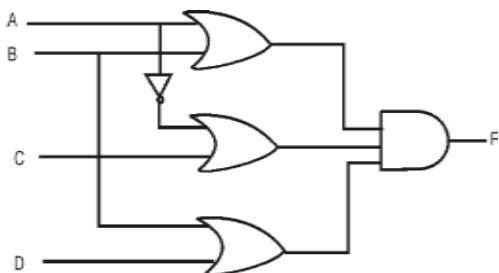


Example 1.162: Draw the logic circuit for $F = AB + CDE$



Example 1.163: Draw the logic circuit using basic gates for the function

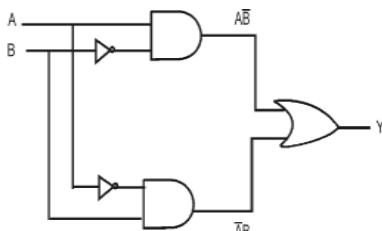
$$F = (A + B)(\bar{A} + C)(B + D)$$



Example 1.164: Draw the logic circuit using basic gates for EX-OR gate and EX-NOR gate.

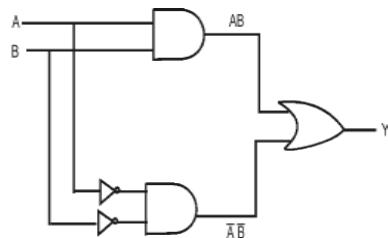
Solution: (i) The Boolean expression for EX-OR gate is

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



(ii) The Boolean expression for EX-NOR gate is,

$$Y = \overline{A \oplus B} = AB + \bar{A}\bar{B}$$



Example 1.165: Draw the logic circuit using basic gates for the function $F = AB(C\bar{D} + EF)$

Solution: $F = AB(C\bar{D} + EF) = ABC\bar{D} + ABEF$ (SOP form)

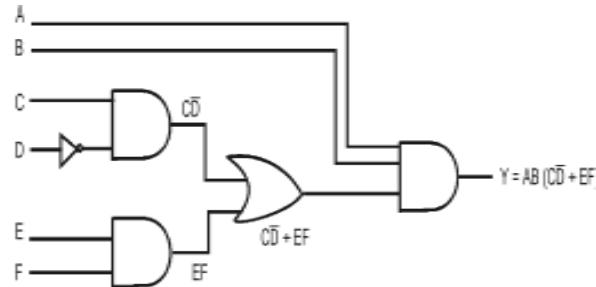


Fig. 1.44 : Logic circuit for $Y=AB(C\bar{D}+EF)$

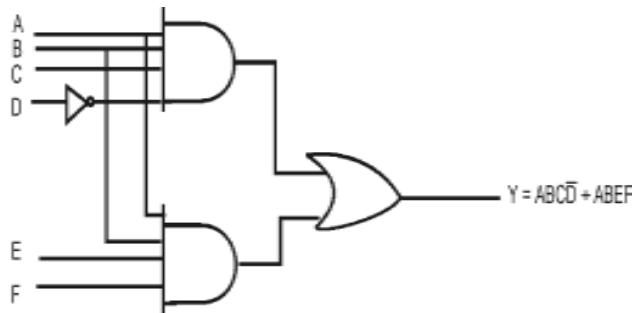
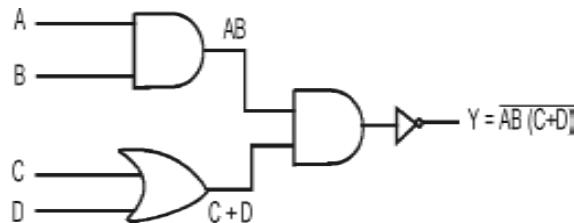


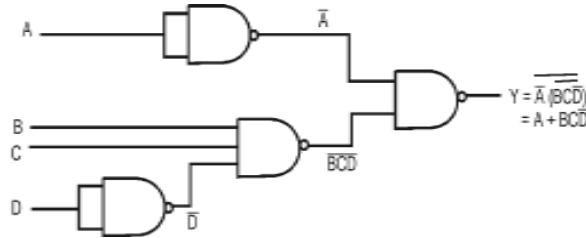
Fig. 1.45 : Logic circuit for $Y=ABC\bar{D}+ABEF$

Example 1.166: Draw the logic diagram for $Y = \overline{AB(C+D)}$

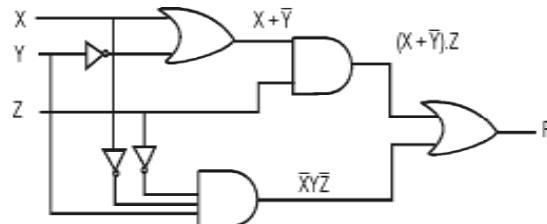


Example 1.167: Draw the logic circuit using NAND gates for

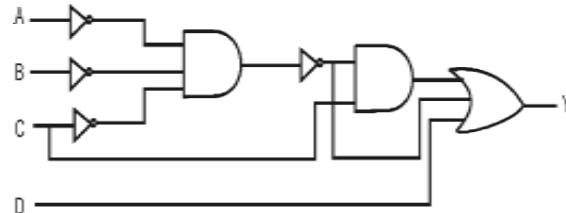
$$Y = A + BC\bar{D}$$



Example 1.168: Draw the logic diagram for the function $F = (X + \bar{Y}) \cdot Z + \bar{X}Y\bar{Z}$.



Example 1.169: Reduce the given combinational logic circuit to a minimum form.



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

Applying DeMorgan's theorem an Boolean algebra,

$$\begin{aligned}
 Y &= (\overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}})C + \overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}} + D \\
 &= AC + BC + CC + A + B + C + D \\
 &= AC + BC + C + A + B + C + D \\
 &= C(A + B + 1) + A + B + D \quad (C + C = C) \\
 &= C + A + B + D \quad (A + B + 1 = 1) \\
 &= A + B + C + D
 \end{aligned}$$

The simplified circuit is,



Example 1.170:

Consider the combinational circuit shown in figure.

- (i) Derive the Boolean expressions for T_1 through T_4 . Evaluate the outputs F_1 and F_2 as a function of the four inputs.

$$T_1 = \overline{BC}$$

$$T_2 = \overline{AB} \quad F_1 = T_3 + T_4$$

$$T_3 = T_1 + A \quad F_2 = T_2 + D$$

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

$$T_4 = T_2 \oplus D$$

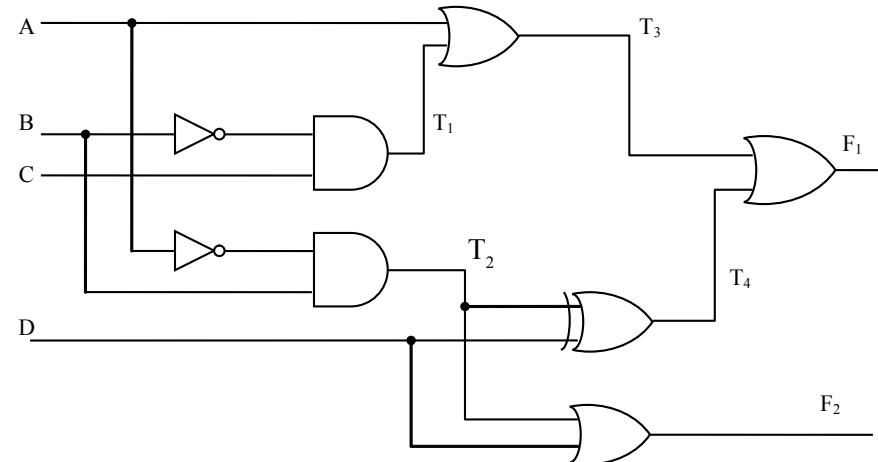
$$= \overline{ABD} + AD + \overline{BD}$$

- (ii) List the truth table with 16 binary combinations of the four input variables. Then list the binary values for T_1 through T_4 and outputs F_1 and F_2 in the table.

A	B	C	D	T_1	T_2	T_3	T_4	F_1	F_2
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	1	1
0	0	1	0	1	0	1	0	1	0
0	0	1	1	1	0	1	1	1	1
0	1	0	0	0	1	0	1	1	1
0	1	0	1	0	1	0	0	0	1
0	1	1	0	0	1	0	1	1	1
0	1	1	1	0	1	0	0	0	1
1	0	0	0	0	0	1	0	1	0
1	0	0	1	0	0	1	1	1	1

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

- (iii) Plot the output Boolean function obtained in part (ii) on maps and show that simplified Boolean expressions are equivalent to the ones obtained in Part (i)



$$F_1 = T_3 + T_4$$

$$= T_1 + A + T_2 \oplus D$$

$$= \overline{B}C + A + [(\overline{A}B) \oplus D]$$

$$= A + \overline{B}C + \overline{A}\overline{B}\overline{D} + D[\overline{\overline{A}B}]$$

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

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

$$F_2 = T_2 + D$$

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

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

$$F_2 = T_2 + D$$

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

1.34 NAND AND NOR IMPLEMENTATIONS

The NAND and NOR gates are the universal gates. NAND and NOR gates can be used to produce the AND, OR and NOT functions. Also, NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families.

When implementing any Boolean expression, it involves various logic gates and it needs various standard ICs. But all gates within the standard ICs are not utilized. For example to implement Boolean expression $\overline{AB} + CD$ we require two AND gates, one OR gate and one NOT gate. This requires 3 standard ICs. But gates utilized from ICs are very less. To improve utilization of ICs and reduce number of ICs required, NAND/NOR gates are used to implement Boolean expression. For this, conversion of AND/OR/NOT gates into NAND/NOR gates is needed.

The graphic symbols for NAND gate are shown in **Figure 1.46** and the graphic symbols for NOR gate are shown in **Figure 1.47**. A one-input NAND or NOR gate behaves like an inverter as shown in **Figure 1.48**.

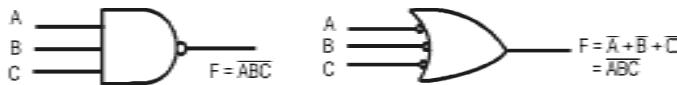


Fig. 1.46 : Graphic symbol for NAND gate

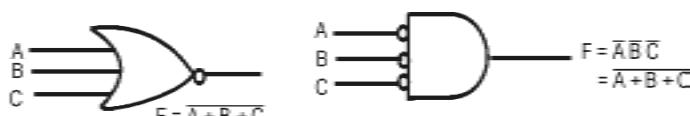


Fig. 1.47 : Graphic symbol for NOR gate



Fig. 1.48 : Graphic symbol for NOT gate

The rules for obtaining NAND/NOR Logic:

1. Simplify the function and express it in sum of products (AND/OR logic).
2. For NAND implementation, add bubbles on the output of each AND gate and add bubbles on the input side to all OR gates.

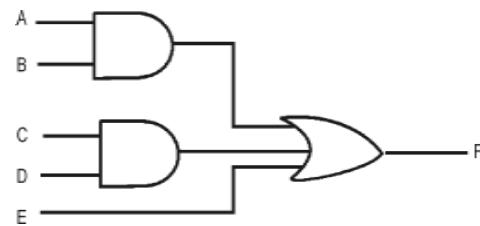
3. For NOR implementation, add bubbles on the output of each OR gate and add bubbles on the input side of all AND gates.
4. Add or subtract an inverter on each line that received a bubble in step 2 or 3.
5. Replace bubbled OR by NAND and bubbled AND by NOR.
6. Eliminate double inversions.

1.34.1 NAND gate implementation

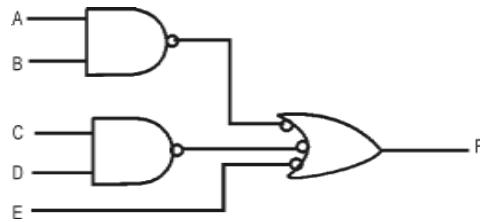
Example 1.171: Implement the Boolean function with NAND gates. $F = AB + CD + E$

Solution:

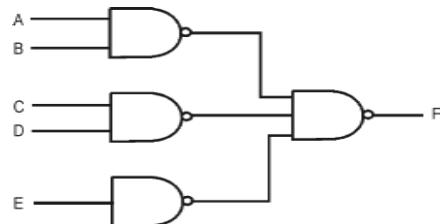
Step 1: Draw AND-OR circuit.



Step 2: Add bubbles on output of each AND gate and input of OR gate.



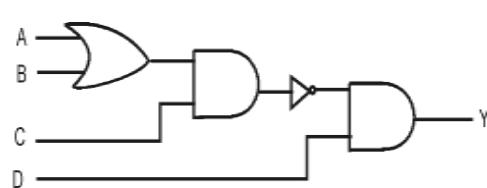
Step 3: Replace other gates by NAND gates.



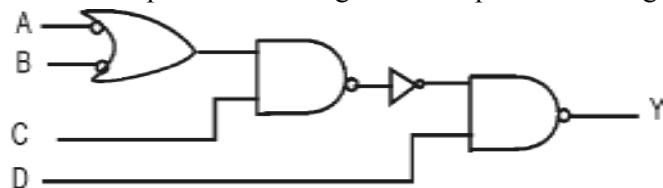
Example 1.172: Implement the Boolean expression with NAND gates $Y = \overline{((A+B)\bar{C})}D$

Solution: Step 1: Draw original logic diagram for

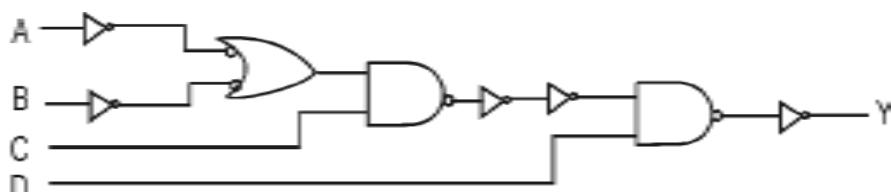
$$Y = \overline{((A+B)\bar{C})}D$$



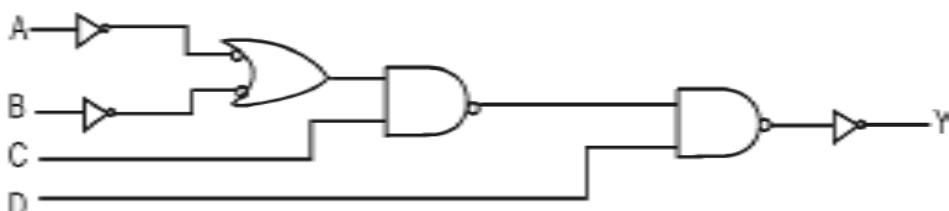
Step 2: Add bubbles on the output of the AND gates and input of the OR gate.



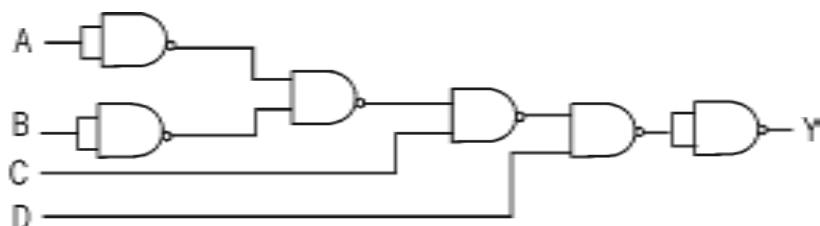
Step 3: Add inverters on each line that received a bubble.



Step 4: Eliminate double inversions.



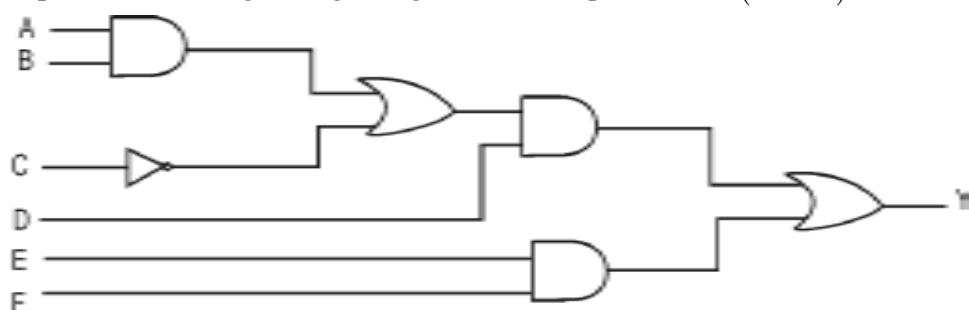
Step 5: Replace the other gates by only NAND gates.



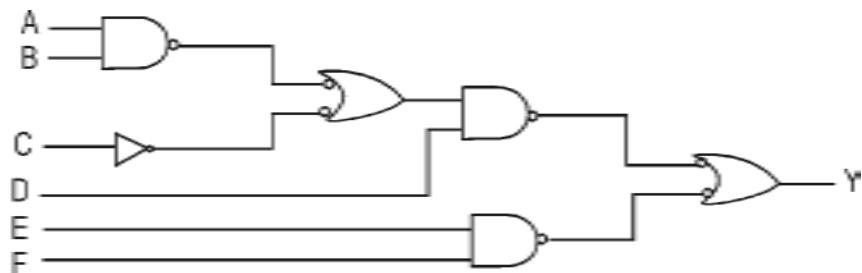
Example 1.173: Implement NAND gates for $Y = (AB + \bar{C})D + EF$.

(April 2004)

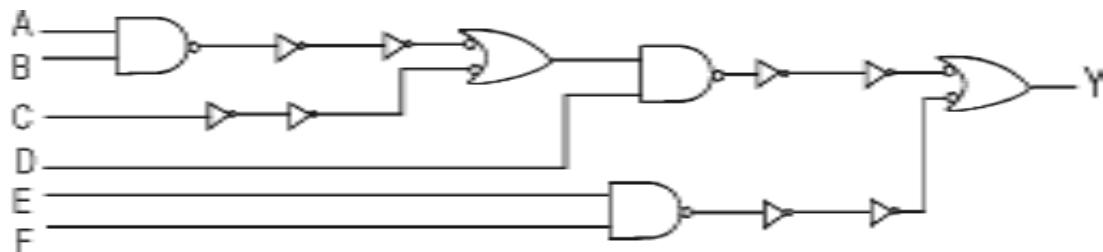
Solution: **Step 1:** Draw the original logic diagram for the expression $Y = (AB + \bar{C})D + EF$



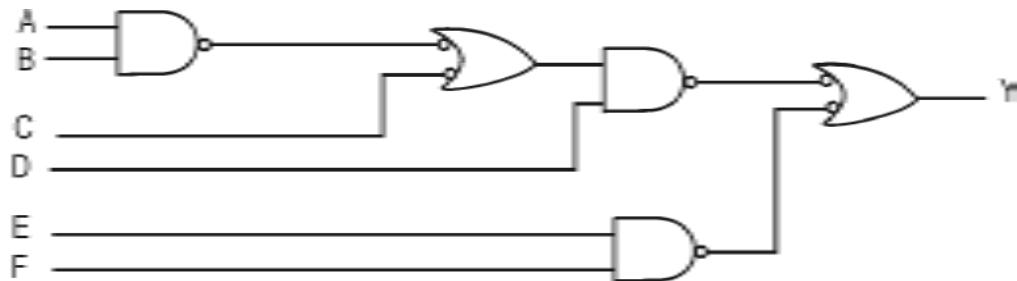
Step 2: Add bubbles on the output of the AND gates and input of the OR gates.



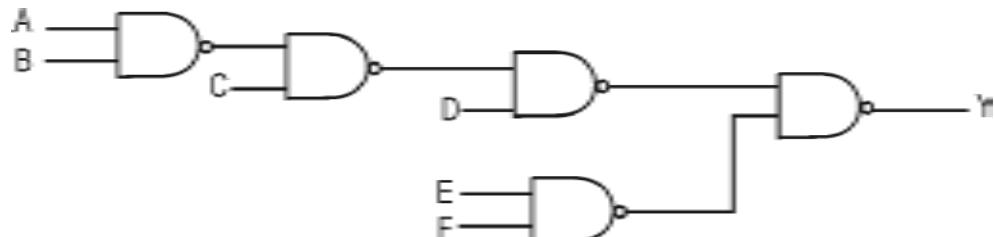
Step 3: Add inverters on each line that received a bubble.



Step 4: Eliminate double inversions.



Step 5: Draw the circuit with only NAND gates.



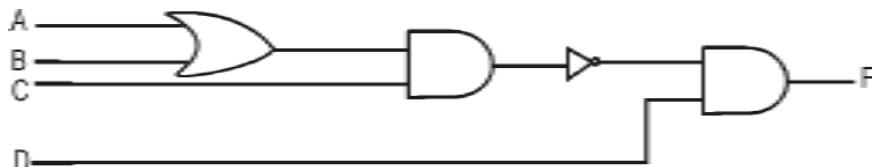
1.34.2 NOR gate implementation

Example 1.174: Implement the Boolean expression with NOR gates.

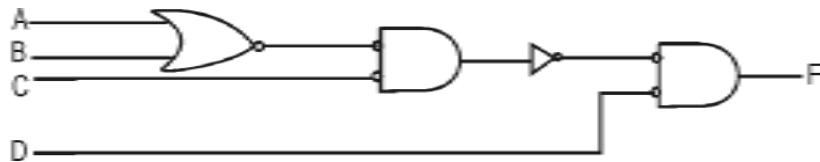
$$F = \overline{(A + B)C}D$$

Solution:

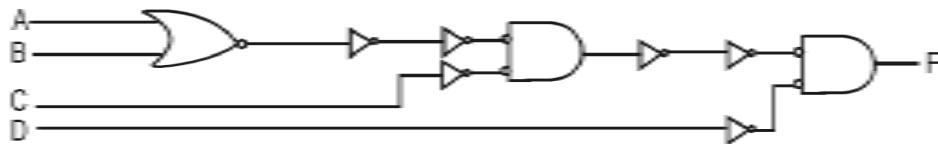
Step 1: Draw the original logic diagram for the given Boolean expression,



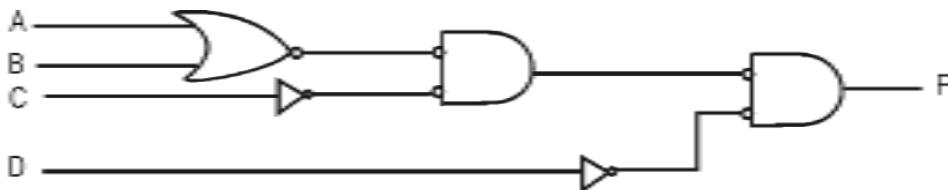
Step 2: Add bubbles on output of each OR gate and add bubbles on input of each AND gate.



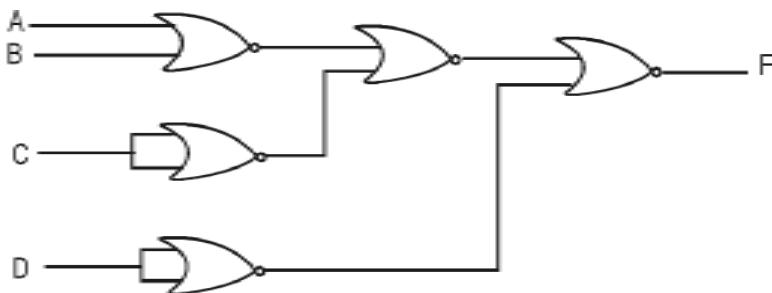
Step 3: Add inverters on each line that received bubbles.



Step 4: Eliminate double inversions.



Step 5: Draw the NOR diagram with one graphic symbol.

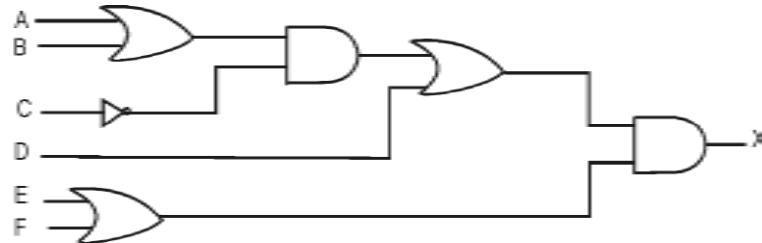


Example 1.175: Draw the multi level NOR circuit for the Boolean expression:

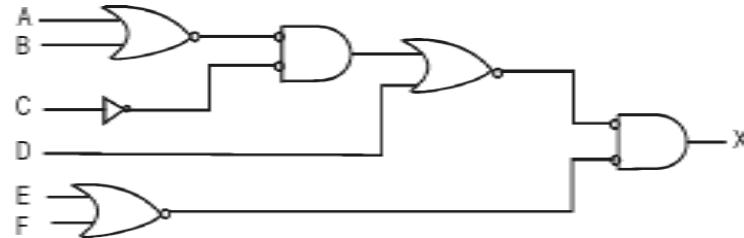
$$X = [(A+B)\bar{C} + D](E+F)$$

Solution:

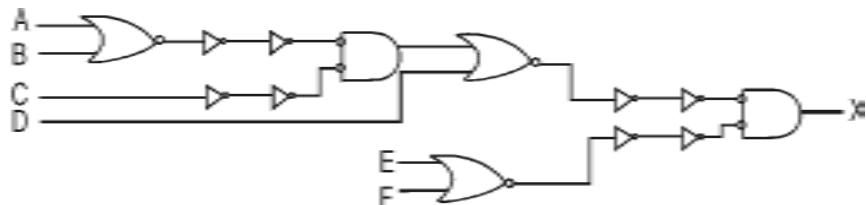
Step 1: Draw the original circuit diagram for $X = [(A+B)\bar{C} + D](E+F)$



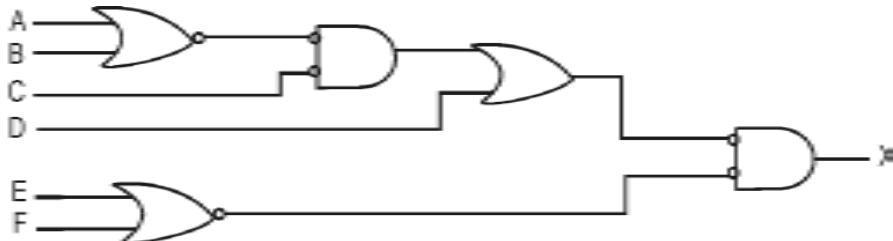
Step 2: Add bubbles on the output of OR gates and add bubbles on the input of AND gates.



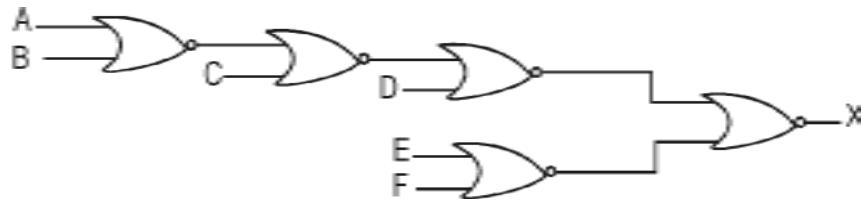
Step 3: Add inverters on each line that received bubbles.



Step 4: Eliminate Double Versions.



Step 5: Draw the NAND diagram using one graphic symbol.



1.35 MULTI LEVEL GATE IMPLEMENTATION

The maximum number of gates cascaded in series between a network input and the output is referred to as the number of levels of gates.

A function written in SOP form or in POS form corresponds directly to a 2-level gate network. AND-OR network means a 2-level network composed of a level of AND gates followed by an OR gate at the output. OR-AND network mean a 2-level network composed of a level of OR gates followed by an AND gate at the output. OR-AND-OR network means a 3-level network. The number of levels in an AND-OR network can usually be increased by factoring SOP expression from which it was derived. Similarly, the number of levels in an OR-AND network can usually be increased by multiplying out some of the terms in POS expression from which it was derived.

Logic designers are concerned with the number of levels in a network for several reasons. Sometime factoring or multiplying to increase the number of levels of gates. This will reduce the required number of gates and gate inputs and thus reduce the cost of the network. In other cases, increasing the number of levels will increase the cost. Now we will study some examples for 2 level, 3 level and 4 level networks.

1.35.1 Two Level AND-OR Network

The two level AND-OR network for the Boolean expression, $F = \overline{A}\overline{C}D + B\overline{C}D + B\overline{C}\overline{D} + A\overline{C}\overline{D}$ is shown in **Figure 1.49**.

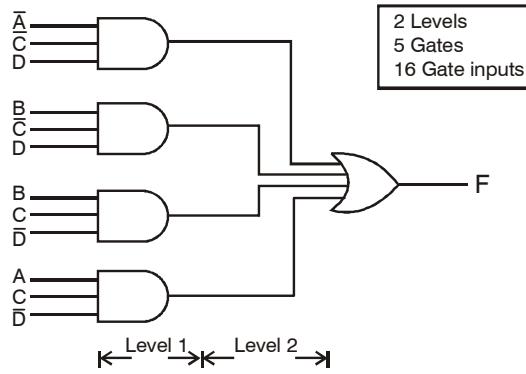


Fig. 1.49 : Two Level AND-OR

1.35.2 2-Level OR-AND Network

The two-level OR-AND network for the Boolean expression

$$F = (C+D)(\bar{A}+B+C)(\bar{C}+\bar{D})(A+B+\bar{C}) \text{ is shown in Figure 1.50.}$$

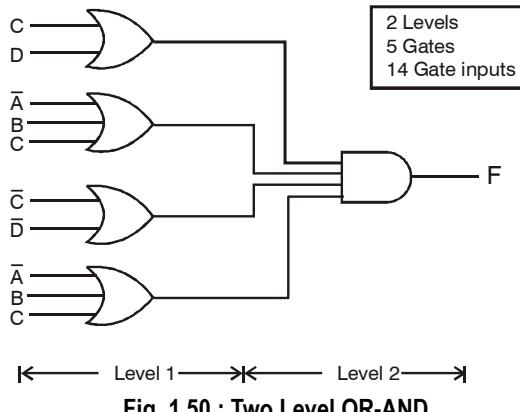


Fig. 1.50 : Two Level OR-AND

1.35.3 Two Level NAND-NAND Network

The two level NAND-NAND network for Boolean expression

$$F = [\bar{A} \cdot (\bar{B}+C) \cdot (B+\bar{C}+\bar{D})] \text{ is shown in Figure 1.51.}$$

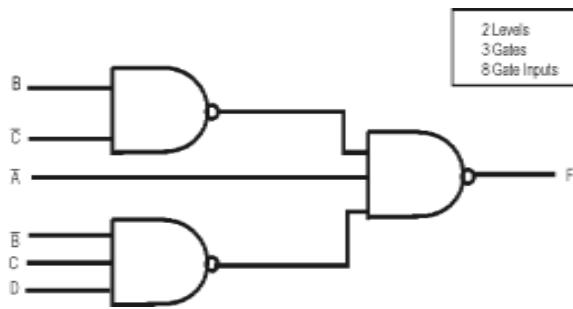


Fig. 1.51 : Two-Level NAND-NAND

1.35.4 Three Level AND-OR-AND Network

The three level AND-OR-AND network for the expression

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

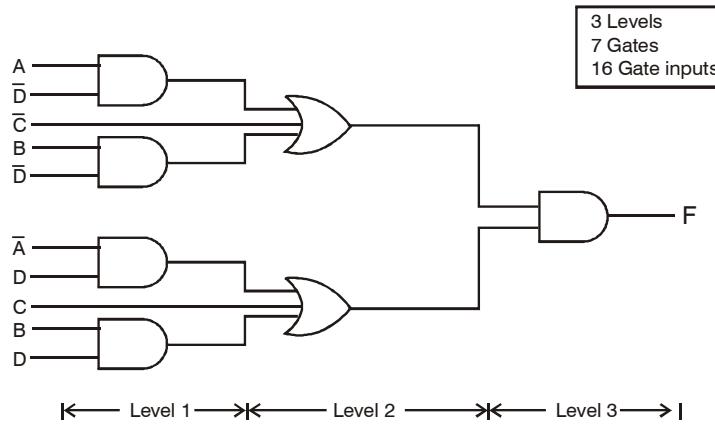


Fig. 1.52 : Three level AND-OR-AND

1.35.5 Four level Gate Network

The four level gate network is shown in **Figure 1.53**.

$$Y = (AB + C) [(D+E) + (F.G)] + H$$

This network will have 4 levels, 6 Gates and 13 gate inputs

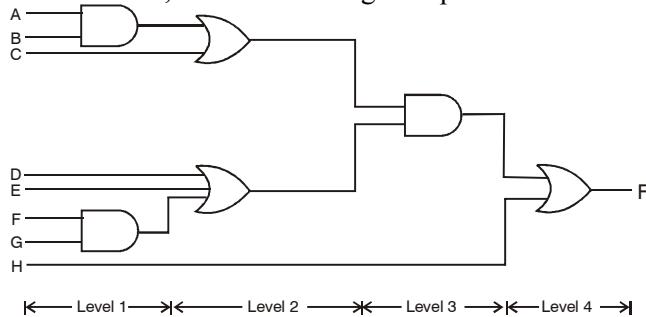


Fig. 1.53 : Four level gate Network

1.35.6 Design of Multilevel Gate Network

The design of multi level gate implementation using NAND gates is explained in **Examples 1.171** and **1.172** and the design of multi level gate implementation using NOR gates is already explained in **Example 1.173** and **1.174**.

1.36 MULTI OUTPUT GATE IMPLEMENTATION

Most practical combinational logic circuits require more than one output. Decoders, ROM, PLD devices are some examples for multi output gate network. These multi outputs are derived from the same input variables. We can simplify each output function separately by the use of Karnaugh Maps

or Tabulation Methods. It is quite possible that the simplified output functions may have common terms. The common term used by one output function can be shared by other output functions. This sharing of common terms reduces the total number of gates.

Example 1.176: Draw the multi output gate network for the Boolean expressions,

$$F_1 = A\bar{B} + \bar{A}B, F_2 = BC + \bar{A}B$$

Solution:

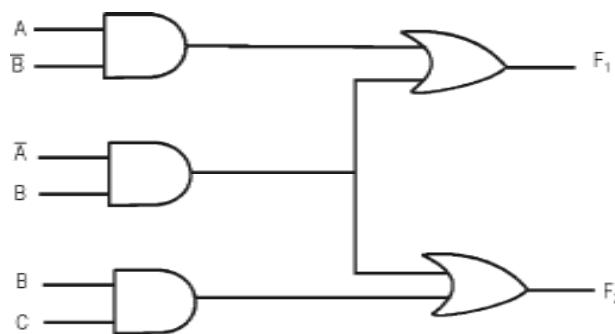


Fig. 1.54 : Multi output gate network

1.37 LOGIC FAMILIES AND DIGITAL IC's

1.37.1 Logic Family

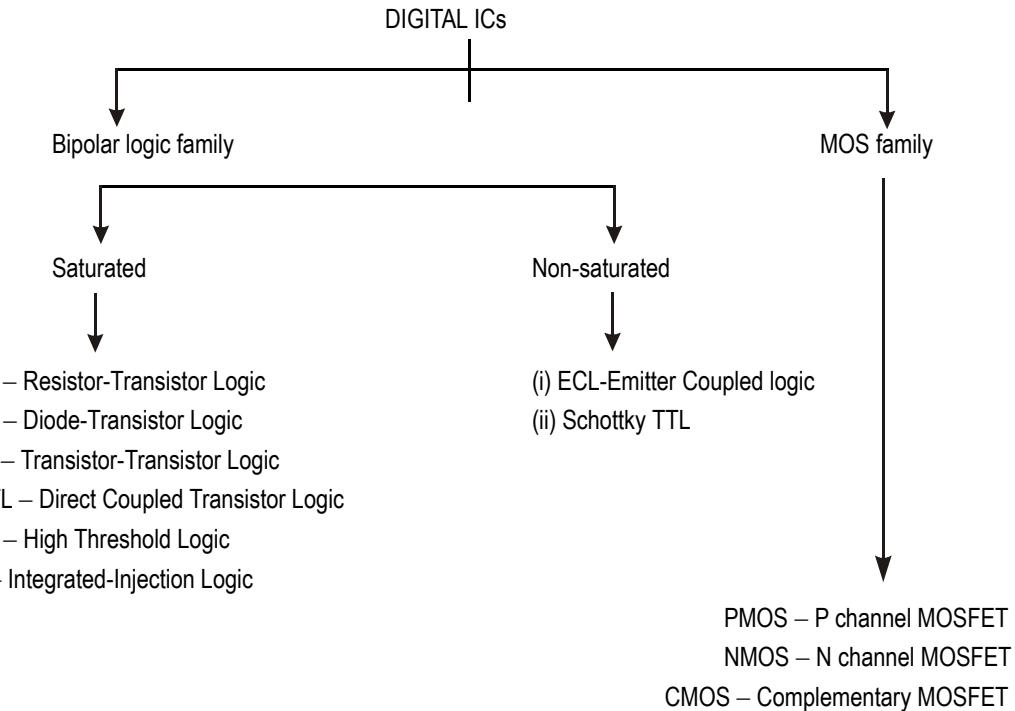
It is a collection of different IC chips that have similar input, output and internal circuit characteristics; but that perform different logic functions.

1.37.2 Integrated circuits

SSI	–	less than 10 gates
MSI	–	10–100 gates/chip
LSI	–	100–5000 gates/chip
VLSI	–	above 5000 gates/chip

ICs are classified into

- (i) Linear IC – operate with continuous signal.
- (ii) Digital IC – operate with binary signals and are invariably constructed with ICs.



1.38 CHARACTERISTICS OF DIGITAL ICs

- ◆ Propagation Delay
- ◆ Power Dissipation
- ◆ Fan-in
- ◆ Fan-out
- ◆ Noise Margin
- ◆ Operating temperature
- ◆ Power supply requirements.

1.38.1 Propagation Delay

Speed of operation expressed in terms of propagation delay. It is defined as the time taken for the output of a gate to change after the inputs have changed. It is measured in ‘ns’.

The delays for a standard TTL gate are

$$t_{PHL} = 7 \text{ ns} \quad ; \quad t_{PLH} = 11 \text{ ns}$$

$$\text{Average propagation delay} = \frac{7+11}{2} = 9 \text{ ns}$$

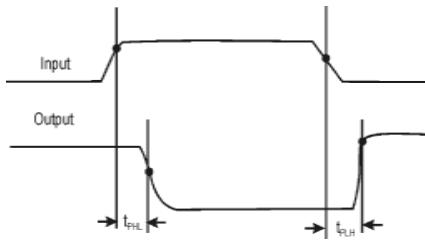


Fig. 1.55 : Measurement of Propagation Delay

1.38.2 Power Dissipation

Power consumed by the logic gate when fully driven by all its inputs.

It is expressed in milliwatts (mw) or nanowatts (nw).

For a standard TTL NAND gate,

I_{CCH} – Current, when the output is in high-voltage level – 4 mA

I_{CCL} – Current, when the output is in low-voltage level – 2mA

$$I_{CC(\text{avg})} = I_{CCH} + I_{CCL} / 2 = (4 + 2) / 2 = 3 \text{ mA}$$

$$P_{D(\text{avg})} = I_{CC(\text{avg})} \times V_{CC}$$

$$P_{D(\text{avg})} = 3 \text{ mA} \times 5\text{V} = 15 \text{ mw}$$

The average power dissipation = 15 mw

An IC has four NAND gates dissipates a total of $15 \times 4 = 60$ mw.

1.38.3 Fan-In

Number of inputs connected to the gate without any degradation in the voltage level.

Fan-in determines functional capabilities of a logic circuit.

Example: An eight input gate requires one unit load (UL) per input. It's fan-in is 8.

1.38.4 Fan-Out

The fan-out of a gate specifies the number of standard loads that can be connected to the output of the gate without degrading its normal operation.

The fan-out is the maximum number of inputs that can be connected to the output of a gate, and is expressed by a number.

The output of the gate is in the high voltage level in **Figure 1.56**. It provides a current source I_{OH} to all the gate inputs connected to it. Each gate input requires a current I_{IH} for proper operation.

Similarly the output of the gate is in the low voltage level provides a current sink I_{OL} for all the gate inputs connected to it. Each gate input supplies a current I_{IL} .

$$\text{Fan out} = \frac{I_{OH}}{I_{IH}} \text{ or } \frac{I_{OL}}{I_{IL}}$$

Example 1.177: A standard TTL gates have the following currents.

$$I_{OH} = 400 \mu\text{A} \quad I_{IH} = 40 \mu\text{A} \quad ; \quad I_{OL} = 16 \text{ mA} \quad I_{IL} = 1.6 \text{ mA}$$

$$\text{Fan-out} = \frac{400}{40} = \frac{16}{1.6} = 10$$

Fan-out of standard TTL is 10. This means that the output of a TTL gate can be connected to no more than 10 inputs of other gates in the same logic family. Otherwise, the gate may not be able to drive or sink the amount of current needed from the inputs that are connected to it.

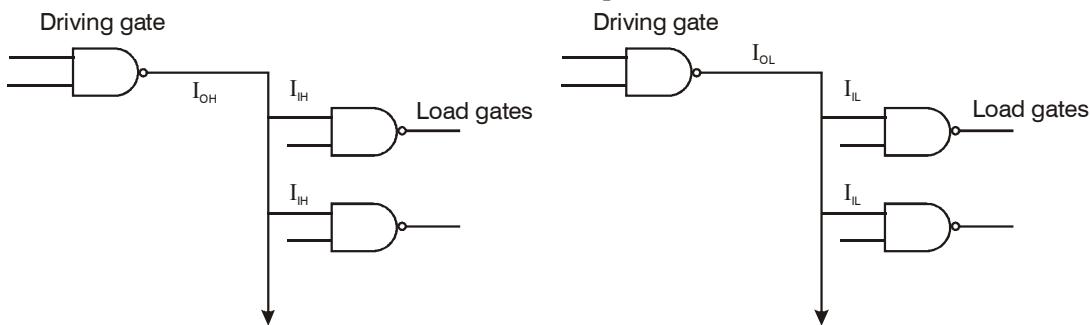


Fig. 1.56 : High Level Input

Fig. 1.57 : Low Level Output

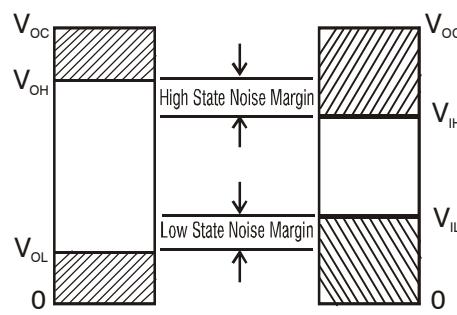
1.38.5 Noise Margin (or) Noise Immunity

Noise margin is the maximum noise voltage added to an input signal of a digital circuit that does not cause an undesirable change in the circuit output. It is expressed in Volts.

The input voltage range and output voltage range for evaluating noise margin are shown in **Figure 1.58**.

$$\text{High state Noise Margin} = V_{OH} - V_{IH} = 2.4 - 2 = 0.4 \text{ V}$$

$$\text{Low state Noise Margin} = V_{IL} - V_{OL} = 0.8 - 0.4 = 0.4 \text{ V}$$



(A) Output Voltage Range (B) Input Voltage Range

Fig. 1.58 : Signals for evaluating noise margin

ANNA UNIVERSITY QUESTIONS

BOOLEAN ALGEBRA

PART A

1. Why binary number system is used in digital system? (April 2003)
2. Represent the following numbers in 2's complement form:
+ 3, + 25, - 5, -11 (April 2003)
3. Define the laws of Boolean Algebra. (April 2003)
4. If A and B are boolean variables check if $(A \cdot \bar{A}) \oplus (A \oplus \bar{A}) = (B + \bar{B}) \oplus (\bar{B} \oplus B)$. (April 2003)
5. For a switching function of 'n' variables, how many distinct minterms and maxterms are possible?
(April 2003)
6. Find the octal equivalent of the decimal number 64. (April 2003)
7. Find the hexadecimal equivalent of the octal number 153.4. (April 2003)
8. Show that the excess-3 code is self complementing. (April 2003)
9. Convert $(2101)_3$ to base 5 number. (Nov. 2003)
10. Subtract the unsigned number $(10101)_2$ from $(11011)_2$ using one's complement and two's complement method. (Nov. 2003)
11. Define canonical form. Express $F = B\bar{C} + AC$ in a canonical SOP form. (Nov. 2003)
12. If A and B are Boolean variables and if $A = 1$ and $A + B = 0$, find B. (Nov. 2003)
13. Express the switching function $f_{(BA)} = A$ in terms of minterms. (Nov. 2003)
14. Determine the decimal equivalent of binary 0.1101. (April 2004)
15. How is the letter A coded as in the ASCII code? (April 2004)
16. Apply De Morgan's theorems to simplify $\overline{A + B\bar{C}}$. (April 2004)
17. Plot the expression on K-map: $F(w, x, y) = \sum (0, 1, 3, 5, 6) + d(2, 4)$ (April 2004)
18. Find the decimal equivalent of $(123)_9$. (April 2004)
19. Express $x + yz$ as the sum of minterms. (April 2004)
20. Convert the decimal number 214 to hexadecimal. (April 2004)
21. Define the following: minterm and maxterm. (April 2004)

22. Find the binary representation of decimal 125. (Nov. 2004)
23. Complement the expression: $X(\bar{Y} + \bar{Z})$. (Nov. 2004)
24. Find the octal equivalent of decimal 200. (Nov. 2004)
25. Determine the product of all 2^n maxterms of 'n' variables. (Nov. 2004)
26. Add $(1A8)_{16}$ and $(67B)_{16}$. (Nov. 2004)
27. State two absorption properties of Boolean Algebra. (Nov. 2004)
28. Perform 2's complement subtraction of 010110 - 100101. (Nov. 2004)
29. What is the advantage of biquinary code? (Nov. 2004)
30. Convert $(FACE)_{16}$ to base 8 number. (April 2005)
31. Add the decimals 67 and 54 using 8421 BCD code. (April 2005)
32. Simplify the following Boolean expression: $\bar{a}\bar{b}\bar{c} + a\bar{b}c + abc$. (April 2005)
33. What is prime implicant ? (April 2005)
34. Find the value of $X = \overline{ABC}(\overline{A+D})$ if A = 0; B = 1, C = 1 and D = 1. (April 2005)
35. State De Morgan's laws. (April 2005)
36. Simplify: $A + AB + \bar{A} + B$. (April 2005)
37. Realise OR gate using NAND gate. (Dec. 2005)
38. Show that $A + \bar{A} \cdot B = A + B$ using the theorems of Boolean algebra. (Dec. 2005)
39. Convert the following numbers with the indicated bases to decimal $(4310)_5$ and $(198)_{12}$. (Dec. 2005)
40. What bit must be complemented to change an ASCII letter from capital to lowercase and vice versa? (Dec. 2005)
41. Simplify the following Boolean expressions to a minimum number of literals:
 (a) $(X+Y)(X+\bar{Y})$ (b) $XY + \bar{X}Z + YZ$ (Dec. 2005)
42. What are prime implicants? (Dec. 2005)
43. What is the number of bits in ASCII code? What is the need for ASCII code? (Dec. 2005)
44. Prove that $A + A'B = A + B$, using Boolean Algebra. (Dec. 2005)
45. Add -176 (decimal) to -204 (decimal); do the arithmetic in binary using 2's complement notation. (Dec. 2005)
46. Construct the truth table of $F = (A \oplus B) \oplus (C \oplus D)$. (Dec. 2005)

47. State a single rule, which can be used to form the complement of a Boolean expression in one step. **(Dec. 2005)**
48. Find the minterm expansion of $f(a, b, c, d) = a'(b' + d) + acd'$. **(Dec. 2005)**
49. Explain how parity can be used for error detection. **(Dec. 2005)**
50. Find the 2's complement and 1's complement of 101101. **(May 2006)**
51. Simplify $x_1 + x_1 x_2$. **(May 2006)**
52. Find the standard sum for the following function:

$$f = x_1 x_2 x_3 + x_1 x_3 x_4 + x_1 x_2 x_4.$$
 (May 2006)
53. Convert 1110011 into hexadecimal through octal. **(May 2006)**
54. What is the feature of Gray code? **(May 2006)**
55. What is the range of values that can be represented using n-bit 2's complement form of representation? What is the corresponding range with n-bit 1's complement form? **(May 2006)**
56. Obtain the complement of $f = w \bar{x}y + x \bar{y} = wxz$ using De Morgan's theorem. **(May 2006)**
57. Convert the following number front one base to other
 $(354.52)_6 = ()_{10}$
 $(100)_{10} = ()_{16}$ **(Dec. 2006)**
58. What are the different ways to represent a negative number? **(Dec. 2006)**
59. Convert the binary number 1011_2 to gray code. **(May 2007)**
60. Convert $(AF3.15)_{16}$ to base 10. **(May 2007)**
61. Plot the following function by using a K-map and determine its minterm and maxterm lists:

$$f = AB + BC'$$
 (May 2007)
62. Convert 1011101010_2 to octal and hexadecimal numbers. **(May 2007)**
63. What is the BCD equivalent for the Gray code 1110? **(May 2007)**
64. Expand the function $F(A,B,C) = A + B'C$ to standard SOP form. **(May 2007)**
65. Using K-map find minimum sum of products for the function $f(a,b,c) = \sum m(0,1,5,6,7)$
(May 2007)
66. What is the advantage of gray codes over the binary number sequence? **(May 2007)**
67. Simplify the following Boolean function:
(a) $x(x' + y)$
(b) $xy + x'z + yz$ **(May 2007)**

68. What are error detecting codes? (Dec. 2007)
69. Find the complements for the following functions.
- $F_1 = xy' + x'y$
 - $F_2 = (xy + y'z + xz)x$ (Dec. 2007)
70. What are the drawbacks of K-map method? (Dec. 2007)
71. What is the largest binary number that can be expressed with 12 bit? What is the equivalent decimal and hexadecimal? (Dec. 2008)
72. Simplify $(x+y)(x+y')$ to a minimum number of literals. (Dec. 2008)
73. Find the minterm of $xy + yz + xy'z$. (Dec. 2008)
74. Simplify the following Boolean expression to a minimum number of literals:

$$\overline{AB} + \overline{A}\overline{C}\overline{D} + \overline{A}\overline{B}D + \overline{AB}\overline{C}\overline{D}$$
 (May 2009)
75. Simplify the following Boolean function by Karnaugh map method:

$$F(A,B,C,D) = \sum m(1,5,9,12,13,15)$$
 (May 2009)
76. Find the octal equivalent of hexa decimal number AB.CD. (Dec 2010)
77. State and prove the consensus theorem. (Dec 2010)
78. Represent the decimal numbers - 200 and 200 using 2's complement binary form.
(April 2011)
79. Perform the following code conversions?

$$(1010.10)_{16} \longrightarrow (?)_2 \longrightarrow (?)_8 \longrightarrow (?)_{10}$$
 (Dec 2011)
80. State the different ways for representing the signed binary numbers. (Dec 2011)
81. Write the application of gray code. (May 2012)
82. The solution to the quadratic equation $x^2 - 11x + 22 = 0$ is $x = 3$ and $x = 6$. What is the base of the numbers? (May 2012)
83. Find the complement and dual of $F = x(y'z' + yz)$. (Dec 2012)
84. Convert $(101101.1101)_2$ to decimal and hexadecimal form. (May 2013)
85. What are the limitations of Karnaugh map? (May 2013)

PART B

1. (i) Specify the radix and the symbols used in (1) binary, (2) ternary, (3) quinary, (4) octal, (5) hexadecimal number system. (4)
- (ii) Convert $(329.678)_{10}$ to an equivalent number in base 6 having a conversion error less than 0.001. (4) (April 2003)
2. Use Quine McClusky method to obtain the minimal sum for the following function:

$$F(X_1X_2X_3X_4) = \sum (0, 1, 3, 6, 7, 14, 15) \quad (16) \text{ (April 2003)}$$
3. Simplify the function using Karnaugh map.
 - (i) $F(A, B, C, D) = \sum (0, 1, 2, 4, 5, 7, 11, 15)$
 - (ii) $F(W, X, Y, Z) = \sum (2, 3, 10, 11, 12, 13, 14, 15) \quad (8) \text{ (April 2003)}$
4. (i) Find a minimal sum of products representation for $f(A,B,C,D,E) = \sum m(1,4,6,10,20,22,24,26) + \sum d(0, 11, 16, 27)$ using Karnaugh map method. Draw the circuit of the minimal expression using only NAND gates. (11)
 - (ii) Prove that $(x_1 + x_2) \cdot (x_1 \cdot \bar{x}_3 + x_3) (\bar{x}_2 + x_1x_3)_3 = \bar{x}_1x_2 \quad (5) \text{ (April 2003)}$
5. Distinguish between Boolean addition and Binary addition. (5) (April 2003)
6. (i) Explain how you will construct an $(n + 1)$ bit Gray code from an n bit Gray code.
 (ii) Determine the MSP form of the switching function, (16) (April 2003)

$$F = \sum (0, 1, 4, 5, 6, 11, 14, 15, 16, 17, 20, 21, 22, 30, 32, 33, 36, 37, 48, 49, 52, 53, 59, 63).$$
7. (i) State and prove DeMorgan's theorem and expand the function $F = [(A+B)C + \bar{C}D]. \quad (5)$
 (ii) Simplify the following switching function using Karnaugh map,

$$F(A, B, C, D) = \sum (0, 5, 7, 8, 9, 10, 11, 14, 15) + \sum (1, 4, 13). \quad (11) \text{ (Nov. 2003)}$$
8. Simplify the five variable switching function:

$$f(A, B, C, D, E) = \sum m(3, 5, 6, 8, 9, 12, 13, 14, 19, 22, 24, 25, 30) \quad (16) \text{ (Nov. 2003)}$$
9. (i) State and prove De Morgan's law. (5)
 (ii) Explain the term prime implicants. (5) (Nov. 2003)
10. (i) Determine the decimal equivalent of the excess-3 number.

$$0110\ 1001\ 1100\ 0111 \quad (6)$$

 (ii) Explain how a 8 bit binary string can be converted to 9 bit number with odd parity using a single 74180. (10) (April 2004)

11. Using the K-map determine the MSP and MPS forces of the function,

$$\Sigma (0, 2, 6, 7, 8, 10, 12, 14, 15). \quad (16) \text{ (April 2004)}$$

12. Simplify using K-map to obtain a minimal POS expression:

$$(A + \bar{B} + C + D)(A + \bar{B} + C + D)(A + B + C + \bar{D}) \quad (8) \text{ (April 2004)}$$

13. Simplify using tabulation method: $F(w, x, y, z) = \Sigma (1, 4, 6, 7, 8, 9, 10, 11, 15)$ **(April 2004)**

14. Find the MSP form of $F(wxyz) = \Sigma (1-3, 5-10, 12 - 14)$ using the Quine-McCluskey technique. **(16) (April 2004)**

15. Simplify the following Boolean function by using the tabulation method.

$$F = \Sigma (0, 1, 2, 8, 10, 11, 14, 15) \quad (16) \text{ (April 2004)}$$

16. State and prove the postulates of Boolean expression. **(16) (April 2004)**

17. Perform the following arithmetic using $\bar{9}_s$ arithmetic. Compare them.

(i) $835 - 274$, (ii) $429 - 476$ using BCD and Excess 3 codes. **(16) (Nov. 2004)**

18. What are codes? Explain the different codes with examples. **(16) (Nov. 2004)**

19. Simplify the following function using K-map and tabular methods. Compare the methods.

$$F(A, B, C, D) = \Sigma (4, 5, 6, 7, 8); d(A, B, C, D) = \Sigma m(11, 12, 13, 14, 15)$$

Implement using NAND gates. **(16) (Nov. 2004)**

20. (i) Find the MSP form using a Karnaugh map.

$$F = \Sigma (0, 1, 6, 7, 9, 13-17, 32, 33, 38, 39, 46-49, 57, 61) \quad (12)$$

(ii) What are the advantages of tabular methods over the Karnaugh map? **(4) (Nov. 2004)**

21. (i) Express $x + yz$ as the product of maxterms. **(6)**

(ii) Minimize the switching function: $F(x_1 x_2 x_3 x_4) = \Sigma (1, 4, 5, 7, 13) + \Sigma f(0, 6, 14, 15)$ on a 4 variable Karnaugh map. **(10) (Nov. 2004)**

22. Convert $(1010111.101)_2$ to octal and hexadecimal. **(4) (April 2005)**

23. (i) Express the function $f(x, y, z) = XY + X\bar{Z}$ as a product of sum terms form. **(4)**

(ii) Express the following function as the minimal sum of products, using a K-map.

$$f(a, b, c, d) = \Sigma (0, 2, 4, 5, 6, 8, 10, 15) + \Sigma f(7, 13, 14). \quad (12) \text{ (April 2005)}$$

24. (i) Simplify the following using the Quine-McCluskey minimisation technique.

$$D = f(a, b, c, d) = \Sigma (0, 1, 2, 3, 6, 7, 8, 9, 14, 15) \}$$

Does Quine McClusky take care of don't care conditions? In the above problem, will you

consider any don't care conditions? Justify your answer (8)

(ii) List also the prime implicants and essential prime implicants for the above case. **(8) (April '05)**

25. $F_3 = f(a, b, c) = \sum (2, 4, 5, 6), F_2 = f(a, b, c) = \sum (2, 3, 6, 7), F_1 = f(a, b, c) = \sum (2, 5, 6, 7)$

Implement the above Boolean functions: (1) When each is treated separately and (2) When sharing common term. **(12) (April 2005)**

26. Minimize the following using Karnaugh map. Implement the resultant function using NOR gates only $f(A, B, C, D, E) = \sum m(2, 4, 7, 9, 26, 28, 29, 31)$ **(12) (April 2005)**

27. (i) Simplify the following function using tabulation procedure. Implement the reduced function using NAND gates only.

$$f = \sum m(0, 1, 3, 5, 6, 9, 11, 14, 21, 23, 24, 31) + \sum d(25, 30). \quad (12)$$

(ii) Define maxterms and minterms. Give examples. **(4) (April 2005)**

28. Simplify the following Boolean function using tabulation method

$$F(A, B, C, D) = \sum m(0, 2, 3, 6, 7, 8, 10, 12, 13) \quad (16) (\text{Dec. 2005})$$

29. (i) Perform each of the following computations using signed, 8 bit words in 1's complement and 2's complement binary arithmetic: **(10)**

$$(1) (+95)_{10} + (-63)_{10} \quad (2) (+42)_{10} + (-87)_{10}$$

$$(3) (-13)_{10} + (-59)_{10} \quad (4) (+38)_{10} + (-38)_{10} \quad (5) (-105)_{10} + (-120)_{10}.$$

(ii) Design a parity circuit that will assign a parity bit to the 8421 BCD code in an odd parity system. **(6) (Dec. 2005)**

30. (i) Simplify the following Boolean function in (1) Sum of products and (2)Product of sums.

$$F(A, B, C, D) = \sum (0, 1, 5, 8, 9, 10) \quad (10)$$

(ii) Plot the following Boolean function on a Karnaugh map and simplify it.

$$F(w, x, y, z) = \sum (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14) \quad (6) (\text{Dec. 2005})$$

31. Determine the prime implicants of the given function using Tabulation method.

$$F(w, x, y, z) = \sum (1, 4, 6, 7, 8, 9, 10, 11, 15). \quad (10) (\text{Dec. 2005})$$

32. What is biquinary code? Where is it used? Tabulate the code for decimal numbers 0 to 9.

(8) (Dec. 2005)

33.(i) Illustrate the following codes with an example: Binary, BCD, 2421, Gray and Excess 3. **(5)**

(ii) Consider decimal number 14 and write the equivalent for this in all the above. **(5)**

(iii) Write a note on ASCII code. **(6) (Dec. 2005)**

34. Reduce the following switching function using tabulation method:

$$\sum m(1, 3, 4, 7, 8, 10, 11, 13, 15)$$

(8) (Dec. 2005)

35. Reduce the following switching functions using Karnaugh map:

(i) $\sum m(0, 1, 2, 6, 7, 9, 12, 28, 29, 31)$

(8)

(ii) $P M(2, 3, 7, 9, 11, 12)$

(8) (Dec. 2005)

36. Simplify the following function using tabulation method:

$$f(a, b, c, d) = \sum m(1, 2, 3, 5, 9, 12, 14, 15) + \sum d(4, 8, 11)$$

(12) (Dec. 2005)

37. Find the reduced POS form of the following equation

$$f(a, b, c, d) = \sum m(1, 3, 7, 11, 15) + d(0, 2, 5).$$

Implement using NAND logic.

(8) (Dec. 2005)

38. (i) Prove the De Morgan's laws using Boolean algebra.

(8)

(ii) Find the minimal sum of product form for the following switching function:

$$f(x_1, x_2, x_3, x_4, x_5) = \sum m(2, 3, 6, 7, 11, 12, 13, 14, 15, 23, 28, 29, 30, 31) \quad (8) (\text{May 2006})$$

39. (i) Simplify the following Boolean expression: $(x_1 + x_2)(x_1 x_3' + x_3)(x_2' + x_1 x_3)' \quad (6)$

(ii) Find the minimal sum-of-product expression for the following switching function:

$$f(x_1, x_2, x_3, x_4, x_5) = \sum m(1, 2, 3, 6, 8, 9, 14, 17, 24, 25, 26, 27, 30, 31) + \sum d(4, 5)$$

(10) (May 2006)

40. (i) Prove by perfect induction:

(1) $A + AB = A$

(2) $A(A + B) = A$

(3) $A + \overline{A}B = A + B$ and

(4) $A(\overline{A} + B) = AB.$

(12)

(ii) Reduce the following function using K map: $f = AB\overline{C} + \overline{A}\overline{B}C + ABC + A\overline{B}C$ and realize using NAND gates only. $(8) (\text{May 2006})$

41. (i) Simplify the Boolean function $F = \sum m(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

using Karnaugh map and obtain the minimum SOP form(s).

(8)

(ii) Illustrate the rules for binary addition and subtraction using 2's complement arithmetic. Give examples. $(8) (\text{May 2006})$

42. What is advantage of using Tabulation method? Determine the prime implications of the following function using Tabulation method.

$$F(w, x, y, z) = \sum(1, 4, 6, 7, 8, 9, 10, 11, 15)$$

(16) (Dec. 2006)

- 43.** (i) Explain about common postulates used to formulate various algebraic structures.
(12) (Dec. 2006)

(ii) Given the following Boolean function.

$$F = A'C + A'B + AB'C + BC$$

Explain it in sum of minterms and find the minimal SOP expression

(4) (Dec. 2006)

- 44.** Find the minimum sum of products expression using K-map for the function $F = \sum m(7, 9, 10, 11, 12, 13, 14, 15)$ and realize the minimized function using only NAND gates.**(16) (May '07)**

- 45.** Simplify using Quine-McClusky method $F = \sum m(0, 1, 2, 3, 10, 11, 12, 13, 14, 15)$. **(16) (May 2007)**

- 46.** (i) Convert 0.95 decimal number to its binary equivalent. **(6)**

(ii) Perform $(756)_8 - (637)_8 + (725)_{16}$. Express the answer in octal form. **(10) (May 2007)**

- 47.** (i) Perform the addition in excess-3 code $16 + 29$. **(6)**

(ii) Write short notes on error detection and error correction codes. **(10) (May 2007)**

- 48.** Find the minimum sum of product function using Quine-McClusky method

$$f(a,b,c,d) = \sum (0,3,4,6,11) + \sum d (0,8,10,12,13) \quad \text{**(16) (May 2007)**}$$

- 49.** Using Tabulation method simplify the Boolean function

$$F(w,x,y,z) = \sum (1,2,3,5,9,12,14,15) \text{ which has the don't care conditions } d(4,8,11).$$

(16) (May 2007)

- 50.** Reduce the Boolean function using K-map technique and implement using gates $F(w,x,y,z) = \sum (0,1,4,8,9,10)$ which has the don't care conditions $d(w,z,y,z) = \sum (2,11)$. **(16) (May 2007)**

- 51.** Using Tabulations method simplify the Boolean function.

$$F(w,x,y,z) = \sum (2,3,4,6,7,11,12,13,14) \text{ which has the don't care conditions } d(1,5,15).$$

(16) (Dec. 2007)

- 52.** Simplify the Boolean function using Variable Entered Mapping method and implement using gates.

$$F(w,x,y,z) = \sum (0,2,4,6,8,10,12,14). \quad \text{**(16) (Dec. 2007)**}$$

- 53.** Find the prime implicants for the following function and determine which are essential.

$$F(w,x,y,z) = \sum (0,2,4,5,6,7,8,10,13,15) \quad \text{**(10) (Dec. 2008)**}$$

- 54.** Simplify the following Boolean function F together with don't care condition using Karnaugh map method.

$$(i) F(A,B,C,D) = \sum m(0,6,8,13,14), d(A,B,C,D) = \sum m(2,4,10) \quad \text{**(6)**}$$

$$(ii) F(A,B,C,D) = \sum m(0,2,4,5,8,14,15), d(A,B,C,D) = \sum m(7,10,13) \quad (5)$$

$$(iii) F(A,B,C,D) = \sum m(4,6,7,8,12,15), d(A,B,C,D) = \sum m(2,3,5,10,11,14) \quad (5)$$

(May 2009)

- 55.** Simplify the following Boolean expressions to a minimum number of literals.

$$(i) ABC + A\bar{B}C + \bar{A}\bar{B} \quad (3)$$

$$(ii) \bar{A}BC + AC + \bar{B} \quad (3)$$

$$(iii) (\bar{A} + \bar{B})(\bar{A} + \bar{B}) \quad (3)$$

$$(iv) BC(AD + A\bar{D}) + A\bar{B} \quad (3)$$

$$(v) (A + \bar{B} + A\bar{B})(AB + \bar{A}C + BC) \quad (4) \text{ (May 2009)}$$

- 56.** Simplify the following 5 variable Boolean expression using McCluskey method.

$$F = \sum m(0,1,9,15,24,29,30) + d(8,11,31). \quad (16) \text{ (Dec. 2010)}$$

- 57.** Determine the minterm sum of product form of the switching function.

$$F = \sum (0,1,4,5,6,11,14,15,16,17,20,22,30,32,33,36,37,48,49,52,53,59,63) \quad (16) \text{ (Dec. 2010)}$$

- 58. (i)** Convert $(1947)_{10}$ into its equivalent octal and hexadecimal representations. **(10)**

- (ii)** Perform $(147-89)$ using 2's complement binary arithmetic. **(6) (May 2011)**

- 59. (i)** Minimize the following expression using Karnaugh map.

$$Y = A'BC'D' + A'BC'D + ABC'D' + AB'C'D + A'B'CD'. \quad (10)$$

- (ii)** State and prove the De Morgan's theorems. **(6) (May 2011)**

- 60.** Simplify the following Boolean function F using Karnaugh map method.

$$(i) F(A,B,C,D) = \sum (1,4,5,6,12,14,15) \quad (4)$$

$$(ii) F(A,B,C,D) = \sum (0,1,2,4,5,7,11,15) \quad (4)$$

$$(iii) F(A,B,C,D) = \sum (2,3,10,11,12,13,14,15) \quad (4)$$

$$(iv) F(A,B,C,D) = \sum (0,2,4,5,6,7,8,10,13,15) \quad (4) \text{ (Dec. 2011)}$$

- 61.** Simplify the following Boolean expression to a minimum number of literals:

$$(i) \bar{A}\bar{C} + ABC + A\bar{C} \quad (2)$$

- (ii) $XYZ + \bar{X}Y + XY\bar{Z}$ (2)
- (iii) $XY + YZ + XY\bar{Z}$ (2)
- (iv) $A\bar{B} + ABD + A\bar{B}\bar{D} + \bar{A}CD + \bar{A}\bar{B}\bar{C}$ (5)
- (v) $BD + B\bar{C}D + \bar{A}BCD$ (5) (Dec. 2011)
- 62.** Minimize the expression using Quine McCluskey (Tabulation) method
 $Y' = A'B'C'D' + A'BC'D + ABC'D' + ABC'D + AB'C'D + A'B'CD'$. (16) (May 2012)
- 63.** (i) Simplify $F(A, B, C, D) = \sum (0, 1, 2, 5, 8, 9, 10)$ in sum of products and product of sums using K-map. (12)
- (ii) Write notes on negative and positive logic. (4) (Dec. 2012)
- 64.** (i) Simplify the expression $F(A, B, C, D) = \sum (1, 4, 6, 7, 8, 9, 10, 11, 15)$ using Quine-McClusky method. (12)
- (ii) Check if NOR operator is associative. (4) (Dec. 2012)
- 65.** (i) Reduce the following function using map technique:
- (a) $f(A, B, C) = \sum m(0, 1, 3, 7) + \sum d(2, 5)$
- (b) $F(w, x, y, z) = \sum (0, 7, 8, 9, 10, 12) + \sum d(2, 5, 13)$ (16) (May 2013)
- 66.** Simplify the Boolean function using Quine McCluskey method:
 $F(A, B, C, D, E, F) = \sum m(0, 5, 7, 8, 9, 12, 13, 23, 24, 25, 28, 29, 37, 40, 42, 44, 46, 55, 56, 57, 60, 61)$ (16) (May 2013)

LOGIC GATES

PART A

1. Obtain the following operations using only NAND gates (a) NOT (b) AND. **(April 2003)**
2. If a manufacturer specifies the minimum logical 1 at a gate output a 4.0 V and also specifies that any voltage down upto 3.6 V will be considered as logical 1, find the noise margin. **(April 03)**
3. How will you use a 4 input NOR gate as a 2 input NOR gate? **(April 2003)**
4. Show that the NAND connective is not associative. **(April 2003)**
5. State and Prove DeMorgan's theorem. **(April 2003)**
6. Show that a positive logic NAND gate is same as a negative logic NOR gate. **(April 2003)**
7. Implement EXOR gate using only NAND gate. **(April 2003)**
8. Define noise margin. **(Nov. 2003)**
9. Show that a bubbled AND gate works like a NOR gate. **(April 2004)**
10. How can a NAND gate be used as an inverter? **(April 2004)**
11. Determine the fanout given $I_{IH(max)} = 40 \mu\text{A}$ and $I_{OH(max)} = 400 \mu\text{A}$. **(April 2004)**
12. Show that the NOR connective is not associative. **(April 2004)**
13. Show that how NAND gates can be used to implement the basic Boolean functions. **(April 04)**
14. How many inputs are needed for the expression, $W = \bar{A}\bar{B}D + A\bar{C}\bar{D} + EF$ **(Nov. 2004)**
15. Define noise margin. **(Nov. 2004)**
16. What are universal gates? **(Nov. 2004)**
17. Obtain 3 level NOR-NOR implementation of $f(a, b, c) = [ab + cd]ef$. **(Nov. 2004)**
18. Define fan-in. **(Nov. 2004)**
19. What is tri-state logic? **(Nov. 2004)**
20. Define $V_{IH}(\text{min})$ and $V_{IL}(\text{max})$ of an IC. **(Nov. 2004)**
21. Show that a positive logic NAND gate is the same as a negative logic NOR gate. **(Nov. 2004)**
22. Define noise margin and noise immunity. **(April 2005)**
23. Realize the function $f(A, B) = \bar{A}\bar{B} + A\bar{B}$ by using only NAND gates. **(April 2005)**
24. Define power dissipation and propagation delay. **(April 2005)**
25. What is meant by multilevel gates networks? **(April 2005)**
26. What is noise margin? **(Dec. 2005)**
27. What is fan-out of a gate? **(Dec. 2005)**
28. What is a tristate gate? **(Dec. 2005)**
29. Realize $f = A'B + AB'$ using minimum universal gates. **(Dec. 2005)**

30. Draw a tristate inverter and draw its truth table. **(Dec. 2005)**
31. Define fan-in. **(Dec. 2005)**
32. Draw the internal circuit of a NOR gate latch and derive the truth table. **(May 2006)**
33. Construct a combinational circuit to convert given binary coded decimal number into an Excess-3 code. For example when the input to the gate is 0110 then the circuit should **(May 2007)**
34. Minimize the function using Boolean algebra $f=x(y+w'z) + wxz$. **(May 2007)**
35. Define propagation delay. **(May 2007)**
36. Write the truth tables of logical AND and XOR gates. **(April 2011)**
37. Realize OR gate using only NAND gates. **(Dec 2012)**
38. Write an HDL behavioral description of a 4-bit comparator with a 6-bit output $y[5;0]$. Bit 5 of y is for equal, bit 4 for unequal, bit 3 for greater than, bit 2 for less than, bit 1 for greater than or equal, and bit 0 for less than or equal to. **(May 2012)**

PART-B

1. Implement the following function with either NAND or NOR gates. Use only 4 gates. Only the normal inputs are available.(i) $d = WYZ$, (ii) $F = \overline{W}XZ + \overline{W}YZ + \overline{X} + WX\bar{Y}Z$ **(8) (April 2003)**
2. Show that if all the gates in a two level OR-AND gate network are replaced by NOR gates, the output function does not change. **(8) (April 2003)**
3. Implement the switching function whose octal designation is 274 using NOR gates only. **(16) (April 2003)**
4. (i) Compare the performance of any five logic families, based on any five suitable parameters. **(8)**
(ii) Define the terms fan out, tristate gates, fan in. **(8) (Nov. 2003)**
5. Obtain a 4-level NAND network for $f(A, B, C, D) = (\overline{A}B + C)D + EF$. **(8) (April 2004)**
6. Show that the NAND operation is not associative. **(6) (Nov. 2004)**
7. (i) Explain how an EX-OR gate can be built by using four NAND gates. **(4)**
(ii) The output of a NAND gate network is $F(A, B, C) = \Rightarrow (3, 6, 7, x)$. The output of the gate network does not change if all the gates are replaced by NOR gates. Determine the value of x . **(12) (Nov. 2004)**
8. Design a logic circuit to simulate the function $f(A, B, C) = A(B + C)$ by using only NAND gates. **(4) (April 2005)**
9. Realize the functions of NOT, AND, OR gates only with NOR gates. **(4) (April 2005)**
10. Convert a NOR with an equivalent AND gate. **(4) (April 2005)**
11. Prove that NOR gate is a universal gate. Also prove the same for NAND gate. **(16) (Dec. 2005)**
12. Implement the following function with NAND gates

$$F(x, y, z) = \Rightarrow (0, 6)$$

(6) (Dec. 2005)

13. Implement the following function using a quad 2–input NOR gates:

$$f = (A' B + C) \cdot D'$$

(8) (Dec. 2005)

14. Design a network with four inputs and three outputs which realizes the following functions:

$$F_1(a, b, c, d) = \sum m(11, 12, 13, 14, 15)$$

$$F_2(a, b, c, d) = \sum m(3, 7, 11, 12, 13, 15)$$

$$F_3(a, b, c, d) = \sum m(3, 7, 12, 13, 14, 15)$$

(8) (Dec. 2005)

15. Draw the symbol, truth table and the equation of the three basic gates and two universal gates and realize all the five gates using either of the universal gates. (16) (May 2006)

16. Design a 4 bit magnitude comparator to compare two 4 bit numbers. (16) (Dec. 2006)

17. Draw a NAND logic diagram that implements the complement of the function.

$$F(A, B, C, D) = \sum(0, 1, 2, 3, 4, 8, 9, 12)$$

(6) (Dec. 2008)

18. Given the Boolean function $F = xy + x'y' + y'z$.

(i) Implement it with AND, OR and inverter gates (4) (Dec. 2008)

(ii) Implement it with OR and inverter gates (6) (Dec. 2008)

(iii) Implement it with AND and inverter gates (6) (Dec. 2008)

19. (i) Define Prime Implicant and Essential Prime Implicant. (4)

- (ii) Write the procedure for obtaining the logic diagram with NAND gates from a Boolean function. (4)

- (iii) Implement the switching function.

$$F(x, y, z) = \sum m(1, 2, 3, 4, 5, 7)$$

(8) (May 2012)

UNIT – II

COMBINATIONAL LOGIC

- **Combinational Circuits**
- **Analysis and Design Procedures**
- **Circuits for Arithmetic Operations**
- **Code Conversion**
- **Decoders and Encoders**
- **Multiplexers and Demultiplexers**
- **Introduction to HDL**
- **HDL Models of Combinational circuits**

COMBINATIONAL LOGIC

2.1 INTRODUCTION

A combinational circuit consists of logic gates whose outputs at any time are determined directly from the present combination of input without regard to previous inputs. A combination circuit consists of input variables, logic gates and output variables. The logic gates accept signals from the inputs and generate signals to the outputs. This process transforms binary information from the given input data to the required output data. A block diagram of a combinational circuit is shown in **Figure 2.1**. It accepts ‘n’ binary input variables and generate ‘m’ output variables depending on the logical combination of gates.

The possible representations for a combinational logic function:

- ◆ A truth table
- ◆ An algebraic sum of minterms, the canonical sum
- ◆ A minterm list using the Σ notation
- ◆ An algebraic product of maxterms, the canonical product
- ◆ A maxterm list using the π notation.

2.2 DESIGN PROCEDURE

Any combinational circuit can be designed by following the design procedure given below:

- ❖ From the given word description of the problem, identify the number of input variables and required output.
- ❖ The variables input and output variables are assigned letter symbols.
- ❖ Draw a truth table such that it completely describes the operation of the circuit for different combinations of inputs.
- ❖ Obtain the Boolean expression for each output using either algebraic or K-map method.
- ❖ Obtain the logic diagram.

In practical design method, some constraints are considered:

- ❖ Minimum number of gates.

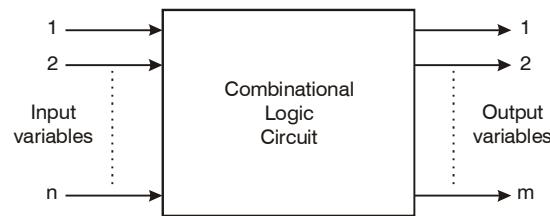


Fig. 2.1 : Block diagram of a combinational circuit

- ❖ Minimum number of inputs to a gate.
- ❖ Minimum propagation time of the signal through the circuit.
- ❖ Minimum number of interconnections.
- ❖ Limitations of the driving capabilities of each gate.

Example 2.1: Design a combination logic circuit with three input variables that will produce a logic 1 output when more than one input variables are logic 1.

Solution: Number of Input variables = 3

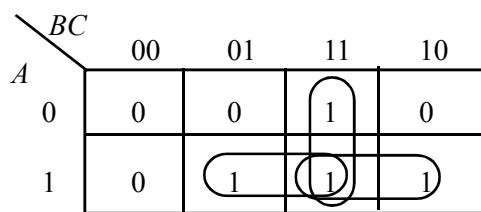
Number of Output variables = 1

Let assign the letter symbols A , B and C to three input variables and assign ' Y ' to one output variable. The relationship between input variables and output variable is tabulated in truth table as given in **Table 2.1**.

TABLE 2.1: Truth Table

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

Using K map method, find the Boolean expression for Y .



$$Y = AB + AC + BC$$

Draw the logic diagram for $Y = AB + AC + BC$.

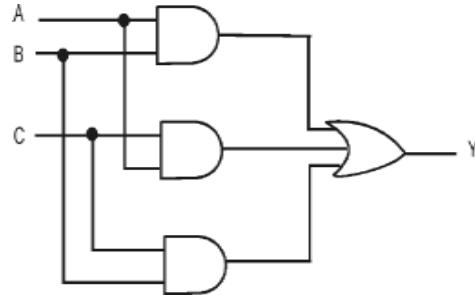


Fig. 2.2: Logic diagram

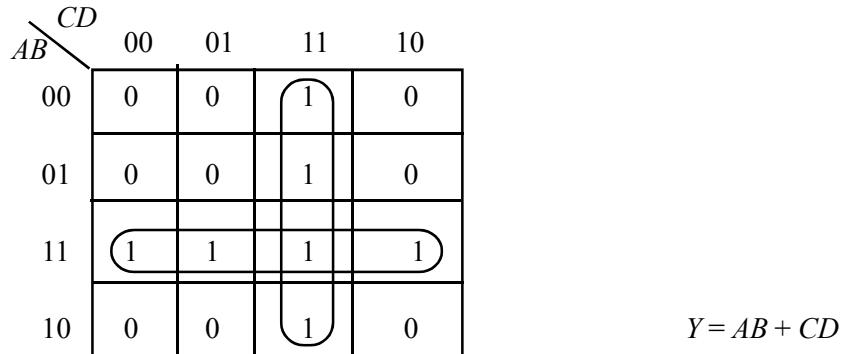
Example 2.2: Design a combinational logic circuit that has four inputs and one output. The output is high if both inputs A and B are high or both inputs C and D are high.

Solution: The relationship between input variables (A, B, C, D) and output variable (Y) is tabulated as shown in **Table 2.2**.

TABLE 2.2 : Truth Table

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

Using K map obtain the Boolean expression for output variable Y



Draw the logic diagram for $Y = AB + CD$.

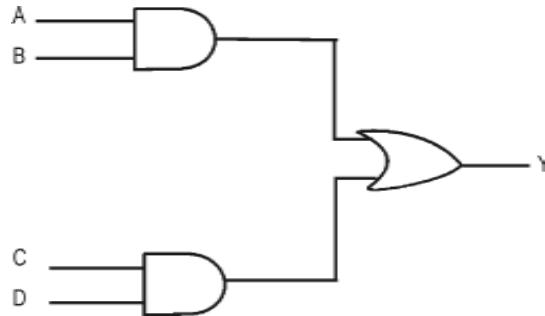
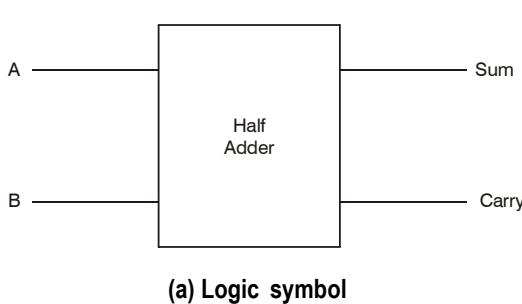


Fig. 2.3 : Logic diagram

2.3 HALF ADDER

“The half adder accepts two binary digits on its inputs and produces two binary digits on its outputs, a sum and a carry bit.”

Figure 2.4(a) shows the logic symbol of a half-adder and **Figure 2.4(b)** shows the truth table for the half-adder.



(a) Logic symbol

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

(b) Truth Table

Fig. 2.4: Half adder

The half adder follows the basic rules for addition:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10$$

By using K-map, determine the expression for output variables (sum and carry out)

	B	0	1
A	0	0	(1)
	1	(1)	0

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

	B	0	1
A	0	0	0
	1	0	(1)

$$\text{Carry} = C_{out} = AB$$

The output carry is produced with an AND gate with A and B on the inputs and the output sum is produced with an EX-OR gate as shown in **Figure 2.5**.

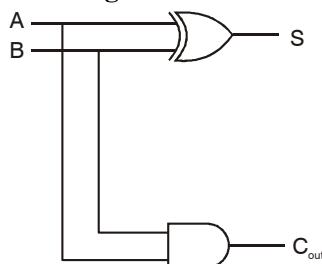


Fig. 2.5: Half adder logic diagram

2.4 FULL ADDER

“The full adder accepts three inputs—two input bits and input carry and generates sum output and output carry.”

A half adder has only two inputs and there is no provision to add a carry coming from the lower order bits when multi addition is performed. For this purpose a full adder is used. A full adder has 3 inputs – A, B, C_{in} and two outputs – Sum (s) and carry out (C_{out}). **Figure 2.6(a)** shows the logic symbol and **Figure 2.6(b)** shows the truth table for full adder.

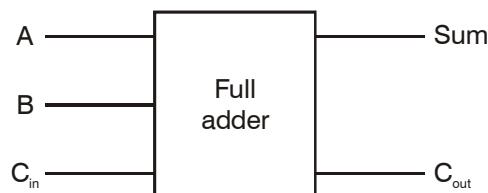


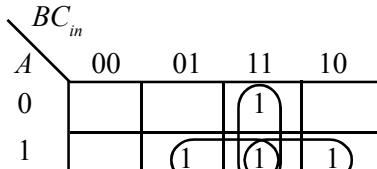
Fig. 2.6(a): Logic symbol

INPUTS			OUTPUTS	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Fig. 2.6(b): Truth Table

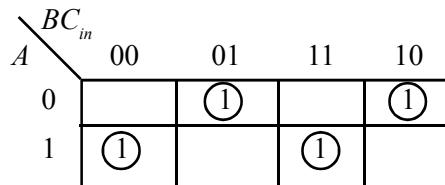
By using K-map simplification, determine the expression for output variables S and C_{out} .

(i) Expression for Carry



$$C_{out} = AC_{in} + AB + BC_{in}$$

(ii) Expression for Sum



$$\begin{aligned} S &= \overline{A} \overline{B} C_{in} + \overline{A} B \overline{C}_{in} + A \overline{B} \overline{C}_{in} + \overline{A} \overline{B} \overline{C}_{in} + ABC_{in} \\ &= \overline{A} (\overline{B} C_{in} + B \overline{C}_{in}) + A (\overline{B} \overline{C}_{in} + BC_{in}) \\ &= \overline{A} (B \oplus C_{in}) + A (\overline{B} \oplus C_{in}) \end{aligned}$$

$$\begin{aligned} \text{Let } X &= B \oplus C_{in}, \text{ then } S = \overline{A}X + A\overline{X} \\ &= A \oplus X \end{aligned}$$

Replacing X with $B \oplus C_{in}$, then

$S = A \oplus B \oplus C_{in}$ $C_{out} = AC_{in} + AB + BC_{in}$
--

The logic diagram is constructed by logic gates as shown in **Figure 2.7**.

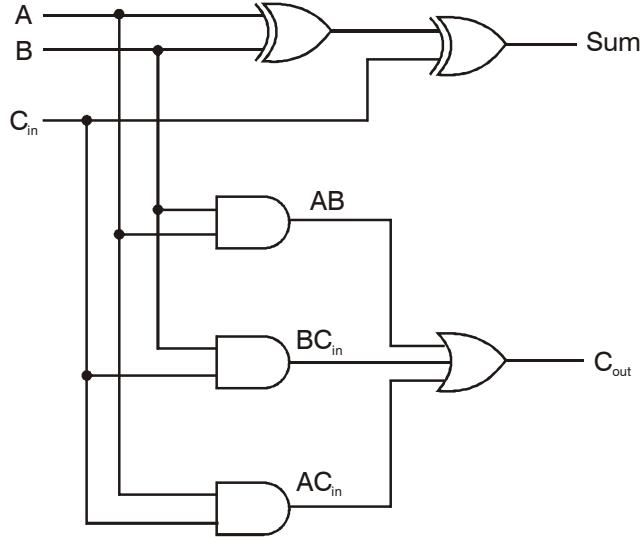


Fig. 2.7 : Full adder logic diagram

2.4.1 Implementation of full adder using Half adders

The full adder can also be implemented with two half-adders and one OR gate as shown in **Figure 2.8**.

For a half adder, $\text{Sum} = A \oplus B$

$$C_{out} = AB$$

For a full adder, $\text{Sum} = (A \oplus B) \oplus C_{in}$

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

Using these expressions draw the logic diagram for full adder.

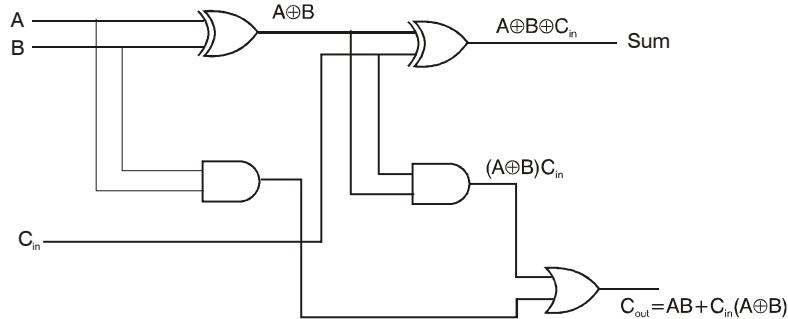


Fig. 2.8: Implementation of full adder

Notice in **Figure 2.8**, there are two half adders, connected as shown in the block diagram **Figure 2.9**, with their output carries ORed.

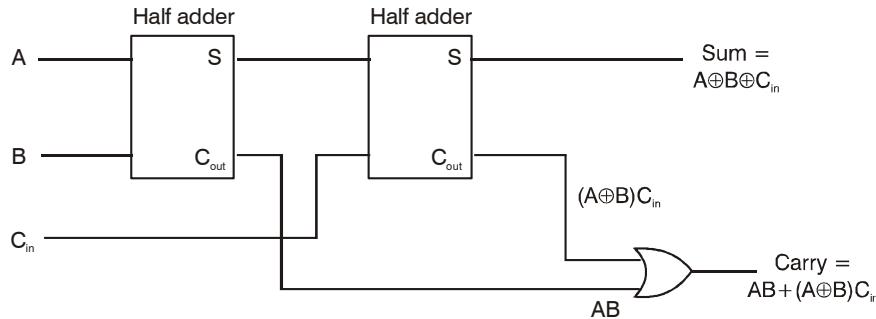
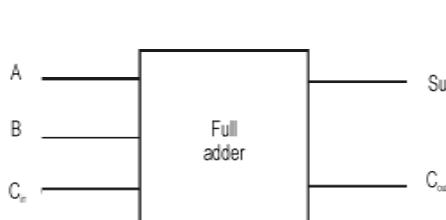


Fig. 2.9 : Implementation of full adder with half adders

2.5 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 $A - B = D$ and the borrow is B_{out} . **Figure 2.10(a)** shows the logic symbol of a half subtractor and **Figure 2.10(b)** shows the truth table for the half-subtractor.



(a) Logic symbol

Inputs		Outputs	
A	B	D	B_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

(b) Truth table

Fig. 2.10 : Half subtractor

The half subtractor follows the basic rules for subtraction:

$$0 - 0 = 0$$

$0 - 1 = 1$ with 1 borrow

$$1 - 0 = 1$$

$$1 - 1 = 0$$

By using K-map, determine the expression for output variables, Difference (D) and Borrow out (B_{out}).

	B	0	1
0	A	0	(1)
1	(1)	0	

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

$$= A \oplus B$$

	B	0	1
0	A	0	(1)
1	(1)	0	

$$B_{out} = \bar{A}B$$

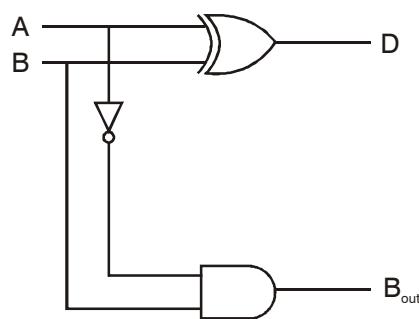
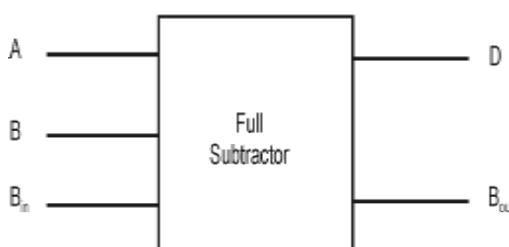


Fig. 2.11: Half subtractor

2.6 FULL SUBTRACTOR

“The full subtractor accepts three inputs – minuend, subtrahend and borrow from the previous stage and generates difference output and borrow output.”

The full subtractor has 3 inputs-Minuend (A), Subtrahend (B) and borrow from the previous stage (B_{in}) and two outputs-difference (D) and borrow out (B_{out}). **Figure 2.12(a)** shows the logic symbol and **Figure 2.12(b)** shows the truth table for full subtractor.



(a) Logic symbol

Inputs			Outputs	
A	B	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

(b) Truth Table

Fig. 2.12 : Full Subtractor

K-map Simplification

(i) Expression for 'D':

	BB_{in}	00	01	11	10
A	0	0	1	0	1
	1	1	0	1	0

$$D = \overline{A} \overline{B} B_{in} + \overline{A} B \overline{B}_{in} + A \overline{B} \overline{B}_{in} + ABB_{in}$$

(ii) Expression for Borrow (B_{out})

	BB_{in}	00	01	11	10
A	0	0	1	1	1
	1	1	0	1	0

$$B_{out} = \overline{A}B + \overline{A}B_{in} + BB_{in}$$

The Boolean function for D is simplified as,

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

The logic diagram of full subtractor is constructed by logic gates is shown in **Figure 2.13**.

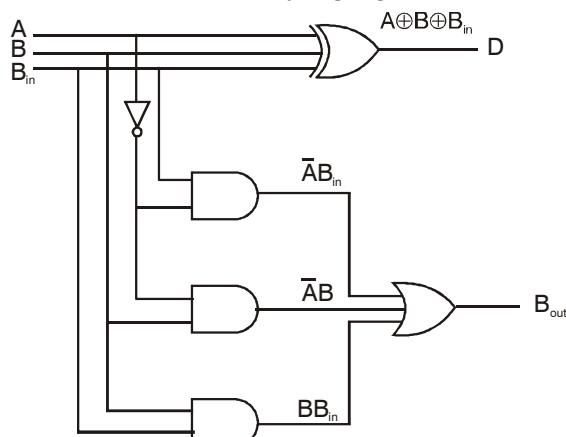


Fig. 2.13 : Full subtractor-logic diagram

2.6.1 Implementation of Full Subtractor using Half Subtractor

A full subtractor can also be implemented with two half subtractors and one OR gate as shown in **Figure 2.14**.

For a half subtractor, $D = A \oplus B$

$$B_{out} = \overline{A}B$$

For a full subtractor, $D = A \oplus B \oplus B_{in}$

$$\begin{aligned} B_{out} &= \overline{AB} + \overline{AB}_{in} + BB_{in} \\ &= \overline{AB} + \overline{AB}_{in}(B + \overline{B}) + BB_{in} \\ &= \overline{AB} + \overline{AB}B_{in} + \overline{A}\overline{B}B_{in} + BB_{in} \\ &= \overline{AB}(1 + B_{in}) + \overline{A}\overline{B}B_{in} + BB_{in} \\ &= \overline{AB} + BB_{in} + \overline{A}\overline{B}B_{in} \\ &= \overline{AB} + BB_{in}(A + \overline{A}) + \overline{A}\overline{B}B_{in} \\ &= \overline{AB} + ABB_{in} + \overline{AB}B_{in} + \overline{A}\overline{B}B_{in} \\ &= \overline{AB}(1 + B_{in}) + ABB_{in} + \overline{A}\overline{B}B_{in} \\ &= \overline{AB} + ABB_{in} + \overline{A}\overline{B}B_{in} \\ &= \overline{AB} + B_{in}(AB + \overline{A}\overline{B}) \end{aligned}$$

$$B_{out} = \overline{A}B + B_{in}(\overline{A} \oplus \overline{B})$$

Using these expressions, draw the logic diagram for full subtractor as given below:

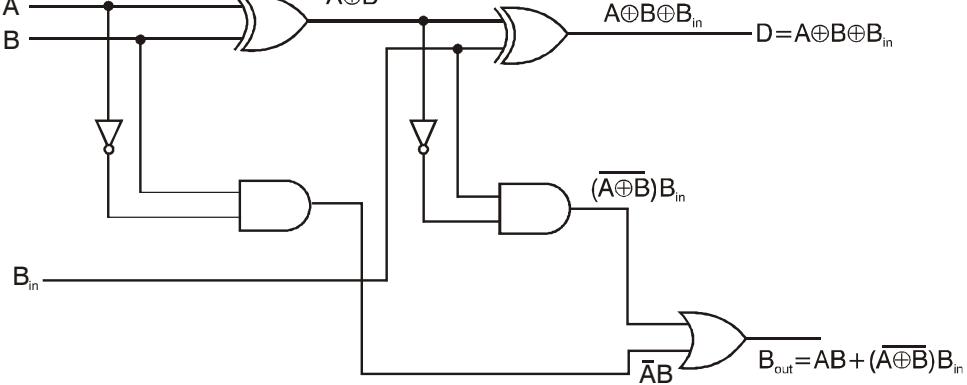


Fig. 2.14 : Implementation of full subtractor

Notice in **Figure 2.14**, there are two half subtractors connected as shown in the block diagram **Figure 2.15**, with their output borrows ORed.

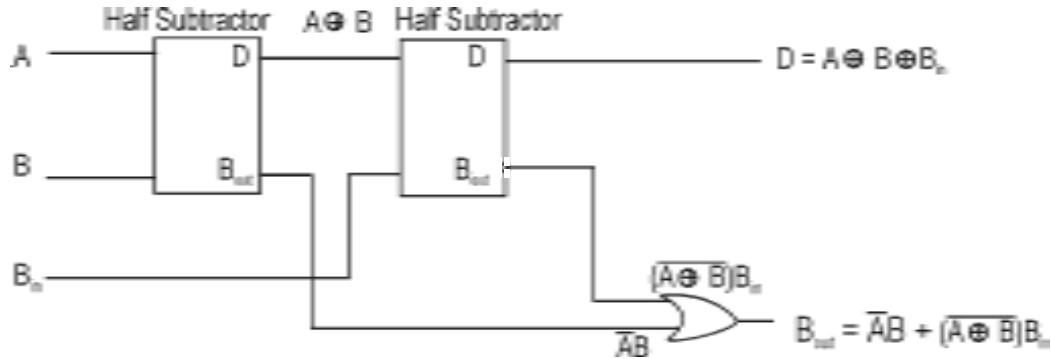


Fig. 2.15 : Implementation of full subtractor using half subtractors

2.7 PARALLEL BINARY ADDERS

A single full adder is capable of adding two 1 bit numbers and an input carry. To add binary numbers with more than one bit, additional full-adders are required. In order to add two binary numbers, a full adder is required for each bit in the number. Thus for two-bit numbers, we need two full-adders. Similarly for the addition of four-bit numbers, we need four full-adders and so on.

A binary parallel adder is a digital circuit that produces the arithmetic sum of two binary numbers in parallel. It consists of full-adders connected in a chain, with the output carry from each full-adder connected to the input carry of the next full-adder in the chain. The least significant bits (LSB) of the two binary numbers being added go into the right most full adder whereas the higher order bits are applied as shown to the successive higher-order adders. The most significant bits (MSB) of the two binary numbers are applied to the left-most full-adder. The block diagram of 4 bit parallel adder is shown in **Figure 2.16**.

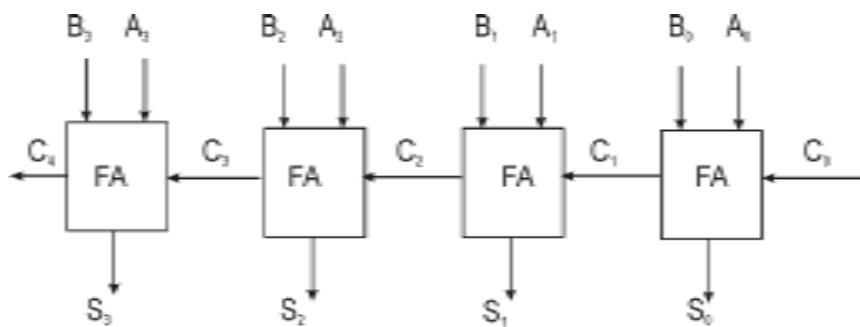


Fig. 2.16 : 4 bit Parallel Adder

The input carry is labelled as C_0 and the output carry is labelled as C_4 .

There are several parallel adders that are available as ICs. **Table 2.3** shows the most commonly used 4 bit parallel adders. In the IC package, a 4 bit parallel adder consisting of 4 terminals for the augend bits, 4 terminals for addend bits, 4 terminals for the sum bits and 2 terminals for the input and output carries. The logic symbol of a 4 bit parallel adder is shown in **Figure 2.17**.

TABLE 2.3: IC 4 Bit Parallel Adder

Family	IC
TTL	7483A
LSTTL	4LS83A
CMOS	74HC283

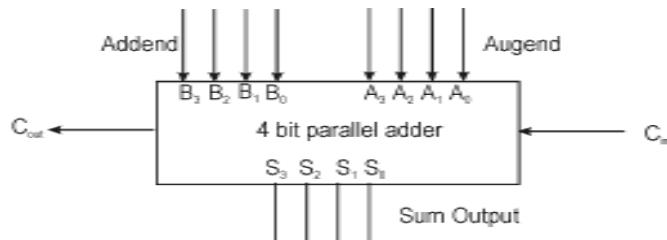


Fig. 2.17 : Logic symbol of a 4-bit parallel adder

8 bit Parallel adder: In order to accomplish addition of large binary numbers, two or more IC adders can be connected together, i.e., cascaded. **Figure 2.18.** Shows two 74LS83A adders connected to add two 8 bit numbers, $A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$ and $B_7 B_6 B_5 B_4 B_3 B_2 B_1 B_0$.

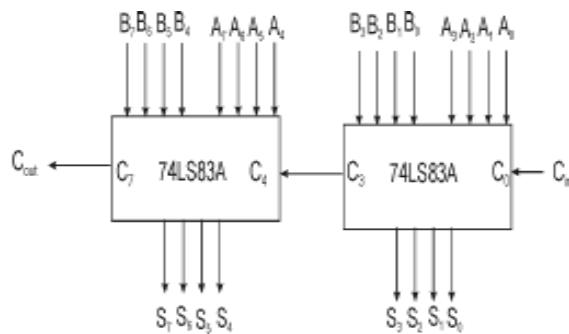


Fig. 2.18 : Cascading of two 74L383A

2.8 PARALLEL SUBTRACTOR

The subtraction of binary numbers can be done here by means of 2's complement method. The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits. In the parallel subtractor, the 1's complement can be implemented with inverters added with each data input B and a one can be added to the sum through the input carry as shown in **Figure 2.19**.

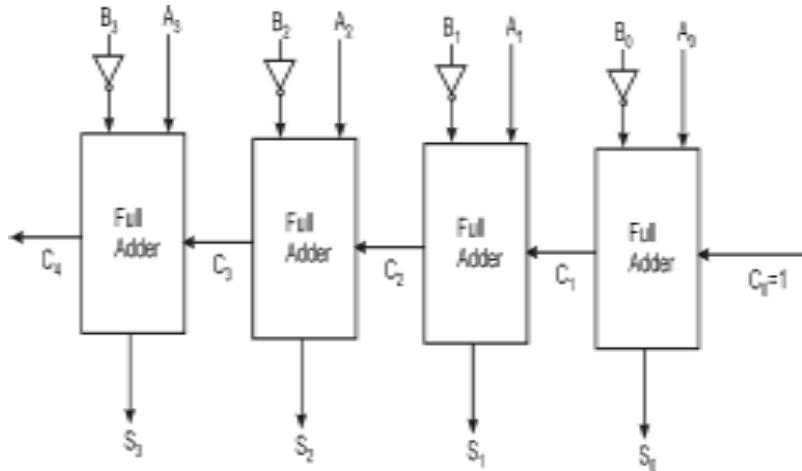


Fig. 2.19 : 4 Bit Parallel Subtractor

2.9 BINARY ADDER/SUBTRACTOR

The 4 bit binary adder/subtractor circuit is shown in **Figure 2.20**. It performs the operations of both addition and subtraction. It has two 4 bit input $A_0 A_1 A_2 A_3$ and $B_0 B_1 B_2 B_3$. The $\overline{\text{ADD}}/\text{SUB}$ line is the control line, connected with input carry C_0 of the least significant bit of the full adder, is used to perform the operations of addition and subtraction.

The addition and subtraction operations can be combined into one circuit with one common binary adder. This is done by including an EX-OR gate with each full-adder. When $\overline{\text{ADD}}/\text{SUB} = 0$, the circuit is an adder and when $\overline{\text{ADD}}/\text{SUB} = 1$, the circuit is a subtractor. Each EX-OR gate receives input $\overline{\text{ADD}}/\text{SUB}$ and one of the inputs of B.

ADDER

When $\overline{\text{ADD}}/\text{SUB} = 0$, the operation is $B \oplus 0 = 0$. The full-adder receives the value of B and the input carry $C_0 = 0$. Thus the circuit performs the addition operation, $A + B$.

SUBTRACTOR

When $\overline{\text{ADD}}/\text{SUB} = 1$, the operation is $B \oplus 1 = \overline{B}$. The full adder receives the value of \overline{B} and the input carry $C_0 = 1$. The B inputs are all complemented and a 1 is added through the input carry (C_0). Thus the circuit performs the subtraction operation, i.e.,

$$A + (\text{2's complement of } B) = A - B.$$

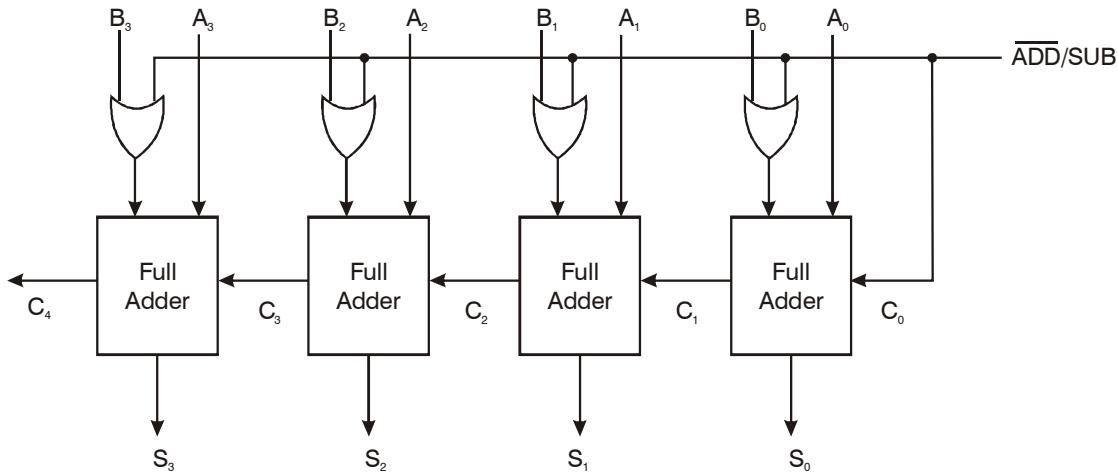


Fig. 2.20 : 4 Bit Adder/Subtractor

2.10 SERIAL ADDER

In the serial adder, the addition operation is done by bit-by-bit. The serial adder requires simpler circuit than a parallel adder. The speed of operation of serial adder is lower than the parallel adder. The operation of serial adder as follows:

Two shift registers A and B are used to store the numbers to be added serially. A single full adder is used to add one pair of bits at a time along with the carry. The D-flipflop is used to store the carry output of the full adder, so that it can be added to the next significant position of the numbers in the registers. The contents of the shift registers shift from left to right and their outputs starting from A_0 and B_0 are fed into a single full adder along with the output of the D-flipflop upon application of each clock pulse. The sum output of the full adder is fed to MSB bit (S_3) of the sum register. For each succeeding clock pulse, the contents of the both shift registers are shifted once to the right and new carry bit are transferred to sum register and D- flipflop respectively. This process continues until all the pairs of bits are added. The diagram of a serial adder is shown in **Figure 2.21** and a four bit serial adder is shown in **Figure 2.22**.

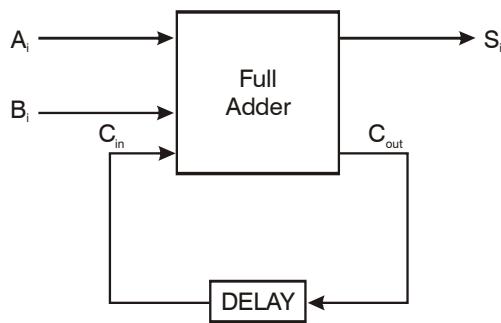


Fig. 2.21 : Serial Adder

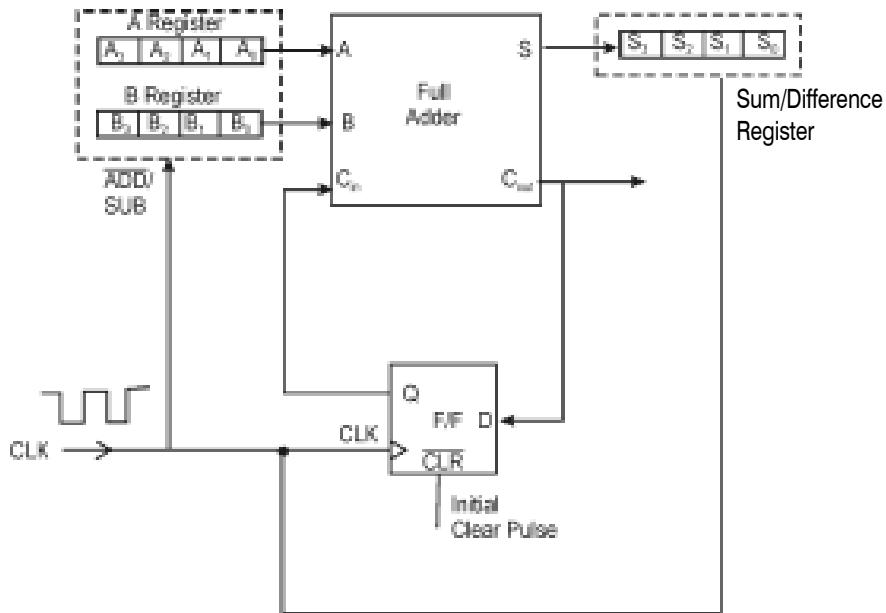


Fig. 2.22 : 4 Bit Serial Adder

Example 2.3: Solve $(0111) + (0010)$ using serial adder.

$$\text{Let, } A_3 A_2 A_1 A_0 = 0111$$

$$B_3 B_2 B_1 B_0 = 0010$$

$$C_0 = C_{in} = 0$$

- (i) Before the first clock pulse occurs, as the inputs to the full adders are $A_0 = 1, B_0 = 0$ and $C_{in} = 0$. The full adder outputs will be $S = 1, C_{out} = 0$.
 - (ii) When the first clock pulse occurs, the value in the A and B registers shift from left to right by one bit. In addition, the sum (S) is transferred to S_3 of the sum register and the C_{out} is transferred to D -flipflop, whose output becomes 0, which is the carry input (C_{in}) of the full adder.
 - (iii) Now $A_0 = 1, B_0 = 1$ and $C_{in} = 0$ and therefore, $S = 0$ and $C_{out} = 1$. When the second clock pulse occurs, A, B and Sum registers again shift right; $S = 0$ is transferred to S_3 and $C_{out} = 1$ is transferred to the D -flipflop.
 - (iv) Now $A_0 = 1, B_0 = 0, C_{in} = 1$ and therefore $S = 0, C_{out} = 1$. When the third clock pulse occurs, A, B and Sum registers shift right; $S = 0$ is transferred to S_3 and $C_{out} = 1$ is transferred to D -flipflop.
 - (v) Now $A_0 = 0, B_0 = 0, C_{in} = 1$ and therefore $S = 1, C_{out} = 0$. When the fourth clock pulse occurs, A, B and Sum registers shift right; $S = 1$ is transferred to S_3 and $C_{out} = 0$ is transferred to D -flipflop.
- At the end of the fourth clock pulse, the result will be available in the sum register as 1001 and C_{out} is 0.

Result: $0111 + 0010 = 1001$

2.11 SERIAL SUBTRACTOR

A serial subtractor can be obtained by converting the serial adder by

- (i) feeding the output \bar{B} , into the full adder instead of B .
- (ii) initially setting the D -flipflop to 1 instead of 0.
- (iii) difference register instead of sum register.

The remaining circuitry is the same as serial adder.

The serial subtractor using 2's complement subtraction method. The subtrahend is stored in the ' B ' register and the minuend is stored in ' A ' register. The subtrahend is converted into 1's complement number by adding a NOT gate with B register and get 2's complement number by adding 1 through C_{in} . The 2's complement of subtrahend is added to the minuend by the full adder and the result is stored in the difference register.

2.12 SERIAL ADDER/SUBTRACTOR

The four bit serial adder/subtractor is shown in **Figure 2.23**. When $\overline{\text{ADD}}/\text{SUB} = 0$, the uncomplemented $B_3 B_2 B_1 B_0$ will be applied to the full adder. The D-flipflop is initially cleared by applying a low pulse at CLR input and the circuit function as a 4-bit serial adder. When $\overline{\text{ADD}}/\text{SUB}=1$, the complemented output of B register ($\bar{B}_3 \bar{B}_2 \bar{B}_1 \bar{B}_0$) will be applied to the full adder. The D-flipflop is set to 1 so as to get the 2's complement of the subtrahend by applying a low pulse at $\overline{\text{PR}}$ input and thus the circuit function as a 4-bit serial subtractor.

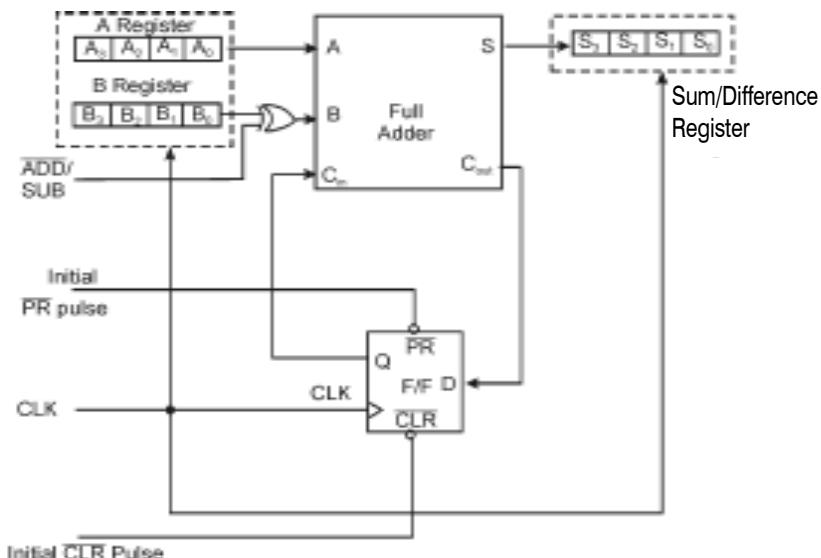


Fig. 2.23: 4 Bit Serial Adder/Subtractor

2.13 BINARY MULTIPLIER

Multiplication operation can be carried out by (i) Multipliers using partial product addition and shifting and (ii) Parallel multipliers.

Multiplier using Shift Method

To understand the multiplication process using shift method, consider the multiplication of two 4-bit binary numbers 1010 and 1011, as an example.

$$\begin{array}{r}
 1010 \rightarrow \text{Multiplicand} \\
 \times 1011 \rightarrow \text{Multiplier} \\
 \hline
 1010 \rightarrow \text{Partial product 1} \\
 1010 \rightarrow \text{Partial product 2} \\
 0000 \rightarrow \text{Partial product 3} \\
 1010 \rightarrow \text{Partial product 4} \\
 \hline
 1101110
 \end{array}$$

From the above multiplication process, one can easily understand that if the multiplier bit is 1, then the multiplicand is simply copied as a partial product; if the multiplier bit is 0, then the partial product is 0. Whenever a partial product is obtained, it is shifted one bit to the left of the previous partial product. This process is continued until all the multiplier bits are checked, and then the partial products are added to this multiplication process, i.e. multiplication by partial product addition and shifting, can be implemented using the block diagram shown in figure.

In the shown figure, the 4-bit multiplier is stored in register Y ($Y_3 Y_2 Y_1 Y_0$); the 4-bit multiplicand is stored in register M ($M_3 M_2 M_1 M_0$), and the X register ($X_4 X_3 X_2 X_1 X_0$) is initially cleared to 00000. Here, to perform multiplication, the least significant bit of the multiplier bit (Y_0) is checked whether it is 0 or 1. If $Y_0 = 1$, the number in the multiplicand register (M) is added with the least significant 4-bits of X register ($X_3 X_2 X_1 X_0$; X_4 is to store carry in addition process) and the combined X and Y register is shifted to the right by 1 bit. If $Y_0 = 0$, the combined X and Y register is shifted to the right by 1 bit without performing any addition. This process has to be repeated four times to perform 4-bit multiplication. Now, the multiplication result ($R_7 R_6 R_5 R_4 R_3 R_2 R_1 R_0$) will be available in X and Y registers ($X_3 X_2 X_1 X_0 Y_3 Y_2 Y_1 Y_0$).

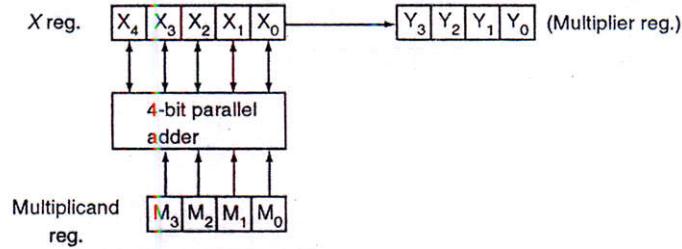


Fig.2.24: 4-bit binary multiplier using shift method

Parallel Multiplier

The 4-bit multiplier using shift method requires 4 cycles of addition and shifting operations, but it requires only a single 4-bit parallel adder. The speed of multiplication process can be increased considerably in parallel multiplier at the extra cost of increased hardware. The circuit diagram for a 4-bit parallel mutliplier is shown in figure.

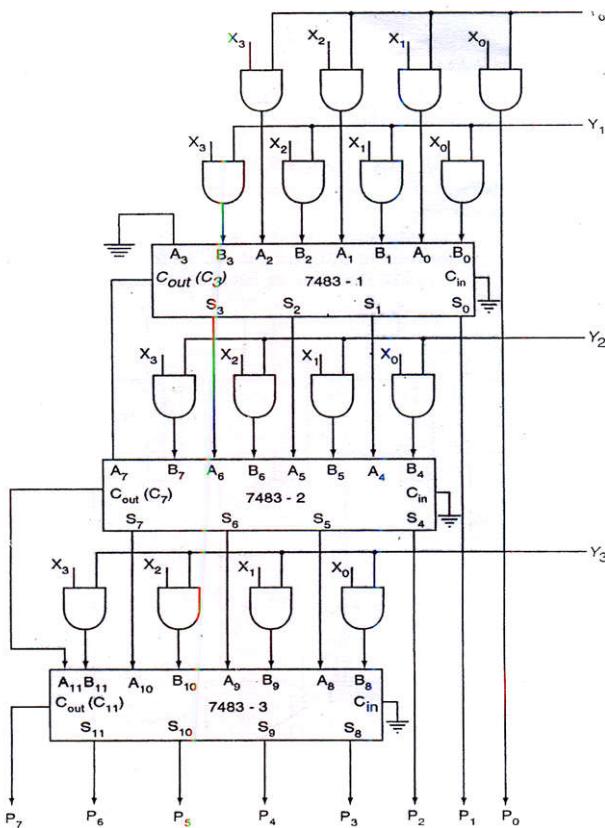


Fig.2.25:4-bit parallel multiplier

It requires three 4-bit parallel binary adders and 16 numbers of 2-input AND gates. Here, each group of 4 AND gates is used to obtain partial products while 4-bit parallel adders are used to add the partial products. Since the generation of partial products and their additions are performed in parallel in the group of AND gates and 4-bit adders respectively, the multiplication result ($P_7P_6P_5P_4P_3P_2P_1P_0$) will be available at the output immediately after the propagation delay in the multiplier circuit.

The operation of the parallel multiplier can be understood in a better manner from the symbolic form of binary multiplication process shown in figure.

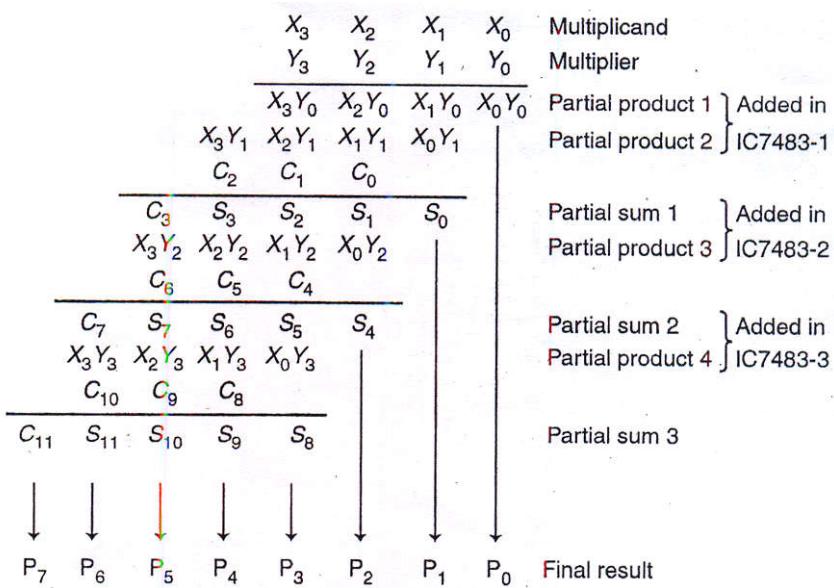


Fig.2.26: Symbolic form of binary multiplication process

2.14 CARRY LOOK AHEAD ADDER

In parallel adder, all the bits of the augend and the addend are available for computation at the same time. The carry output of each full-adder stage is connected to the carry input of the next higher-order stage. Since each bit of the sum output depends on the value of the input carry, time delay occurs in the addition process. This time delay is called as **carry propagation delay**.

For example, addition of two numbers (0011 + 0101) gives the result as 1000. Addition of the LSB position produces a carry into the second position. This carry when added to the bits of the second position, produces a carry into the third position. This carry when added to the bits of the third position, produces a carry into the last position. The sum bit generated in the last position (MSB) depends on the carry that was generated by the addition in the previous positions. i.e., the adder will not produce correct result until LSB carry has propagated through the intermediate full-adders. This represents a time delay that depends on the propagation delay produced in each full-adder. For example if each full adder is considered to have a propagation delay of 30 ns, then S_3 will not react its correct value until 90 ns after LSB carry is generated. Therefore total time required to perform addition is $90 + 30 = 120$ ns.

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

$$\text{Carry generate, } G_i = A_i B_i$$

$$\text{Carry propagate, } P_i = A_i \oplus B_i$$

$$\text{Sum, } S_i = P_i \oplus C_i$$

$$= A_i \oplus B_i \oplus C_i$$

$$\text{Carry, } C_{i+1} = G_i + P_i C_i$$

The structure of one stage of a carry look ahead adder is shown in **Figure 2.27**.

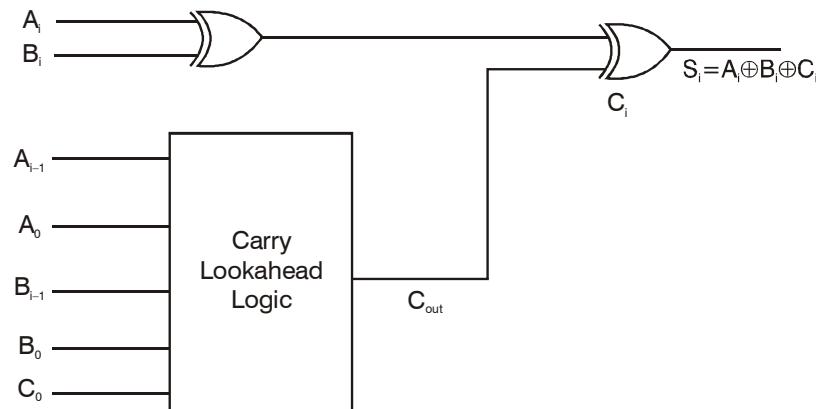


Fig. 2.27 : Structure of one stage of a carry look ahead adder

G_i (carry generate) generates carry if both A_i and B_i are 1 regardless of the input carry.

P_i (carry propagate) propagates carries if atleast one of its addend bits is 1. The carry output of a stage can now be written in terms of the generate and propagate signals:

$$C_{i+1} = G_i + P_i C_i$$

To eliminate carry ripple, expand the C_i term for each stage and multiply out to obtain a 2-level AND-OR expression. Using this technique, we can obtain the following carry equations for the first four adder stages are obtained.

$$C_1$$

$$C_2 = G_1 + P_1 C_1$$

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

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

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

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

From these equations, it can be seen that C_4 does not have to wait for C_3 and C_2 to propagate. Infact, C_4 is propagated at the same time as C_2 and C_3 . **Figure 2.28** shows the implementation of carry equations for C_2 , C_3 and C_4 using AND-OR logic.

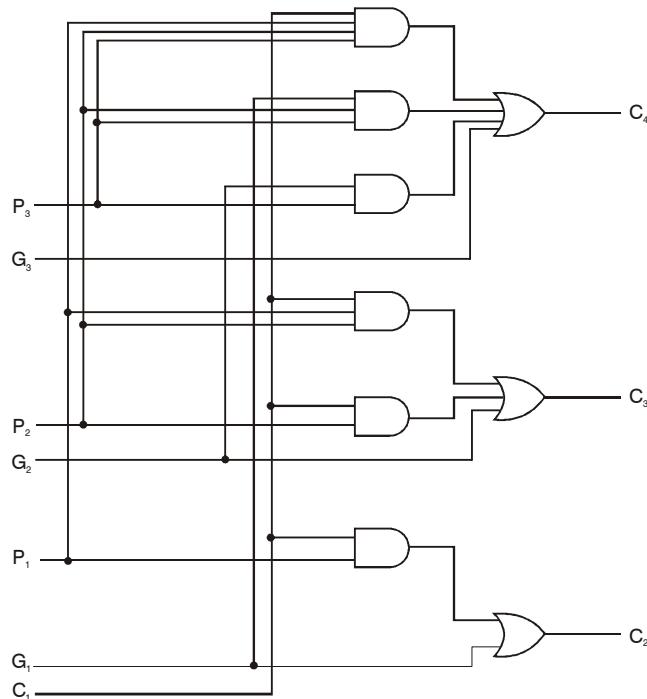


Fig. 2.28 : Logic diagram of a look ahead carry generator

2.15 BCD ADDER

The BCD adder is used to add two BCD digits and produces a sum in BCD digit. We know that, BCD number means 0 to 9 (10 digits) and are represented in the binary form 0000 to 1001.

BCD numbers cannot be greater than 9 and 10 is represented in BCD as 0001 0000. In BCD addition, the sum is greater than 9(1001), we obtain a non-valid BCD representation. The addition of binary 6(0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.

Let us consider BCD addition of 5 and 10,

$$5 \Rightarrow 0101$$

$$10 \Rightarrow 1010 (+)$$

$$15 \Rightarrow \underline{1111}$$

1111 is an invalid BCD number. It can be corrected by the addition of 6(0110) to the invalid BCD number.

$$15 \Rightarrow \quad \quad 1111$$

$$6 \Rightarrow \quad \quad \underline{0110} \quad (+)$$

$$\underline{\underline{0001 \ 0101}} \Rightarrow 15 \text{ (BCD)}$$

Thus we can summarize the BCD addition procedure as follows:

1. Add two BCD numbers using ordinary binary addition.
2. If the result is equal to or less than 9, no correction is needed. The sum is in correct BCD form.
3. If the result is greater than 9 or if a carry is generated from the result, the result is invalid and the correction is needed.
4. To correct the invalid sum add 6(0110) to the result. If a carry results from this addition, add it to the next higher order BCD digit.

Implementation of BCD adder using logic circuit as follows:

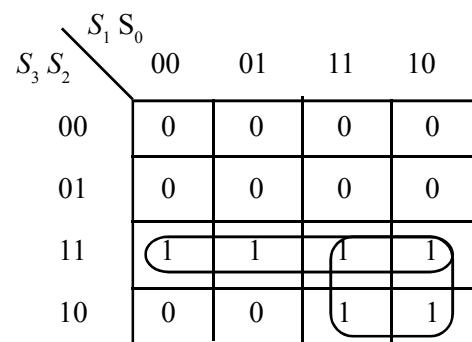
- ◆ 4-bit binary adder for initial addition.
- ◆ Logic circuit to detect sum greater than 9.
- ◆ Second 4-bit binary adder to add 6(0110) to the result if result is greater than 9 or carry is 1.

The logic circuit to detect result greater than 9 can be determined by simplifying the boolean expression of given **Table 2.4**.

TABLE 2.4: Truth Table

Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Draw the K-map for the above truth table and find out the expression



$$Y = S_3 S_2 + S_3 S_1$$

This Binary to BCD correction expression is shown in **Figure 2.29**.

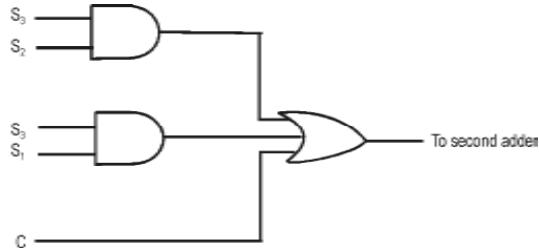


Fig. 2.29 : Binary to BCD Correction

In first binary adder, two BCD numbers together with input carry are added. When the result is equal to zero (i.e., result ≤ 9 or $C_{out} = 0$), nothing is added to the result. When the result is one (i.e., result ≥ 9 or $C_{out} = 1$) binary 0110 is added to the result through second binary adder. The C_{out} generated by second adder can be ignored, since it supplies information already available at the output carry terminal. The single digit BCD adder is shown in **Figure 2.30**. Multiple digit BCD adders can be constructed by cascading as many single digit adders as needed. The BCD carry-out from each stage would be connected to the carry-in of the next higher order stage.

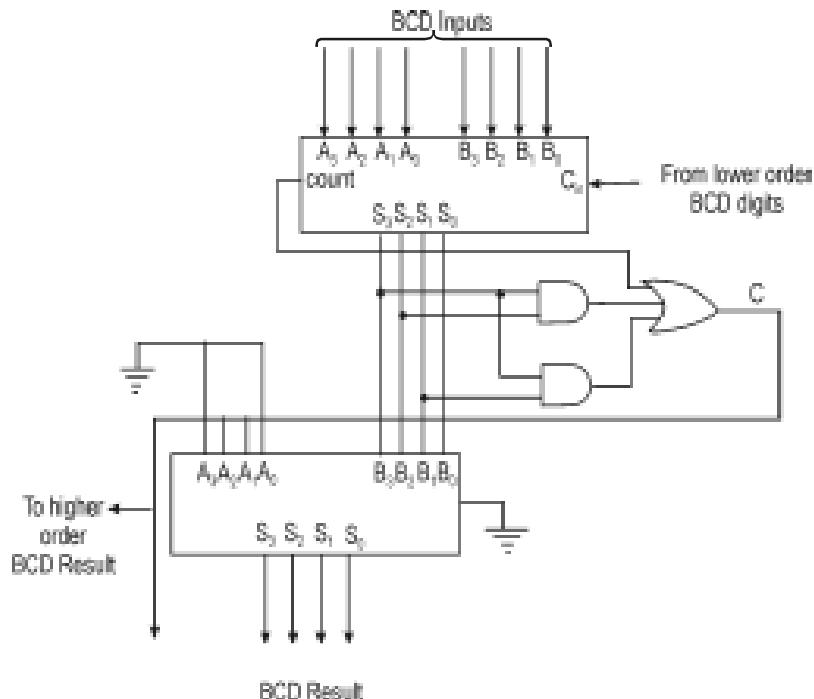


Fig. 2.30 : Single Digit BCD Adder

2.16 MAGNITUDE COMPARATOR

A magnitude comparator is a combinational circuit that compares the magnitude of two numbers A and B and generates one of the following outputs:

$$A = B$$

$$A < B$$

$$A > B .$$

The block diagram of n-bit magnitude comparator is shown in **Figure 2.31**.

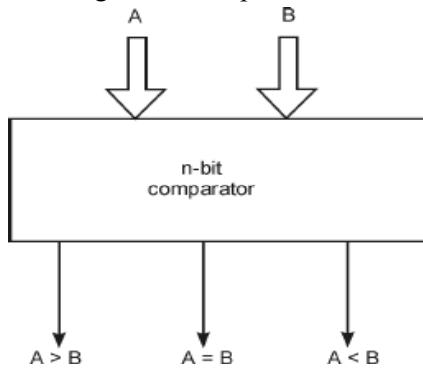


Fig. 2.31 : Block diagram of n-bit magnitude comparator

To implement the magnitude comparator, the EX-NOR gates and AND gates are used. The EX-NOR gate is used to find whether the two binary digits are equal or not, and the AND gates are used to find whether a binary digit is less than or greater than another binary digit as shown in **Figure 2.32**. **Figure 2.33** shows a single bit magnitude comparator.

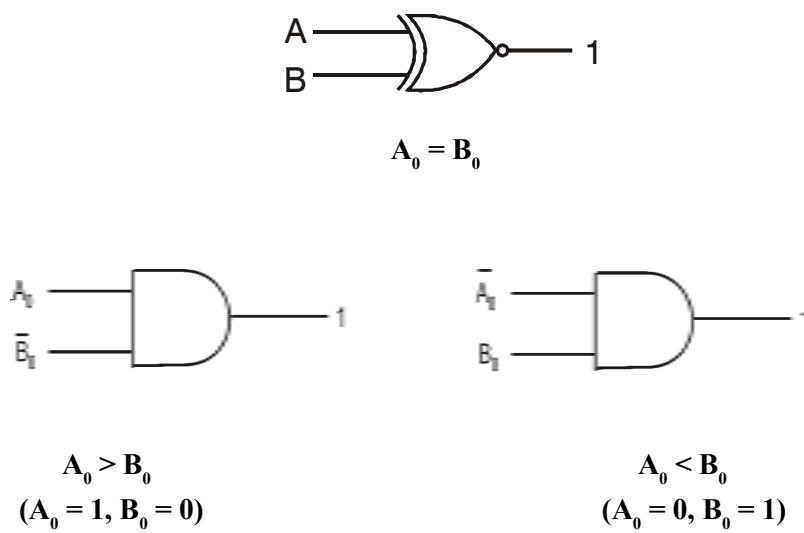


Fig. 2.32 : Comparator Operation

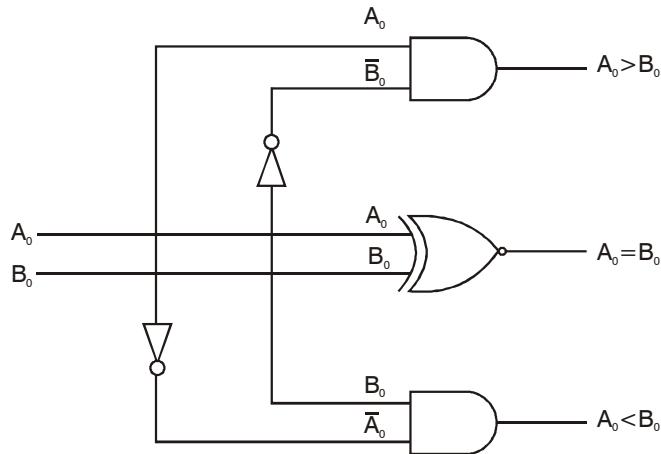


Fig. 2.33: Single bit magnitude comparator

The same principle can be extended to an n-bit magnitude comparator. The design of a 2 bit magnitude comparator is as follows:

The truth table for 2 bit comparator is given in **Table 2.5**.

TABLE 2.5 : Comparator Truth Table

Inputs				Outputs		
A ₁	A ₀	B ₁	B ₀	A > B	A = B	A < B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

For $A > B$

$A_1 A_0$	$B_1 B_0$	00	01	11	10
00		0	0	0	0
01		1	0	0	0
11		1	1	0	1
10		1	1	0	0

$$A_0 \bar{B}_1 \bar{B}_0 + A_1 \bar{B}_1 + A_1 A_0 \bar{B}_0$$

For $A < B$

$A_1 A_0$	$B_1 B_0$	00	01	11	10
00		0	1	1	1
01		0	0	1	1
11		0	0	0	0
10		0	0	1	0

$$\bar{A}_0 B_1 B_0 + \bar{A}_1 B_1 + \bar{A}_1 \bar{A}_0 B_0$$

$A_1 A_0$	$B_1 B_0$	00	01	11	10
00		(1)	0	0	0
01		0	(1)	0	0
11		0	0	(1)	0
10		0	0	0	(1)

$$\bar{A}_1 \bar{A}_0 \bar{B}_1 \bar{B}_0 + \bar{A}_1 A_0 \bar{B}_1 B_0 + A_1 A_0 B_1 B_0 + A_1 \bar{A}_0 B_1 \bar{B}_0$$

$$= \bar{A}_1 \bar{B}_1 (\bar{A}_0 \bar{B}_0 + A_0 B_0) \bar{A}_1 B_1 (A_0 B_0 + \bar{A}_0 \bar{B}_0)$$

$$= (\bar{A}_1 \bar{B}_1 + A_1 B_1) (\bar{A}_0 \bar{B}_0 + A_0 B_0) = (\overline{A_1 \oplus B_1}) (\overline{A_0 \oplus B_0})$$

The expressions for determining whether,

$$A > B \text{ is } A_0 \bar{B}_1 \bar{B}_0 + A_1 \bar{B}_1 + A_1 A_0 \bar{B}_0$$

$$A < B \text{ is } \bar{A}_0 B_1 B_0 + \bar{A}_1 B_1 + \bar{A}_1 \bar{A}_0 B_0$$

$$A = B \text{ is, } (\overline{A_1 \oplus B_1}) (\overline{A_0 \oplus B_0})$$

The implementation of 2 bit magnitude comparator using EX-NOR and AND gates using the above expressions is shown in **Figure 2.34**.

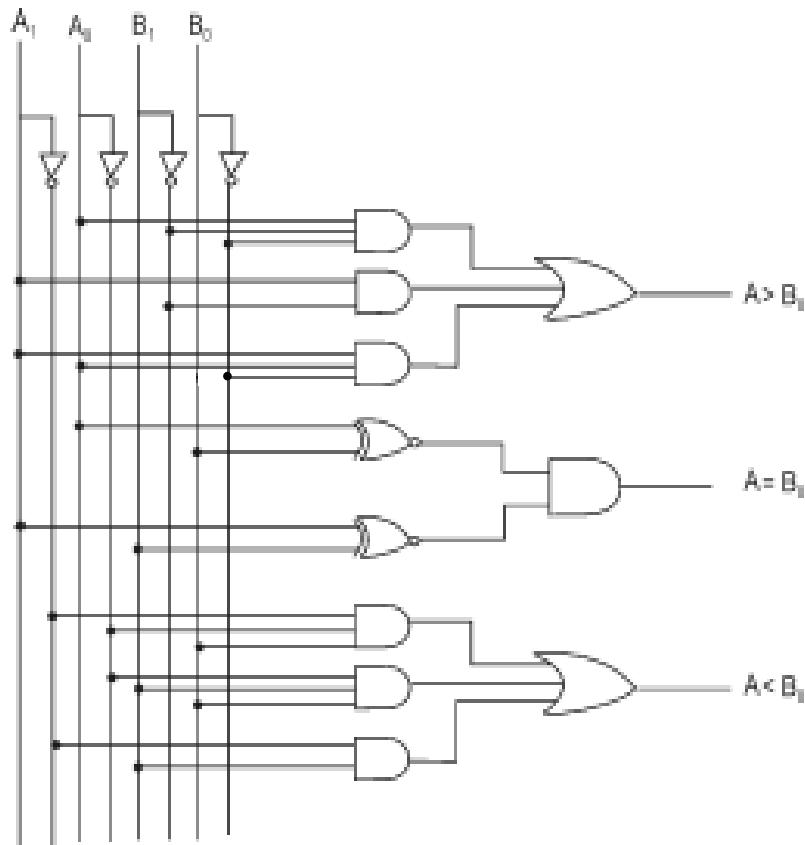


Fig. 2.34 : 2-bit Magnitude comparator

2.17 PARITY GENERATOR AND CHECKER

A parity bit is used for the purpose of detecting errors during transmission of binary information. A parity bit is an extra bit included with a binary message to make the number of 1s either odd or even. The message including the parity bit is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted. The circuit that generates the parity bit in the transmitter is called a parity generator and the circuit that checks the parity bit in the receiver is called a parity checker.

Even parity means an ‘n’ bit input has an even number of 1s. For example, 110101 has even parity because it contains four 1s.

Odd parity means an ‘n’ bit input has an odd number of 1s. For example, 110100 has odd parity because it contains three 1s.

2.17.1 Parity Checker

Exclusive-OR gates are used for checking the parity of a binary number because they produce an output 1 when the input has an odd number of 1s. Therefore, an even parity input to an EX-OR gate produces a low output, while an odd parity input produces a high output. Remember the truth table for EX-OR gate as given below:

<i>Inputs</i>	<i>Outputs</i>
0 0	0
0 1	1
1 0	1
1 1	0

Figure 2.35 shows a 4 bit parity checker. The output is 1 when the number of 1s in the inputs is odd and output is 0 when the number of 1s in the inputs is even. For example, when 1001 is the input given to the 4 bit parity generator, the output is 0, because the number has two 1s.

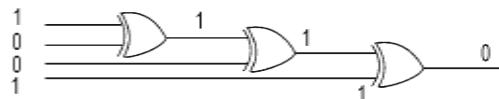


Fig. 2.35: 4 bit parity checker

Figure 2.36 shows a 16 input EX-OR gate. A 16 bit number drives the input. The EX-OR gate produces an output 1 because the input (1010101010101000) has odd parity.

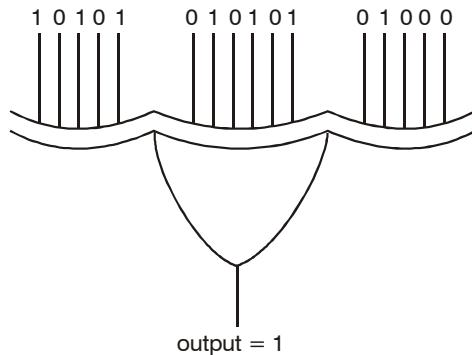


Fig. 2.36: 16 bit parity checker

2.17.2 Parity Generator

Figure 2.34 shows the odd-parity generator. Let the 8 bit binary number,

$$X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0 = 0100\ 0001$$

Then the number has even parity, which means the EX-OR gate produces an output of 0. Because of the inverter $X_8 = 1$ and the final 9-bit output is,

$$1\ 0100\ 0001$$

This 9 bit output has odd parity.

Suppose the input is 0110 0001. Now it has odd parity. The EX-OR produces an output 1. But the inverter produces a 0 ($X_8 = 0$), so that the final 9 bit output is,

0 0100 0001.

Again the final output has odd parity,

Thus odd-parity generator produces a 9 bit output number with odd parity. If the 8 bit input has even parity, a 1 comes out of the inverter to produce a final output with odd parity. If the 8 bit input has odd parity, a 0 comes out of the inverter and the final 9 bit output again has odd parity.

To get **even parity generator** delete the inverter in the circuit shown in **Figure 2.37**.

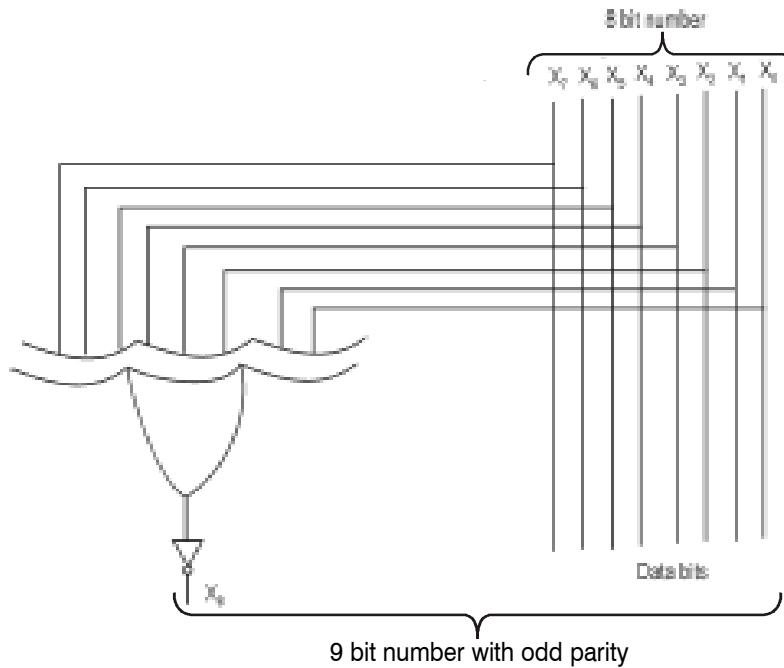


Fig. 2.37 : Odd-parity generator

2.18 CODE CONVERTERS

A code converter is a logic circuit that changes data presented in one type of binary code to another type of binary code. Now we will discuss about some code converters.

2.18.1 Binary to BCD Converter

Table 2.6 shows the binary codes and corresponding BCD codes.

(Dec. 2005)

TABLE 2.6 : Truth Table for Binary to BCD converter

Binary Code				BCD code				
D	C	B	A	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0	1
0	1	1	0	0	0	1	1	0
0	1	1	1	0	0	1	1	1
1	0	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	1	0	0	0	0
1	0	1	1	1	0	0	0	1
1	1	0	0	1	0	0	1	0
1	1	0	1	1	0	0	1	1
1	1	1	0	1	0	1	0	0
1	1	1	1	1	0	1	0	1

For B_0 For B_1

DC \ BA	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

$$B_0 = A$$

DC \ BA	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	1	0	0
10	0	0	0	0

$$B_1 = DC\bar{B} + \bar{D}\bar{B}$$

For B_2 For B_3

DC \ BA	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	1	1
10	0	0	0	0

$$B_2 = \bar{D}C + CB$$

DC \ BA	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	1	1	0	0

$$B_3 = D\bar{C}\bar{B}$$

For B_4

		00	01	11	10
		00	0	0	0
		01	0	0	0
		11	1	1	1
		10	0	0	1

$B_4 = DC + DB$

$$B_0 = A$$

$$B_1 = DC\bar{B} + \bar{D}B$$

$$B_2 = \bar{D}\bar{C} + CB$$

$$B_3 = D\bar{C}\bar{B}$$

$$B_4 = DC + DB$$

Logic Diagram

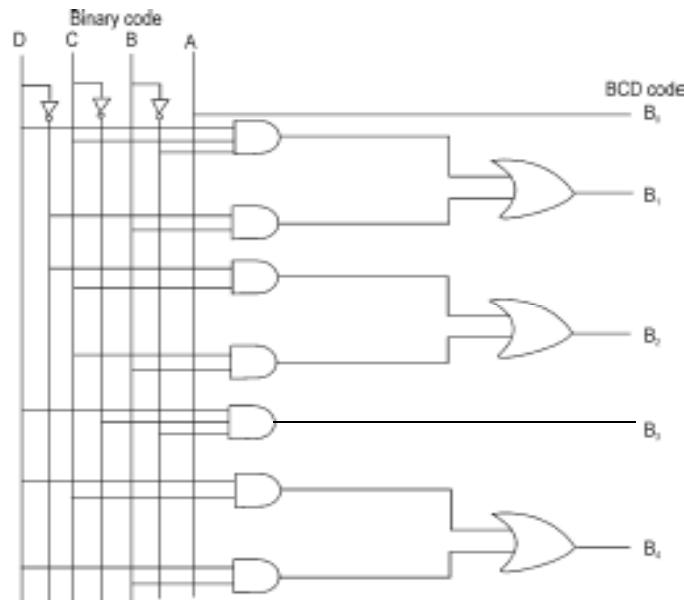


Fig. 2.38: Binary to BCD converter

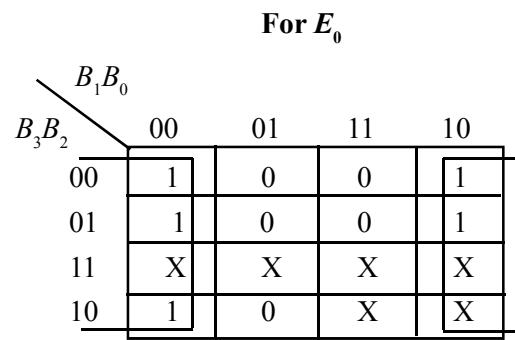
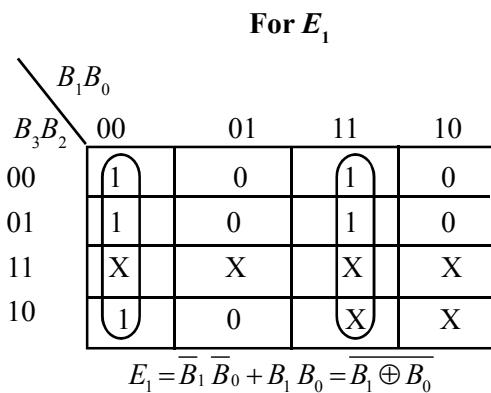
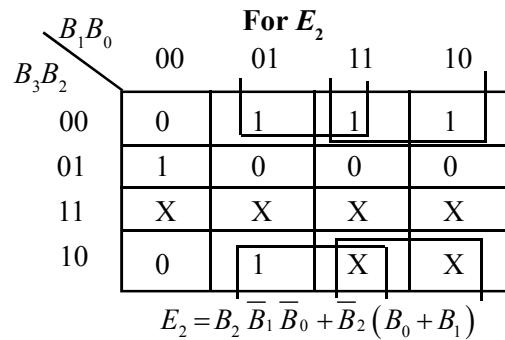
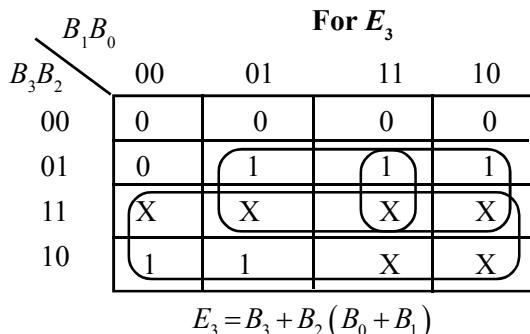
2.18.2 BCD to Excess 3 Converter

(April 2004)

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. The truth table for BCD to excess 3 code converter is shown in **Table 2.7**.

TABLE 2.7 : Truth Table for BCD to Excess 3 converter

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

K-map Simplification

$$E_3 = B_3 + B_2(B_0 + B_1)$$

$$E_2 = B_2 \bar{B}_1 \bar{B}_0 + \bar{B}_2(B_0 + B_1)$$

$$E_1 = \bar{B}_1 \oplus B_0$$

$$E_0 = \bar{B}_0$$

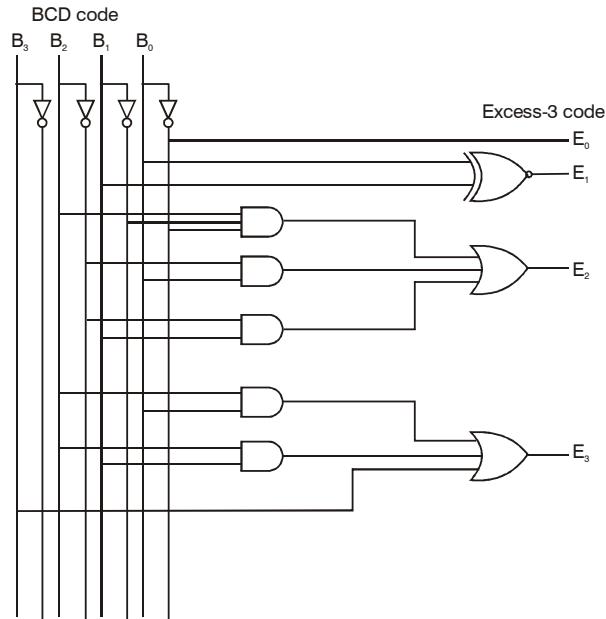
BCD code

Fig. 2.39: BCD to Excess-3 code converter

2.18.3 Binary to Gray Code Converter

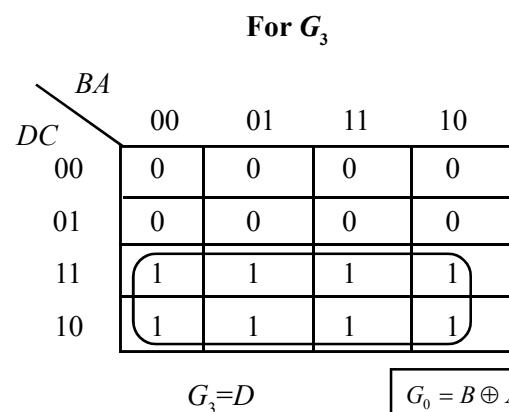
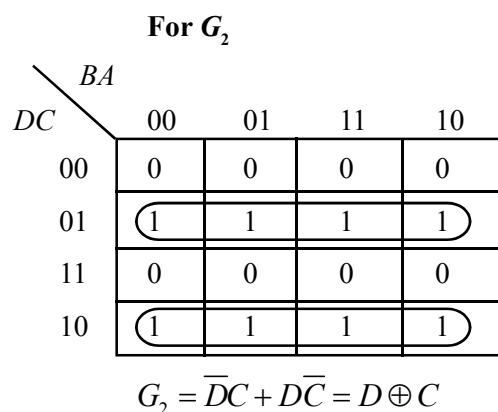
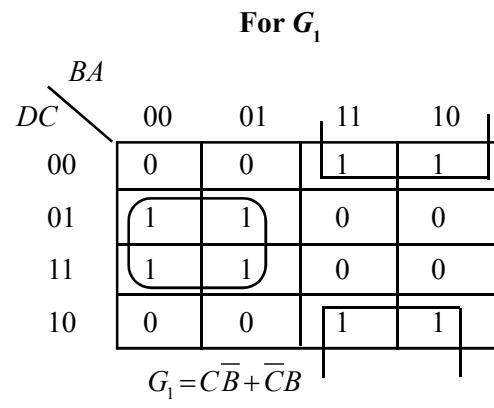
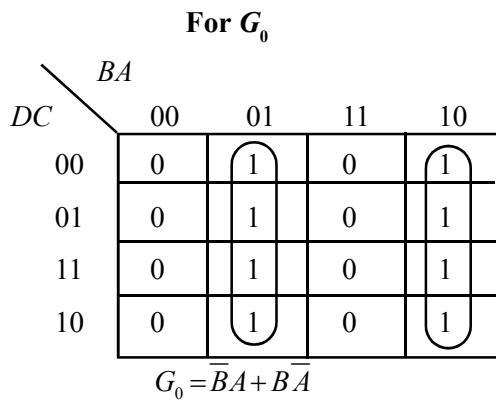
(Dec. 2005)

The Gray code is often used in digital systems because it has the advantage that only one bit in the numerical representation changes between successive numbers. **Table 2.8** shows decimal and Binary codes and corresponding Gray code.

TABLE 2.8

Decimal	Binary code				Gray code			
	D	C	B	A	G ₃	G ₂	G ₁	G ₀
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

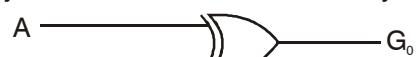
K-map Simplification



Logic Diagram

Binary Code

A ——————



Gray Code

G_0 ——————

G_1 ——————

G_2 ——————

G_3 ——————

$$\boxed{\begin{aligned} G_0 &= B \oplus A \\ G_1 &= C \oplus B \\ G_2 &= D \oplus C \\ G_3 &= D \end{aligned}}$$

Fig. 2.40 : Binary to gray code converter

2.18.4 Gray Code to Binary Code Converter

Table 2.9 shows the truth table for gray code to binary code converter.

TABLE 2.9 : Truth Table for gray code to binary code converter

Gray code				Binary code			
G_3	G_2	G_1	G_0	D	C	B	A
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

K-map Simplification

For A

		$G_1 G_0$				
		$G_3 G_2$	00	01	11	10
G_3	G_2	00	0	(1)	0	(1)
		01	(1)	0	(1)	0
		11	0	(1)	0	(1)
		10	(1)	0	(1)	0

For B

		$G_1 G_0$				
		$G_3 G_2$	00	01	11	10
G_3	G_2	00	0	0	(1)	(1)
		01	(1)	(1)	0	0
		11	0	0	(1)	(1)
		10	(1)	(1)	0	0

$$\begin{aligned}
 A &= (\overline{G}_3 G_2 + G_3 \overline{G}_2) \overline{G}_1 \overline{G}_0 + (\overline{G}_3 \overline{G}_2 + G_3 G_2) \overline{G}_1 G_0 + (\overline{G}_3 G_2 + G_3 \overline{G}_2) G_1 G_0 + (\overline{G}_3 \overline{G}_2 + G_3 G_2) G_1 \overline{G}_0 \\
 &= (G_3 \oplus G_2) \overline{G}_1 \overline{G}_0 + (G_3 \square G_2) \overline{G}_1 G_0 + (G_3 \oplus G_2) G_1 G_0 + (G_3 \square G_2) G_1 \overline{G}_0 \\
 &= (G_3 \oplus G_2)(\overline{G}_1 \overline{G}_0 + G_1 G_0) + (G_3 \square G_2)(\overline{G}_1 G_0 + G_1 \overline{G}_0) \\
 &= (G_3 \oplus G_2)(G_2 \square G_0) + (G_3 \square G_2)(G_1 \oplus G_0) \\
 &= (G_3 \oplus G_2)(\overline{G}_1 \oplus \overline{G}_0) + (G_3 \oplus G_2)(G_1 \oplus G_0) \\
 &= (G_3 \oplus G_2) \oplus (G_1 \oplus G_0) \\
 B &= (\overline{G}_3 \overline{G}_2 + G_3 G_2) G_1 + (\overline{G}_3 G_2 + G_3 \overline{G}_2) \overline{G}_1 \\
 &= (G_3 \square G_2) G_1 \oplus (G_3 \oplus G_2) \overline{G}_1 \\
 &= (\overline{G}_3 \oplus G_2) G_1 + (G_3 \oplus G_2) \overline{G}_1 = G_3 \oplus G_2 \oplus G_1
 \end{aligned}$$

$$\begin{aligned}
 A &= (G_3 \oplus G_2) \oplus (G_1 \oplus G_0) \\
 B &= G_3 \oplus G_2 \oplus G_1 \\
 C &= G_3 \oplus G_2 \\
 D &= G_3
 \end{aligned}$$

For C

		G_1	G_0	00	01	11	10	
		G_3	G_2	00	0	0	0	0
				01	1	1	1	1
				11	0	0	0	0
				10	1	1	1	1

$C = \overline{G}_3 G_2 + G_3 \overline{G}_2 = G_3 \oplus G_2$

For D

		G_1	G_0	00	01	11	10	
		G_3	G_2	00	0	0	0	0
				01	0	0	0	0
				11	1	1	1	1
				10	1	1	1	1

$D = G_3$

Logic Diagram

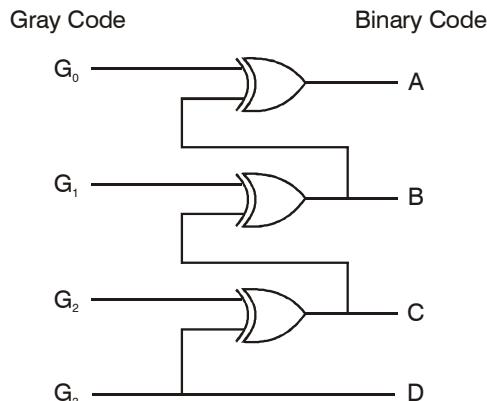


Fig. 2.41 : Gray code to binary code converter

2.19 DECODERS

Decoder is a digital device that converts coded information into another code or non-coded form. It is a multi-input multi-output logic circuit. The number of outputs is greater than the number of inputs ($n : 2^n$). The encoded information is presented as ' n ' inputs producing 2^n possible outputs as shown in **Figure 2.42**.

If the number of inputs and outputs are equal in a digital system, then it can be called as code converters. (BCD to XS-3 code, Binary to BCD code, Gray to Binary Code converters, etc.)



Fig. 2.42: Decoder

2.19.1 Binary Decoder

A binary decoder has ' n ' bit binary input and a one activated output out of 2^n outputs. A binary decoder is used when it is necessary to activate exactly one of 2^n outputs based on an n -bit input value. **Figure 2.43** shows 2 to 4 line decoder. 2 inputs are decoded into 4 outputs, each output representing one of the minterms of the 2 input variables. The **Table 2.10** shows the truth table for 2 to 4 line decoder.

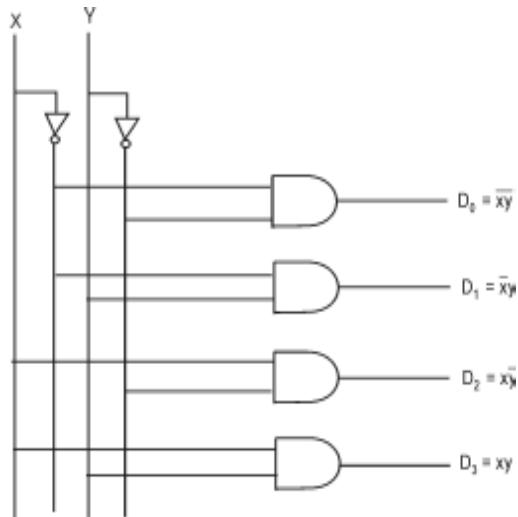


Fig. 2.43: 2 to 4 line decoder

TABLE 2.10: Truth Table

<i>Inputs</i>		<i>Outputs</i>			
<i>X</i>	<i>Y</i>	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

2.19.2 3-to-8 line Decoder

A 3-to-8 line decoder has three inputs (x, y, z) and eight outputs (D_0-D_7). Based on the 3 inputs one of the 8 outputs is selected.

The logic diagram of 3-to-8 line decoder is shown in **Figure 2.44** and the truth table of this decoder is given in **Table 2.11**. The three inputs are decoded into eight outputs, each output representing one of the minterms of the 3-input variables. This decoder is used for binary-to-octal conversion. The input variables may represent a binary number and the outputs will represent the eight digits in the octal number system. The output variables are mutually exclusive because only one output can be equal to 1 at any one time. The output line whose value is equal to 1 represents the minterm equivalent of the binary number presently available in the input lines.

TABLE 2.11: Truth Table

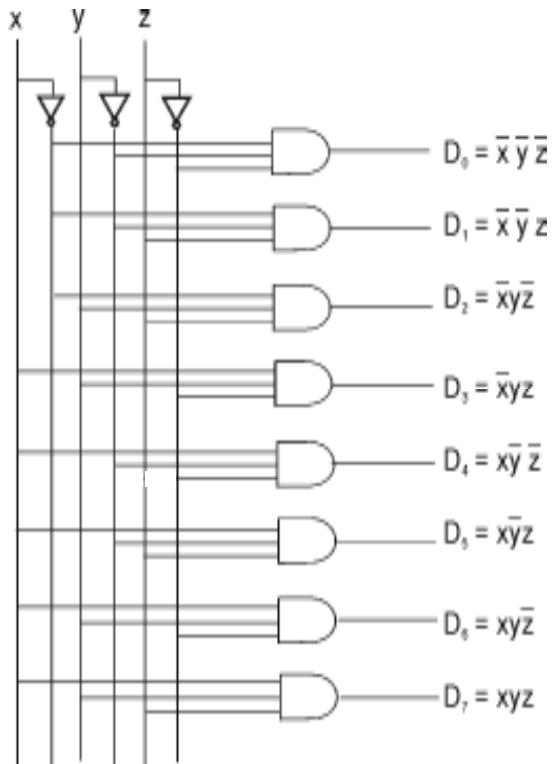


Fig. 2.44: 3-to-8 line decoder

2.19.3 1-of-16 Decoder

The 1-of-16 decoder is shown in **Figure 2.45**. The 4 inputs ABCD are the control bits. It has 16 output lines and only 1 of the 16 output lines is high. For instance, when $ABCD = 0001$, only the Y_1 AND gate has all inputs high, therefore only the Y_1 output is high. If $ABCD = 0100$, only the Y_4 AND gate has all input high, therefore only the Y_4 output goes high. This circuit is also known as 4-to-16 line decoder.

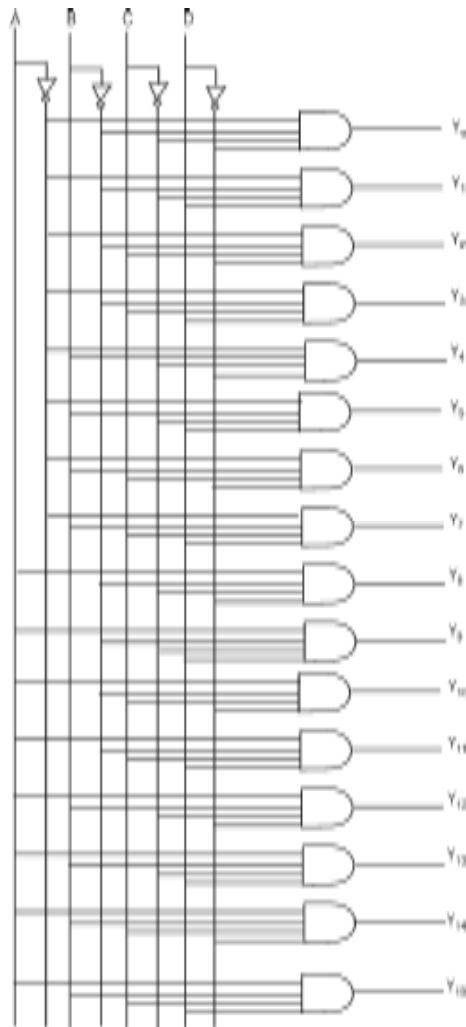


Fig. 2.45: 1-of-16 [4-to-16 line] Decoder

2.19.4 BCD-to-Decimal Decoder

The **Figure 2.46** shows a 1-of-10-decoder because only one of the 10 output lines is high. For example, when $ABCD = 0011$, only the Y_3 AND gate has all high inputs, therefore only the Y_3 output is high. If $ABCD$ changes to 1000, only the Y_8 AND gate has all high inputs, therefore only the Y_8 output goes high. The $ABCD$ possibilities are from 0000 to 1001 (9). The high output always equals the decimal equivalent of the input BCD digit. For this reason, this circuit is also called a BCD-to-Decimal converter.

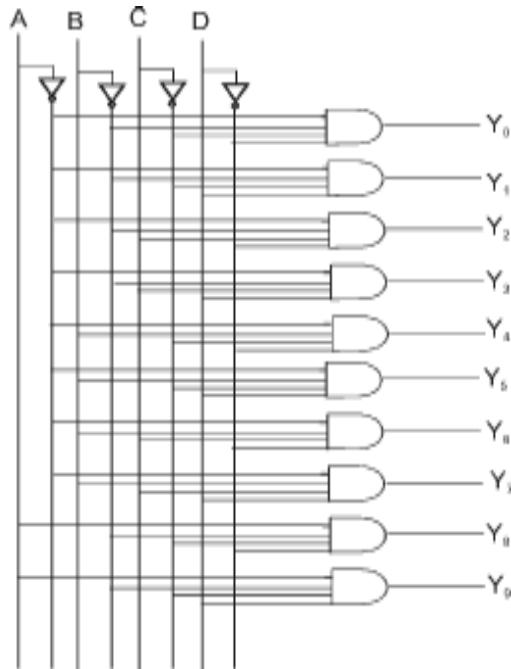


Fig. 2.46: BCD-to-Decimal Decoder

2.20 ENCODERS

An encoder is a digital circuit that performs the inverse operation of a decoder. It is a combinational logic circuit, that output logic circuit, that output lines generate the binary code corresponding to the input value.

It has 2^n input lines and ' n ' output lines. The octal-to-binary encoder has $8(2^3)$ inputs, one for each of the octal digits and three ($n = 3$) outputs that generate the corresponding binary number. It is assumed that only one input has a value of 1 at any given time; otherwise the circuit has no meaning.

2.20.1 Octal-to-Binary encoder

The octal-to-binary encoder truth table is given in **Table 2.12**. The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1 when the input octal digit is 1 or 3 or 5 or 7. Output y is 1 for octal digits 2, 3, 6 or 7 and output x is 1 for digits 4, 5, 6 or 7. These conditions can be expressed by the following output Boolean functions:

$$z = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$x = D_4 + D_5 + D_6 + D_7$$

The octal-to-binary encoder is implemented for these Boolean functions using OR gates. The octal-to-binary encoder is shown in **Figure 2.47**.

TABLE 2.12 : Truth Table

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

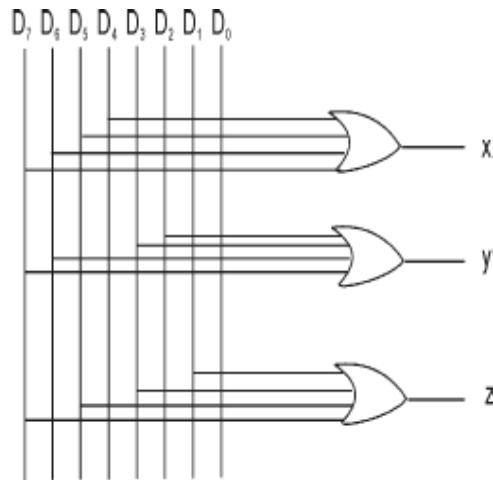


Fig. 2.47: Octal-to-Binary Encoder

2.20.2 Priority Encoder

A priority encoder is an encoder circuit that includes the priority function. The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence. The truth table of a 4 input priority encoder is given in **Table 2.13**.

Input D_3 has the highest priority. When $D_3 = 1$, the output $XY = 11$. D_2 has the next priority level. If $D_2 = 1, D_3 = 0$, the output $xy = 10$, regardless of the values of the other two lower priority inputs. The output for D_1 is generated only if higher-priority inputs are 0 and so on down the priority level.

The output V (Valid-output indicator) = 1 only when one or more inputs are equal to 1. $V=0$, if all inputs are 0 and the other two outputs (X and Y) of the circuit are not used.

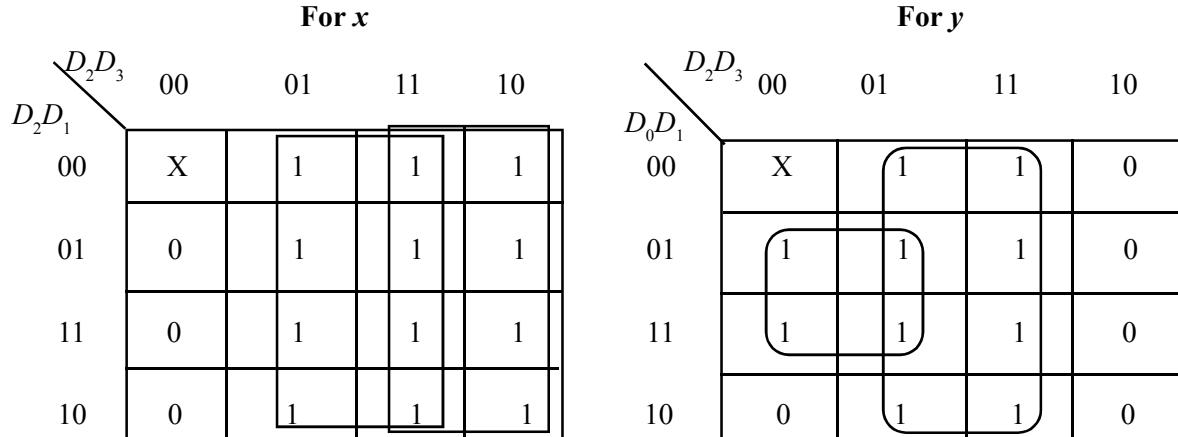
TABLE 2.13 : Truth Table of a Priority Encodes

Inputs				Outputs		
D_0	D_1	D_2	D_3	X	Y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Although the **Table 2.13** has only five rows, when each don't care condition is replaced first by 0 and then by 1, we obtain all 16 possible input combinations. For example the third row in the table with X100 represents minterms 0100 and 1100. The don't care condition is replaced by 0 and 1 as shown in **Table 2.14**.

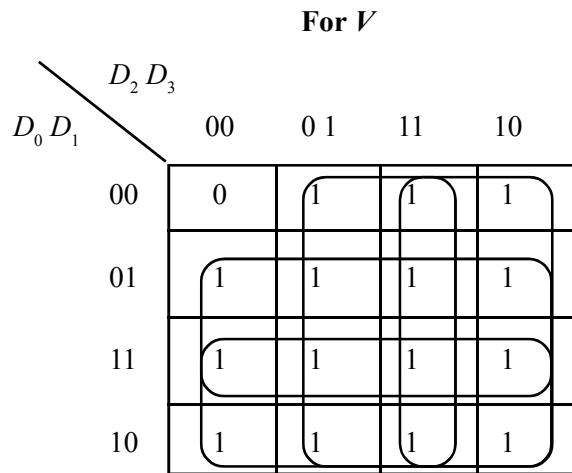
TABLE 2.14: Modified Truth Table

Inputs				Outputs		
D_0	D_1	D_2	D_3	X	Y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
0	1	0	0			
1	1	0	0	0	1	1
0	0	1	0			
0	1	1	0			
1	0	1	0	1	0	1
1	1	1	0			
0	0	0	1			
0	0	1	1			
0	1	0	1			
0	1	1	1			
1	0	0	1	1	1	1
1	0	1	1			
1	1	0	1			
1	1	1	1			

2.46*Digital Principles and System Design*

$$x = D_2 + D_3$$

$$y = D_3 + D_1 \bar{D}_2$$



$$V = D_0 + D_1 + D_2 + D_3$$

The simplified Boolean expressions for the priority encoder are obtained from the K-maps as follows:

$$x = D_2 + D_3$$

$$y = D_3 + D_1 \bar{D}_2$$

$$v = D_0 + D_1 + D_2 + D_3$$

The priority encoder is implemented in **Figure 2.48** according to the above Boolean functions.

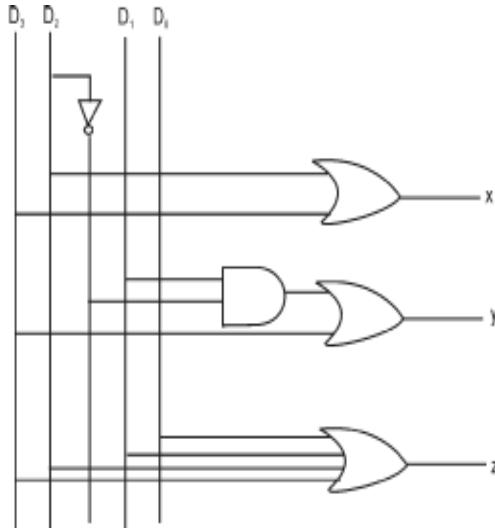


Fig. 2.48: 4 input priority encoder

2.21 MULTIPLEXERS

Multiplex means “many into one”. A multiplexer is a combinational circuit with many inputs but only one output. By applying control signals, we can steer any input to the output. It has ‘ n ’ input signals, ‘ m ’ control signals and 1 output signal. Multiplexer is called as **data selector** or because the output bit depends on the input data bit that is selected. The block diagram of multiplexer is shown in **Figure 2.49**.

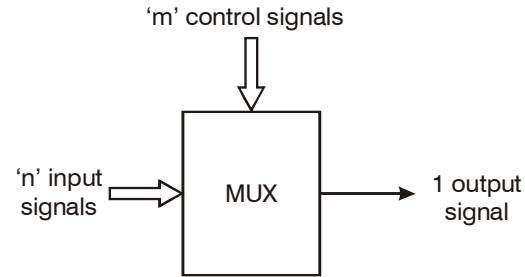


Fig. 2.49

2.21.1 4-to-1 Line Multiplexer

A 4-to-1 line multiplexer has four (n) input lines, two (m) select lines and one output line. The selection (control) lines decide the number of input lines. If the number of ‘ n ’ input lines is equal to 2^m , then ‘ m ’ select lines are required to select one of the ‘ n ’ input lines.

The logic symbol of a 4-to-1 multiplexer is shown in **Figure 2.50**. If has 4 input lines (I_0 to I_3), two select lines (S_0, S_1) and a single output line. The function table of 4-to-1 multiplexer is shown in **Table 2.15**.

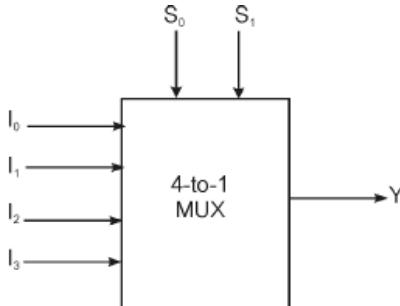


Fig. 2.50: Logic Symbol

TABLE 2.15 : Function Table

Select Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$$1. \quad Y = I_0 \text{ iff } S_1 = 0, S_0 = 0$$

$$\therefore Y = I_0 \bar{S}_1 \bar{S}_0 = I_0 \cdot 1 \cdot 1 = I_0$$

$$2. \quad Y = I_1 \text{ iff } S_1 = 0, S_0 = 1$$

$$\therefore Y = I_1 \bar{S}_1 S_0 = I_1, \text{ when } S_1 S_0 = 01$$

$$3. \quad Y = I_2 \text{ iff } S_1 = 1, S_0 = 0$$

$$\therefore Y = I_2 S_1 \bar{S}_0 = I_2 \text{ when } S_1 S_0 = 10$$

$$4. \quad Y = I_3 \text{ iff } S_1 = S_0 = 1$$

$$\therefore Y = I_3 S_1 S_0 = I_3 \text{ when } S_1 S_0 = 11$$

The final expression for the data output,

$$Y = I_0 \bar{S}_1 \bar{S}_0 + I_1 \bar{S}_1 S_0 + I_2 S_1 \bar{S}_0 + I_3 S_1 S_0$$

Using this expression, the 4-to-1 multiplexer can be implemented using gates as shown in **Figure 2.51**.

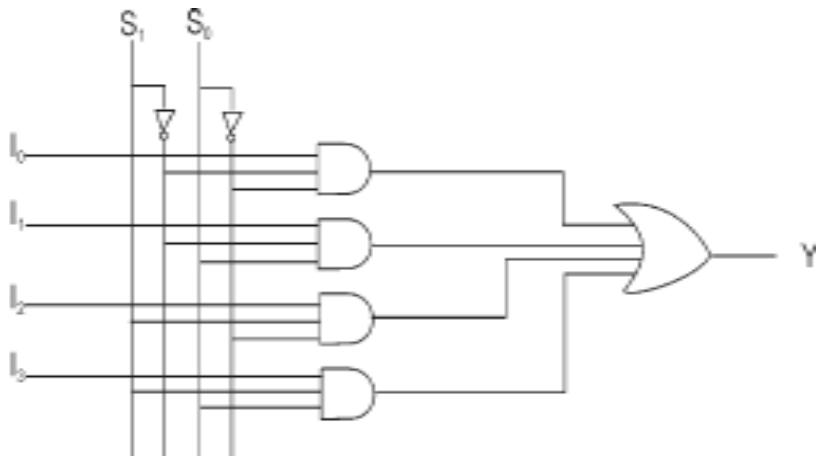


Fig. 2.51: Logic Diagram of 4-to-1 Multiplexer

2.21.2 16-to-1 Multiplexer

The 16-to-1 multiplexer has 16 data input lines ($D_0 - D_{15}$), a single output line (Y) and 4 select lines (A, B, C, D) to select one of the 16 input lines. The truth table for a 16-to-1 multiplexer is shown in **Table 2.16**.

For example, when $ABCD = 0000$, the upper AND gate is enabled while all other AND gates are disabled. Therefore data bit D_0 is transmitted to the output.

$$Y = D_0$$

If $D_0 = 0, Y = 0$

$D_0 = 1, Y = 1$ i.e., Y depends only on the value of D_0

When $ABCD = 1111, Y = D_{15}$.

Thus the control bits (A, B, C, D) determines which of the input data bits is transmitted to the output.

Figure 2.52 shows the logic diagram of 16-to-1 multiplexer.

TABLE 2.16 : Truth Table of 16-to-1 MUX

<i>Enable</i> \bar{E}	<i>Select Inputs</i>				<i>Output</i> Y
	S_3	S_2	S_1	S_0	
0	0	0	0	0	\bar{D}_0
0	0	0	0	1	\bar{D}_1
0	0	0	1	0	\bar{D}_2
0	0	0	1	1	\bar{D}_3
0	0	1	0	0	\bar{D}_4
0	0	1	0	1	\bar{D}_5

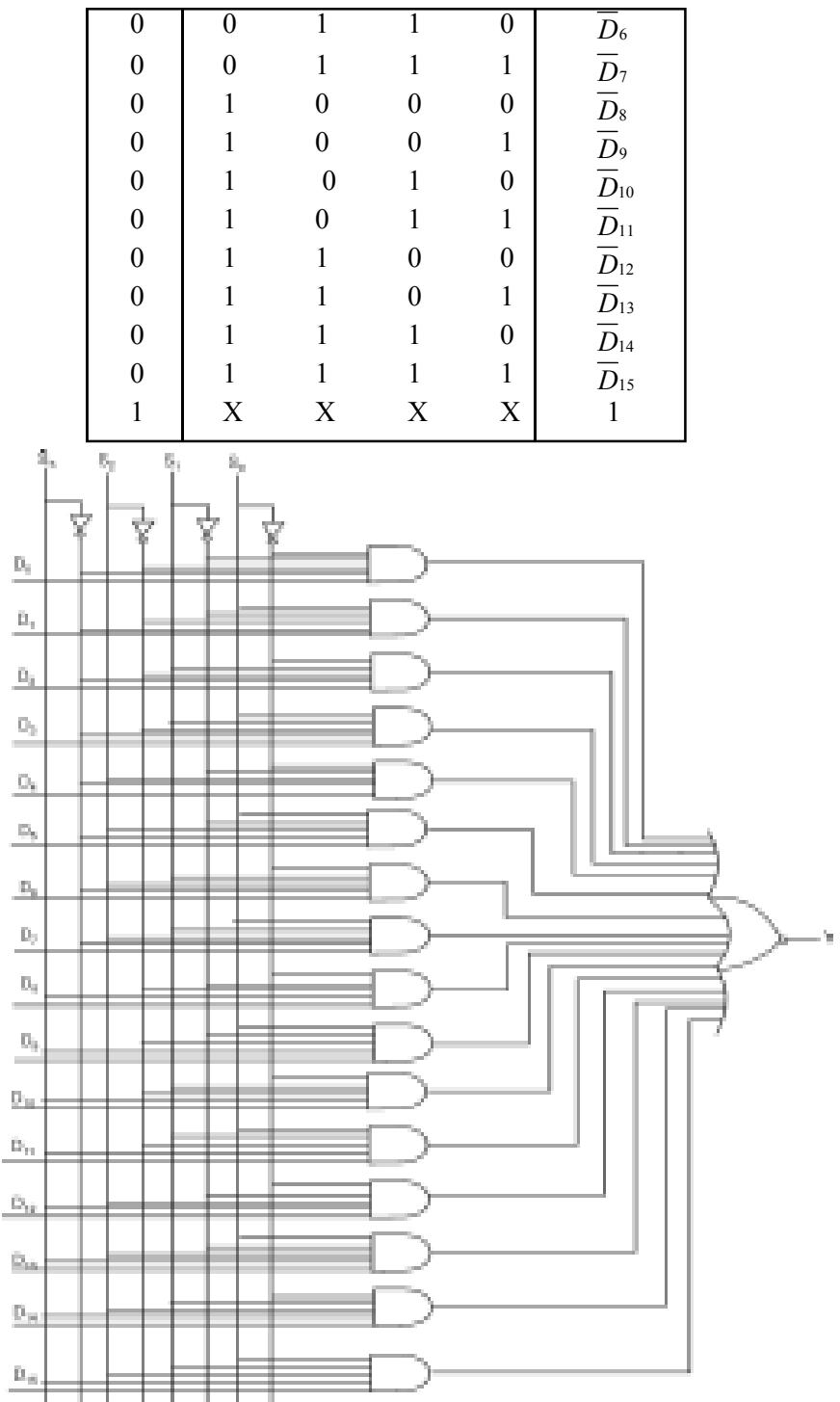


Fig. 2.52: Logic diagram of 16-to-1 multiplexer

We can implement a 16-to-1 multiplexer using two 8-to-1 multiplexer as shown in **Figure 2.53**.

To select one of the 16 inputs, 4 select lines ($S_3S_2S_1S_0$) are required. Among the 4 select lines ($S_2 S_1 S_0$) are connected with 3 select inputs of both 4 to 1 multiplexers. S_3 is connected directly to \bar{E} (Enable) input of MUX1 and it is connected through an NOT gate to \bar{E} input of MUX2. Therefore, when $S_3 = 0$, MUX1 is selected and the inputs (D_0 to D_7) are multiplexed to the output and MUX2 is disabled. When $S_3 = 1$, the MUX 1 is disabled while MUX 2 is enabled and the inputs (D_8 to D_{15}) are multiplexed to the output.

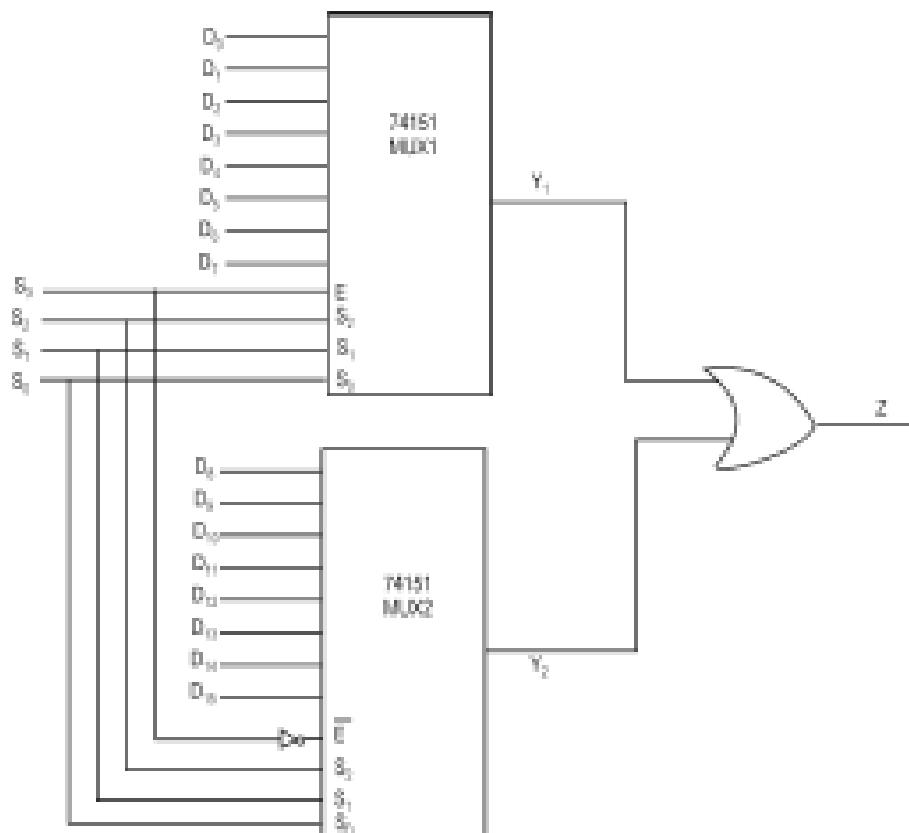


Fig. 2.53: 16-to-1 Multiplexer using IC 74151

2.22 DEMULTIPLEXERS

Demultiplex means “one into many”. A demultiplexer is a combinational logic circuit with one input and many outputs. By applying control signal, we can steer the input signal to one of the output lines. **Figure 2.54** shows the block diagram of demultiplexer. It has 1 input signal, ‘m’ control signals and ‘n’ output signals.

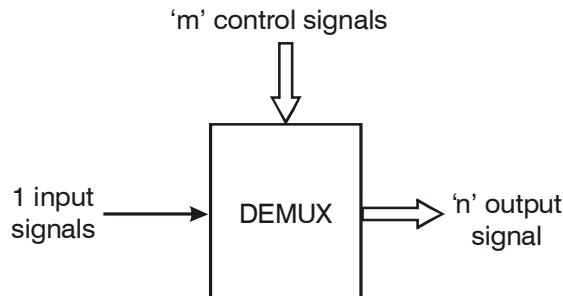


Fig. 2.54: Block diagram of demultiplexer

The select inputs (Control Signals) determine to which output the data input will be connected. As the serial data is changed to parallel data, i.e., the input caused to appear on one of the ' n ' output lines, the demultiplexer is called **data distributor** or **serial-to-parallel converter**.

2.22.1 1-to-4 Demultiplexer

A 1-to-4 demultiplexer has a single input (D), 4 outputs ($Y_0 Y_1 Y_2 Y_3$) and two select lines ($S_1 S_0$). The truth table of the 1 to 4 demultiplexer is shown in **Table 2.17**.

When $S_1 S_0 = 00$, the data input is connected to output Y_0

$$\therefore Y_0 = \bar{S}_1 \bar{S}_0 D = D$$

When $S_1 S_0 = 01$, the data input is connected to output Y_1

$$\therefore Y_1 = \bar{S}_1 S_0 D = D$$

When $S_1 S_0 = 10$, the data input is connected to output Y_2

$$Y_2 = S_1 \bar{S}_0 D = D$$

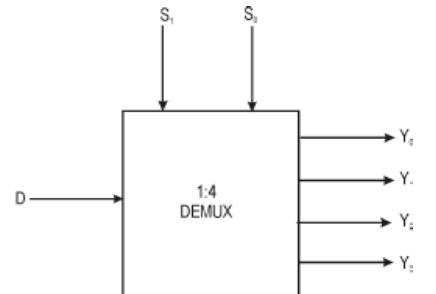
When $S_1 S_0 = 11$, the data input is connected to output Y_3

$$Y_3 = S_1 S_0 D = D$$

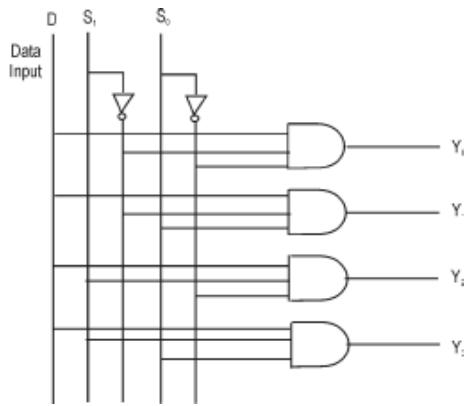
Using these expressions, a 1-to-4 demultiplexer is implemented using AND gates as shown in **Figure 2.55**.

TABLE 2.17 : Truth Table

<i>Data Input</i>	<i>Select Inputs</i>		<i>Outputs</i>			
	S_1	S_0	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0



(a) Logic Symbol



(b) Logic Diagram

Fig. 2.55: 1 to 4 multiplexer

2.23 IMPLEMENTATION OF COMBINATIONAL LOGIC USING MULTIPLEXER

Procedure

1. List the inputs of the multiplexer.
2. List under them all the given minterms in two rows. The first half of the minterms associated with \bar{A} and the second half with the A.
3. The given function is implemented by circling the minterms of the function and applying the following rules to find the values for the inputs of the multiplexer:
 - (i) If the two minterms in a column are not circled, apply 0 to the corresponding multiplexer input.
 - (ii) If the two minterms in a column are circled, apply 1 to the corresponding multiplexer input.
 - (iii) If the bottom minterm is circled and the top is not circled, apply A to the corresponding multiplexer input.
 - (iv) If the top minterm is circled and the bottom is not circled, apply \bar{A} to the corresponding multiplexer input.

4. Draw the multiplexer implementation diagram.

Example 2.4: Implement the following function using a multiplexer.

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

Solution

Variables, $n = 3 (A, B, C)$

Select lines = $n - 1 = 2 (S_1, S_0)$

2^{n-1} to 1 MUX. i.e., 2^2 to 1 \Rightarrow 4 to 1 MUX

Input lines = $2^{n-1} = 2^2 = 4 (I_0, I_1, I_2, I_3)$

Implementation Table

	I_0	I_1	I_2	I_3
\bar{A}	0	1	2	3
A	4	5	6	7
	0	1	A	\bar{A}

$$I_0 = 0, I_1 = 1, I_2 = A, I_3 = \bar{A}$$

Multiplexer Implementation

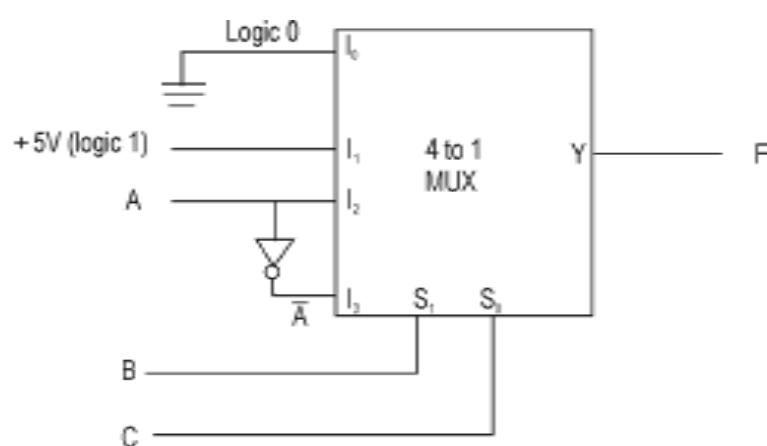


Fig. 2.56: 4 to 1 multiplexer

Example 2.5: Implement the following function with a multiplexer.

$$F(A, B, C, D) = \Sigma (0, 1, 3, 4, 8, 9, 15)$$

Solution: Variables, $n = 4$ (A, B, C, D)

Select lines, $n - 1 = 3$ (S_2, S_1, S_0)

Input lines, $2^{n-1} = 2^3 = 8$ ($I_0 - I_7$)

2^{n-1} to 1 MUX = 2^3 to 1 \Rightarrow 8 to 1 MUX

Use B, C and D variables as selection lines.

Implementation Table

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
\bar{A}	(0)	(1)	2	(3)	(4)	5	6	7
A	(8)	(9)	10	11	12	13	14	(15)
	1	1	0	\bar{A}	\bar{A}	0	0	A

Multiplexer Implementation

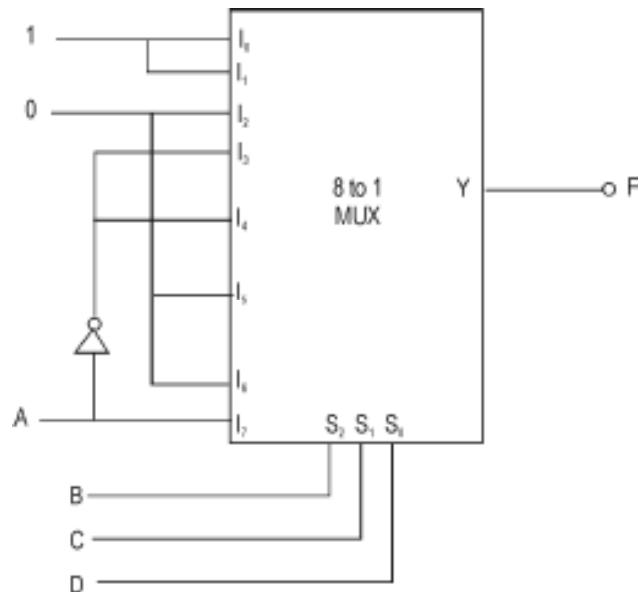


Fig. 2.57: 8 to 1 multiplexer

Example 2.6: Implement the following Boolean function using 8 : 1 MUX.

$$F(A, B, C, D) = \overline{ABD} + ACD + \overline{BCD} + \overline{ACD}$$
(Dec. 2010)

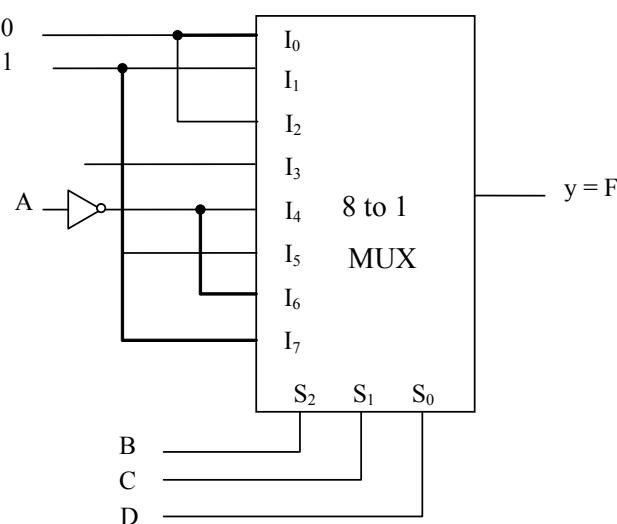
Solution:

$$\begin{aligned}
 F(A, B, C, D) &= \overline{ABD} + ACD + \overline{BCD} + \overline{ACD} \\
 &= \overline{ABD}(\overline{C} + \overline{C}) + ACD(\overline{B} + \overline{B}) + \overline{BCD}(\overline{A} + \overline{A}) + \overline{AD} + \overline{CD} \\
 &= \overline{ABDC} + \overline{AB\overline{DC}} + ACDB + AC\overline{DB} + \overline{BCDA} + \overline{BC\overline{DA}} + \overline{ABD} + \overline{ABD} + \overline{CDA} + \overline{C\overline{DA}} \\
 &= \overline{ABDC} + \overline{AB\overline{DC}} + ABCD + A\overline{BCD} + A\overline{BCD} + \overline{ABC}D + \overline{ABC}D + \overline{ABC}\overline{D} \\
 &\quad + \overline{ABC}\overline{D} + \overline{ABC}D + AB\overline{CD} + A\overline{BCD} + \overline{ABC}D + \overline{ABC}\overline{D} \\
 &= \Sigma[6, 4, 15, 11, 11, 3, 7, 5, 3, 1, 13, 9, 5, 1] \\
 &= \Sigma[1, 3, 4, 5, 6, 7, 9, 11, 13, 15]
 \end{aligned}$$

Implement table: [Use B, C, D as selection live]

	I ₀							
\overline{A}	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	0	1	0	1	\overline{A}	1	\overline{A}	1

MUX Implementation



Example 2.7: Design a combinational logic using a suitable multiplexer to realize the following Boolean expression. $AD'B'C + BC'D'$. (May 2011)

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

$$F(A, B, C, D) = \Sigma(4, 10, 12)$$

Variables n = 4(A, B, C, D)

Select lines $\Rightarrow n - 1 = 4 - 1 = 3$ (S_2, S_1, S_0)

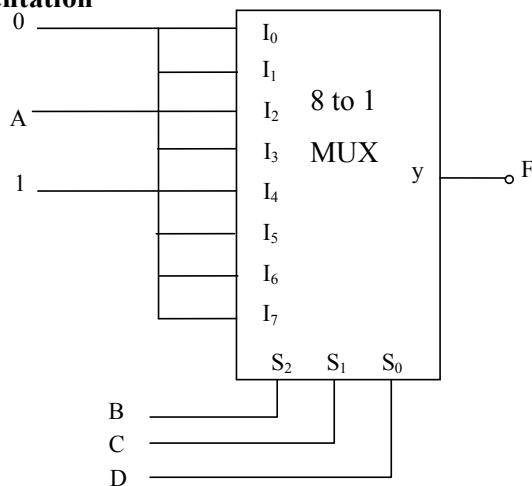
Input lines, $2^{n-1} = 2^3 = 8$ ($I_0 - I_7$)

2^{n-1} to 1MUX = 2^3 to 1 = 8 to 1 MUX

Use B, C, D variables as selection lines.

Implementation Table:

	I_0							
\bar{A}	0	1	2	3	(4)	5	6	7
A	8	9	(10)	11	(12)	13	14	15
	0	0	A	0	1	0	0	0

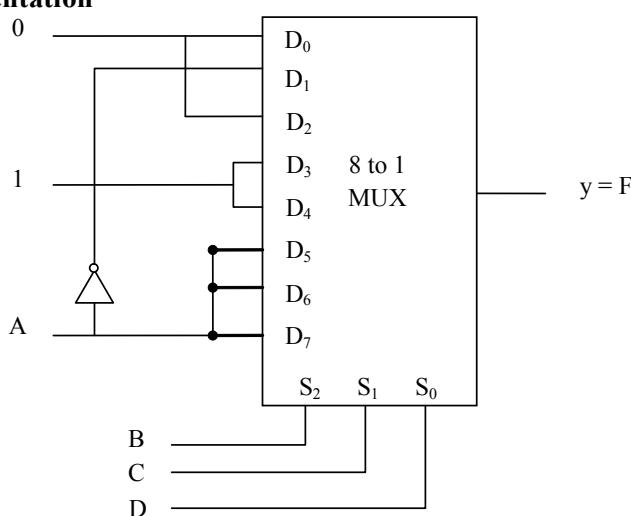
Multiplexer Implementation

Example 2.8: Implement $F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$ using 8×1 multiplexer.

(Dec. 2012)

Solution**Implementation Table:**

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
\bar{A}	0	(1)	2	(3)	(4)	5	6	7
A	8	9	10	(11)	(12)	(13)	(14)	(15)
	0	\bar{A}	0	1	1	D	D	D

Multiplexer Implementation

HARDWARE DESCRIPTION LANGUAGE

2.24 INTRODUCTION TO HARDWARE DESCRIPTION LANGUAGE (HDL)

The complexity of digital designs has increased drastically with the advent of the smaller geometry semi-conductor process technology. This increase has made enormous demands on the industry, giving rise to Hardware Description Languages (HDL) which responded with the HDL-based design process, methodology and design tools. HDL not only manages the increased complexity, but it also allows a shorter design cycle.

HDL is a powerful language with numerous language constructs that are capable of describing very complex behaviour. The characteristics of an ideal HDL are:

- ◆ Supports multiple level of abstraction (gates to systems)
- ◆ Concise
- ◆ Describes functional blocks and their interconnections
- ◆ Well suited for synthesis and verification

2.24.1 Types of HDLS

There are many different systems for modeling and simulating hardware.

- ◆ Verilog
- ◆ VHDL
- ◆ Super log
- ◆ System C
- ◆ L-language and M-language (Mentor)
- ◆ Aida (IBM/HaL)
- ◆ and many others

2.24.2 Major HDLS

Two major HDLS are Verilog and VHDL. These both are programming languages. They are text-based, easier to create a design over schematic entry/capture. VHDL and Verilog both enjoy widespread use and share the logic synthesis market roughly 50/50.

Verilog has its synthetic roots in ‘C’ and is in some respects an easier language to learn and use, while VHDL is more like ADA (U.S. Department of Defense–Sponsored Software programming language) and has more features that support large project development.

2.24.3 Verilog HDL

Verilog HDL is invented by Philip Moorbyin in 1984 at Gate way Design Automation. It enables specification of a digital system at a rang eof levels of abstraction: switches, gates, RTL and higher. It was initially developed in conjunction with the Verilog simulator. The verilog-based synthesis tool is introduced by Synopsys in 1987. In 1989, Gateway Design Automation is bought by Cadence Design Systems. Verilog was placed in public domain as Open Verilog International (OVI), IEEE 1364.

2.24.4 VHDL

VHDL is an international IEEE standard specification language (IEEE 1076-1993) for describing digital hardware used by industry worldwide. VHDL stands for “VHSIC Hardware Description Language”. VHSIC stands for “Very High Speed Integrated Circuit”. VHDL enables hardware modeling from the gat eto system level. It provides a mechanism for digital design and reversable design documentation.

- ❖ In the mid 1980s, the U.S. Department of Defence (DOD) and the IEEE sponsored the development of VHDL.
- ❖ In July 1983, a team of Intermetrics, IBM and Texas Instruments were awarded a contract to develop VHDL.
- ❖ In August 1985, the final version of the language under government contract was released: VHDL Version 7.2.
- ❖ In December 1987, VHDL became IEEE standard 1076-1987 and in 1988 an ANSI standard.
- ❖ In September 1993, VHDL was restandardized to clarify and enhance the language.
- ❖ VHDL has been accepted as a Draft International Standard by the IEC.

2.24.5 Features of VHDL

- ❖ Designs may be decomposed hierarchically.
- ❖ Behavioural specifications can use either an algorithm or an actual hardware structure to define an element's operation.
- ❖ Each design element has both a well-defined interface (for connecting it to other elements) and a precise behavioural specification (for simulating it).
- ❖ Concurrency, timing and clocking can all be modeled.
- ❖ The logical operation and timing behaviour of a design can be simulated.
- ❖ VHDL handles asynchronous and synchronous sequential circuit structures.

Thus VHDL started out as a documentation and modeling language, allowing the behaviour of digital-system designs to be precisely specified and simulated.

2.24.6 Advantages of HDLS

The VHDL and Verilog have the following advantages:

- ❖ They are IEEE standards.
- ❖ They are supported by government and the industry.
- ❖ The HDL texts provide high portability among platforms and design tools.
- ❖ They have the flexibility to model very complex systems down to very primitive circuits.
- ❖ They facilitate design reuse.
- ❖ They allow technology and foundry independence.
- ❖ The documentations can be built-in.
- ❖ Combining with the synthesis, a new design methodology is emerged which reduces design cycle and costs.

2.25 HDL BASED DESIGN FLOW

The several steps in a HDL based design process or design flow is shown in **Figure 2.58**.

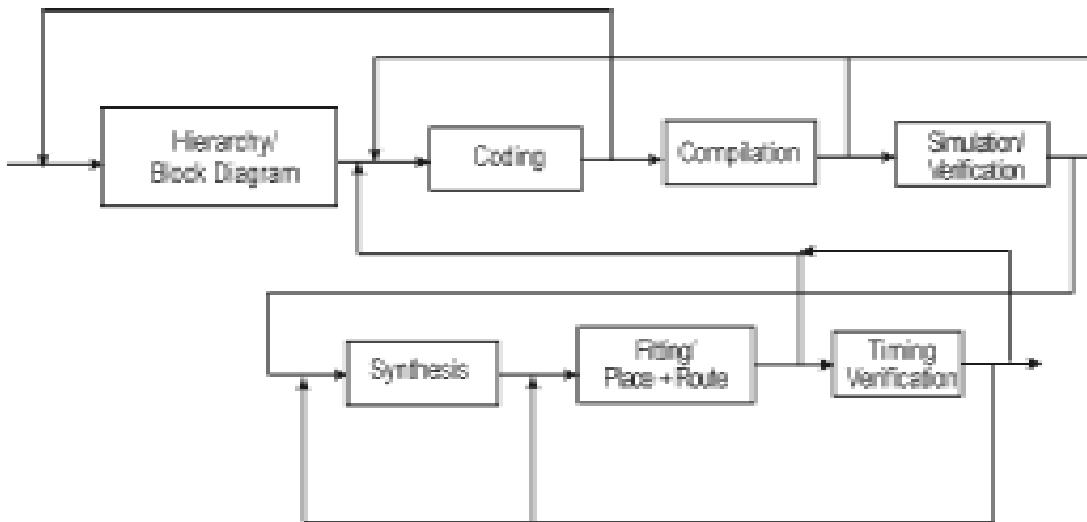


Fig. 2.58: HDL based design flow

2.25.1 Block diagram

A block diagram is an informal pictorial description of the system's major functional modules and their basic interconnections. A block diagram shows the inputs, outputs, functional modules, internal data paths and important control signals of a system. Large logic designs, like software programs are usually hierarchical and VHDL gives a good framework for defining modules and their interfaces.

2.25.2 Coding

The next step is the actual writing of VHDL code for modules, their interfaces and their internal details. Since VHDL is a text based language, VHDL text editor is used for this part of the job. VHDL text editors include features like automatic highlighting of VHDL keywords, automatic indenting, built-in templates for frequently used program structures and built-in syntax checking and one-click access to the compiler.

2.25.3 Compiler

A VHDL compiler analyzes code for syntax errors and also checks it for compatibility with other modules on which it relies. It also creates the internal information that is needed for a simulator to process the design.

2.25.4 Simulator

A VHDL simulator allows to define and apply inputs to the design and to observe its outputs, without even having to build the physical circuit.

2.25.5 Verification

Actually, simulation is just one piece of a larger step called verification. It is satisfying to watch the simulated circuit produce simulated outputs, but the purpose of simulation is larger-it is to verify that the circuit works as desired. The two dimensions to verification are:

- ◆ Functional Verification
- ◆ Timing Verification.

In functional verification, we study the circuit's logical operation independent of timing considerations: gate delays and other timing parameters are considered to be zero.

In timing verification, we study the circuit's operation including estimated delays and we verify that the setup, hold and other timing requirements for sequential devices like flip-flops are met.

2.25.6 Synthesis

Synthesis means converting the VHDL description into a set of primitives or components that can be assembled in the target technology. For example, with programmable logic devices (PLD) the synthesis tool may generate two-level sum-of-products equations. With application-specific ICs(ASIC), it may generate a list of gates and a netlist that specifies how they should be interconnected.

The designer may help the synthesis tool by specifying certain technology-specific constraints, such as maximum number of logic levels or the strength of logic buffers to use.

2.25.7 Fitting

A fitting tool or fitter maps the synthesized primitives or components onto available device resources. For a PLD, this may mean assigning equations to available AND-OR elements. For an ASIC, this may mean laying down individual gates in a pattern and finding ways to connect them within the physical constraints of the ASIC die; this is called the place and routes process.

2.26 VHDL BUILDING BLOCKS

2.26.1 Entity

A VHDL entity is a declaration of a module's inputs and outputs. All designs are expressed in terms of entities. The uppermost level of the design is the top-level entity. If the design is hierarchical, then the top-level description will have lower-level descriptions contained in it. These lower-level descriptions will be lower-level entities contained in the top-level entity description.

A VHDL entity specifies the name of the entity, the ports of the entity and entity-related information. The syntax of a VHDL entity declaration is,

```
entity      entity-name is
port       (signal-names : mode signal-type;
            signal-names : mode signal-type;
            ....
            signal-names : mode signal-type);
end        entity-name;
```

entity-name: A user-selected identifier to name the entity.

signal-names: A comma-separated list of one or more user-selected identifiers to name external-interface signals.

mode: Modes (in, out, inout, buffer) are specifying the signal direction.

signal type: A built-in or user-defined signal type (bit, integer, real, etc.)

An entity example is,

```
entity mux is
port      (a , b , c , d : in bit ;
            S0 , S1 : in bit ;
            x , : out bit );
end        mux ;
```

The entity describes the interface to the outside world. It specifies the number of parts, the direction of the ports and the types of the ports.

The name of the entity is *mux*. The entity has 7 ports in the port clause. 6 ports are of mode ‘in’ and one port is of mode ‘out’. The four data input ports (*a*, *b*, *c*, *d*) are of type ‘bit’. The two multiplexer select inputs. (S_0 , S_1) are also of type ‘bit’. The output port is of type ‘bit’.

2.26.2 Architecture

All entities that can be simulated have an architecture description. The architecture describes the behaviour of the entity. A single entity can have multiple architectures. One architecture might be behavioural while another might be a structural description of the design. The syntax of a VHDL architecture definition is,

architecture architecture-name of entity-name is

type declarations
signal declarations
constant declarations
function definitions
procedures definitions

begin

concurrent-statement

....

concurrent-statement

end architecture-name ;

An architecture for the counter device is,

architecture dataflow of mux is

signal select : integer;

begin

select <= 0 WHEN $S_0 = 0$ AND $S_1 = 0$ ELSE
1 WHEN $S_0 = 1$ AND $S_1 = 0$ ELSE
2 WHEN $S_0 = 0$ AND $S_1 = 1$ ELSE
3 ;

condition

x <= a AFTER 0.5 NS WHEN select = 0 ELSE
b AFTER 0.5 NS WHEN select = 1 ELSE
c AFTER 0.5 NS WHEN select = 2 ELSE
d AFTER 0.5 NS ;

end dataflow;

In this example,

architecture name = *dataflow*

entity name = *mux*

The architecture ‘*dataflow*’ describes the underlying functionality of the entity ‘*mux*’ and contains the statements that model the behaviour of the entity.

2.26.3 Configuration

A configuration statement is used to bind a component instance to an entity- architecture pair. A configuration can be considered like a parts list for a design. It describes which behaviour to be used for each entity, much like a parts list that describes which part to use for each part in the design. The syntax of configuration declaration is,

```
Configuration configuration_name of entity_name is
  for architecture_name
    for instance_name : entity_name use entity
      library_name . entity_name
      (architecture_name) ;
    end for;
    for instance_name : entity_name use configuration
      library_name . configuration_name ;
    end for;
  end for;
end configuration_name ;
```

2.26.4 Package

A VHDL package is a file containing definitions of objects that can be used in other programs. The kind of objects that can be put into a package include signal, type, constant, function, procedure and component declarations.

The primary purpose of a package is to encapsulate elements that can be shared among two or more design units. A package is a common storage area used to hold data to be shared among a number of entities. Declaring data inside of a package allows the data to be referenced by other entities; thus the data can be shared.

A package consists of two parts:

- ◆ package declaration
- ◆ package body.

(a) *Package Declaration*

It defines the interface for the package, much the same way that the entity defines the interface for a model. The package declaration section can contain the following declarations:

- ◆ Subprogram declaration
- ◆ Type, subtype declaration
- ◆ File declaration
- ◆ Alias declaration
- ◆ Constant declaration
- ◆ Deferred constant declaration
- ◆ Component declaration
- ◆ Attribute declaration
- ◆ Signal declaration
- ◆ Use clause declaration

(b) *Package Body*

The main purpose of the package body is to define the values for deferred constants and specify the subprogram bodies for any subprogram declarations from the package declaration. The package body contain the following declarations:

- ◆ Subprogram declaration
- ◆ Type, subtype declaration
- ◆ Constant declaration
- ◆ File declaration
- ◆ Alias declaration
- ◆ Subprogram body
- ◆ Use clause

Syntax of a VHDL Package Declaration

Package *package-name* is

 — *declare some stuff*

end package;

Syntax of a VHDL Package Body:

```
package body package_name is
    — put subprogram bodies here
end package_name;
```

2.26.5 Driver

It is a source on a singal. If a signal is driven by two sources, then when both sources are active, the signal will have two drivers. VHDL has a unique way of handling multiply driven signals. Multiply driven signals are very useful for modeling a data bus, a bi-directional bus, and so on. A multiply driven signal has many drivers. The values of all of the drivers are resolved together to create a single value for the signal.

Drivers are created by signal assignment statements.

Consider the following architecture:

```
architecture test of test is
```

```
begin
```

```
    a <= b after 20 ns;
```

```
    a <= c after 30 ns;
```

```
end test;
```

Signal ‘a’ is being driven from two sources, ‘b’ and ‘c’. Each concurrent signal assignment statement creates a driver for signal ‘a’. The first stagement creates a driver that contains the value of signal ‘b’ delayed by 20 nanoseconds. The second statement creates a driver that contains the value of signal ‘C’ delayed by 30 nanoseconds.

2.26.6 Attribute

An attribute is data that are attached to VHDL objects or predefined data about VHDL objects. Predefined attributes are the data that can be obtained from blocks, signals and types or subtypes. The data obtained falls into one of the following categories:

- ◆ Value kind – A simple value is returned.
- ◆ Function kind – A function call is performed to return a value.
- ◆ Signal kind – A new signal is created whose value is derived from another signal.
- ◆ Type kind – A type mark is returned.
- ◆ Range kind – A range value is returned.

VHDL user defined attributes are a mechanism for attaching data to VHDL objects. The data

attached can be used during simulation. Data such as the disk file name of the model, loading information, driving capability, resistance, capacitance, physical location and So on can be attached to objects. The user-defined attributes can be assigned to the following list of objects:

- ◆ Entity
- ◆ Architecture
- ◆ Configuration
- ◆ Procedure
- ◆ Function
- ◆ Package
- ◆ Type and subtype
- ◆ Constant
- ◆ Signal
- ◆ Variable
- ◆ Component
- ◆ Label

Table 2.18 lists predefined attributes with examples.

TABLE 2.18 : Predefined Attributes

Attribute	Explanation	Examples
T'BASE	Returns the base type of datatype it is attached to	NATURAL'BASE returns INTEGER
T'LEFT	Returns left value specified in type declaration	INTEGER'LEFT is -2147483647 BIT'LEFT is '0'
T'RIGHT	Returns right value specified in type declaration	INTEGER'RIGHT is 2147483647 BIT'RIGHT is '1'
T'HIGH	Returns largest value specified in declaration	TYPE bits is 255 downto 0 bits'HIGH is 255
T'LOW	Returns smallest value specified in declaration	TYPE bits is 255 downto 0 bits'LOW is 0
T'POS(X)	Returns position number of argument in type (first) position is 0)	TYPE color IS (red, green, blue, orange); color'POS (green) is 1
T'VAL(X)	Returns value in type at specified position number	TYPE color IS (red, green, blue, orange); color'VAL (2) is blue
T'SUCC(X)	Returns the successor to the value passed in	TYPE color is (red, green, blue, orange) color'SUCC (green) is blue
T'PRED (X)	Returns the predecessor to the value passed in	TYPE color IS (red, green, blue, orange); color' PRED (blue) is green

<i>Attribute</i>	<i>Explanation</i>	<i>Examples</i>
T'LEFTTOP(X)	Returns the value to the left of the value passed in	TYPE color IS (red, green, blue, orange); color'LEFTTOP (green) is red
T'RIGHTTOP(X)	Returns the value to the right of the value passed in	TYPE color IS (red, green, blue, orange); color'RIGHTTOP (blue) is orange
A'LEFT (N)	Returns left array bound of selected index range	a_type'LEFT(1) is a 0 a_type'LEFT (2) is 7
A'RIGHT(N)	Returns right array bound of selected index range	a_type'RIGHT (1) is 3 a_type'RIGHT (2) is 0
A'RIGHT (N)	Returns largest array bound value of selected index range	a_type'HIGH (1) is 3 a_type'HIGH (2) is 7
A'LOW (N)	Returns smallest array bound value of selected index range	a_type LOW(1) is 0 a_type'LOW(2) is 0
A'RANGE (N)	Returns selected index	a_type'RANGE (1) is 0 TO 3 a_type'RANGE (2) is 7 DOWNTO 0
A'REVERSE_ RANGE (N)	Returns selected index range reversed	a_type'REVERSE_Range (1) is 3 DOWNTO 0 a_type'REVERSE_RANGE (2) is 0 TO 7
A'LENGTH (N)	Returns size of selected index range	a_type'LENGTH (1) is 4 a_type'LENGTH (2) is 8
S'DELAYED (T)	Creates a new signal delayed by T	clock'DELAYED (10 ns)
S'QUIET (T)	Creates a new signal that is true when signal S has had no transactions for time T; otherwise, false	reset'QUIET (5 ns)

<i>Attribute</i>	<i>Explanation</i>	<i>Examples</i>
S'STABLE (T)	Creates a new signal that is true when signal S has no events for time T; otherwise, false	clock'STABLE (1 ns)
S'TRANSACTION	Creates a signal of type BIT that toggles for every transaction on signal S	load TRANSACTION
S'EVENT	Returns true when an event has occurred for signal S is delta	clock'EVENT
S'ACTIVE	Returns true when a transaction has occurred for signal S is delta	load'ACTIVE
S'LAST_EVENT	Returns the elapsed time since the last event on signal S	data'LAST_EVENT
S'LAST_ACTIVE	Returns the elapsed time since the last transaction on signal S	clock ' Last_Active
S'LAST_VALUE	Returns the previously assigned value of signal S	data'LAST_VALUE

2.26.7 Generic

A generic is a general mechanism that passes information to an entity. For instance, if an entity is a gate level model with a rise and a fall delay, values for the rise and fall delays could be passed into the entity with generics. The syntax of a VHDL generic declaration within an entity declaration is,

```

entity entity-name is
  generic    (constant-names : constant-type ;
              ....
              constant-names : constant-type);
  Port      (signal-names : mode signal-type ;
              ....
              signal-names : mode signal-type);
end entity-name;

```

The following is an example of an entity for an AND gate that has 3 generics associated with it entity *and* is

```
generic (rise, fall : time; load : integer);
port   (a, b : in bit;
        c : out bit);
end and
```

This entity allows to pass in values for the rise and fall delays, as well as the loading device that has on its output.

2.26.8 Process

A process is a collection of “sequential” statements that executes in parallel with other concurrent statements and other processes. A VHDL process statement can be used anywhere that a concurrent statement can be used. The syntax of a VHDL process statement is given below:

Process (*signal-names* , *signal-name* , , *signal-name*)

- type declarations*
- variable declarations*
- constant declarations*
- function definitions*
- procedure definitions*

begin

- sequential-statement*

.....

- sequential-statement*

end process;

A process statement has a declaration section and a statement part. In the declaration section, types, variables, constants, subprograms and so on can be cleared. The statement part contains only sequential statements. Sequential statements consist of CASE statements, IF THEN ELSE statements, LOOP statements and so on.

2.26.9 Bus

In VHDL, a bus is a special kind of signal that may have its drivers turned off.

2.27 LIBRARY

A VHDL ‘library’ is a place where the VHDL compiler stores information about a particular design project, including intermediate files that are used in the analysis, simulation and synthesis of the design. The designer can specify the name of a library using a ‘library clause’ at the beginning of the design file. For example, the IEEE library is specified as,

Library ieee ;

A design can use a package by including a ‘use clause’ at the beginning of the design file. For example, to use all of the definitions in the IEEE standard 1164 package, we can specify as

use ieee . std_logic_1164 . all ;

where, ‘ieee’ is the name of a library

‘std_logic_1164’ is the file name, contains the desired definitions ‘all’ tells the compiler to use all of the definitions in this file.

Instead of ‘all’, we can write the name of a particular object to use just its definition, for example,

use ieee . std_logic_1164 . std_ulogic

2.28 TYPES AND CONSTANTS

All signals, variables and constants in a VHDL program must have an associated “type”. The type specifies the set or range of values that the object can take on and there is also typically a set of operators (add, AND, etc.) associated with a given type.

- ❖ SIGNAL, which represents interconnection wires that connect component instantiation ports together.
- ❖ VARIABLE, which is used for local storage of temporary data, visible only inside a process.
- ❖ CONSTANT, which names specific values.
- ♦ Signal *signal-name* : *signal-type* [: = *initial-value*];

Example:

Signal *VCC* : std_logic := ‘1’ ;

Signal *ground* : std_logic := ‘0’ ;

- ♦ Variable *variable-name* : *variable-type* ;

Example:

Variable *state* : std_logic ;

Variable *delay* : time ;

- Constant `constant_name : type_name := value ;`

Example:

Constant PI : `real` := 3.1414 ;

Constant Bus_SIZE : `integer` := 32 ;

2.29 PREDEFINED TYPES

VHDL has a few predefined types listed in **Table 2.19**. Type ‘`integer`’ is defined as the range of integers ($-2^{31} + 1$ through $+2^{31} + 1$)

Type ‘`boolean`’ has two values – true and false

Built-in operators for the integer and boolean types are listed in **Table 2.20**.

TABLE 2.19 : Predefined Types

bit	real
bit_vector	severity_real
boolean	string
character	time
integer	

TABLE 2.20: Predefined Operators

<i>integer</i>	<i>operators</i>	<i>boolean operators</i>	
+	addition	and	AND
-	subtraction	or	OR
*	multiplication	nand	NAND
/	division	nor	NOR
mod	modulo division	xor	Exclusive OR
rem	modulo remainder	xnor	Exclusive NOR
abs	absolute value	not	Complement
* *	exponentiation		

2.30 USER-DEFINED TYPES

(i) type:

type *type-name* is (*value-list*) ;

Example:

type *color* is (*red* , *yellow* , *blue*) ;

type *traffic_light_state* is (*reset* , *stop* , *wait* , *go*) ;

(ii) subtype:

subtype declarations are used to define subsets of a type.

subtype *subtype-name* is *type-name* start to *end* ;

Example:

subtype *fourval_logic* is *std_logic* range 'X' to 'Z'

subtype *eightval* is *integer* range 0 to 7 ;

(iii) std-logic type

The IEEE 1164 STD_LOGIC type is actually defined as a subtype of an unresolved type, STD_ULOGIC. In VHDL, an unresolved type is used for any signal that may be driven in two or more processes. The definition of VHDL STD_LOGIC type is given as,

```
type STD_ULOGIC is ('U' — Uninitialized
                     'X' — Forcing unknown
                     'O' — Forcing 0
                     'I' — Forcing 1
                     'Z' — High Impedance
                     'W' — Weak unknown
                     'L' — Weak 0
                     'H' — Weak 1
                     '-' — Don't care
);

```

subtype STD_LOGIC is resolved STD_ULOGIC;

(iv) Array type

An array is an ordered set of elements of the same type, where each element is selected by an array.

Syntax

```
type type_name is array (start to end) of element_type ;
type type_name is array (start downto end) of element_type ;
```

Examples

```
type monthly_count is array (1 to 12) of integer;
type byte is array (7 downto 0) of STD_LOGIC;
```

2.31 SEQUENTIAL STATEMENTS

The sequential statements exist inside the boundaries of a process statement as well as in subprograms. The sequential statements are:

```
IF
CASE
LOOP
EXIT
ASSERT
WAIT
```

(i) IF Statement

The IF statement starts with the keyword ‘IF’ and ends with the keywords ‘END IF’. There are also two optional clauses: ELSIF and ELSE. The ESLIF (if-then- else) clause is repeatable-more than one ELSIF clause is allowed; but the ELSE clause is optional and only one is allowed. The syntax of IF statement is,

```
if statement :: =
    if condition then
        sequence of statements
    [ elsif condition then
        sequence of statements
    [else
        sequence of statements]
    end if;
```

Example

1. If ($X < 10$) then

```
a := b;
```

```
end if;
```

2. If (day = Sunday) then

```
weekend := TRUE;
```

```
elsif (day = Saturday) then
```

```
weekend := TRUE;
```

```
else
```

```
weekday := TRUE;
```

```
end if;
```

(ii) CASE Statement

The CASE statement is used whenever a single expression value can be used to select between a number of actions. The syntax of a VHDL CASE statement is,

Case expression is

```
when choices => sequential-statements
```

```
...
```

```
when choices => sequential-statements
```

```
end case;
```

This statement evaluates the given expression, finds a matching value in one of the choices, and executes the corresponding sequential-statements. One or more sequential statements can be written for each set of choices. The choices may take the form of a single value or of multiple values separated by vertical bars (|). Prime-number-detector architecture using CASE statement is shown below:

```
architecture prime_8_arch of prime is
begin
    process(N)
        begin
            case CONV_INTEGER (N) is
                when 1 => F <= '1';
                when 2 => F <= '1';
                when 3 | 5 | 7 | 11 | 13 => F <= '1';
                when others => F <= '0';
            end case;
        end process;
    end prime_8_arch;
```

(iii) Loop Statement

The Loop Statement is used whenever an operation needs to be repeated. Loop statements are used when powerful iteration capability is needed to implement a model. Following is the syntax of Loop statement:

```
loop
    sequential-statement
    ....
    sequential-statement
end loop;
```

Example

```
for I in 1 to 10 loop
    I_squared (I): <$E=> I * I;
end loop;
```

(iv) EXIT Statement

The EXIT statement allows to exit or jump out of a Loop Statement currently in execution. This causes execution to halt at the location of the EXIT Statement. Execution continues at the statement following the Loop Statement.

```
exit;
exit when a <= b ;
exit loop_label when X = Z;
```

(v) ASSERT Statement

The ASSERT statement checks the value of a boolean expression for true or false. If the value is true, the statement does nothing. If the value is false, the ASSERT statement outputs a user-specified text string to the standard output to the terminal

```
assert_statement ::= 
    assert condition
    [REPORT expression]
    [SEVERITY expression];
```

(vi) WAIT Statement

WAIT statement is used to suspend a process for a specified time period. It is used for specifying clock inputs to synthesis tools. The options available to the WAIT Statement are:

- ◆ WAIT ON signal changes
- ◆ WAIT UNTIL an expression is true
- ◆ WAIT FOR a specific amount of time

```
process
begin
     $XT <= '0' ; YT <= '0' ;$ 
    wait for 10 ns;
end process;
```

2.32 CHALLENGES in HDL

HDL is new and different from other traditional programming languages. Traditional programming languages execute sequentially. In general, they are written so that a conclusion may be reached. HDL models hardware which is concurrent in nature. It provides timing concepts and hardware implications. Because of this, a person with both software and hardware background can learn HDL much easier than a person who only knows either software or hardware.

A person who have more software than hardware experience find their challenges in the following areas:

- ❖ Process concurrency.
- ❖ Sequential statements regions and Concurrent statements regions.
- ❖ Hardware implication with schematic and simulation waveform.
- ❖ Timing concepts.
- ❖ Mixing other program languages with VHDL.

A person who have more hardware than software experience find their challenges in the following areas:

- ❖ VHDL syntax.
- ❖ Writing VHDL to imply hardware.
- ❖ Sequential statements regions and concurrent statements regions.
- ❖ Taking advantages of VHDL constructs.
- ❖ Software concepts such as subprograms, packages, libraries and configurations.

2.33 HDL FOR COMBINATIONAL CIRCUITS

2.33.1 3 input OR gate

The HDL program for 3 input OR gate is given below. The 3 inputs of the gate are X, Y, Z and one output D. This program uses a simple concurrent assignment statement to describe the functionality of the OR gate.

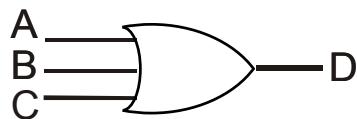


Fig. 2.59: 3 input OR gate

```
library IEEE;
use IEEE . Std_logic_1164 . all;
entity or_3_1S
    port (X, Y, Z : in STD_LOGIC;
          D : out STD_LOGIC);
end or_3;
architecture Synth of or_3 is
begin
    D <= X or Y or Z;
end Synth;
```

2.33.2 3 INPUT XOR GATE



Fig. 2.60: 3 input XOR

3 input exclusive OR gate is shown in **Figure 2.60**.

The VHDL program for a 3-input XOR gate is,

```
library IEEE;
use IEEE . Std_logic_1664 . all;
entity xor1 is
port (
A, B, C : in STD_LOGIC;
Y : out STD_LOGIC
);
end xor1;

architecture xor1 of xor1 is
begin
Y <= A xor B xor C;
end xor1;
```

2.33.3 2-input XNOR gate

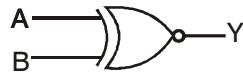


Fig. 2.61: 2-input XNOR

The VHDL program for 2-input XNOR gate using the built-in ‘xnor’ operator is given below. The ‘xnor’ operator can be overloaded to work with any types. In this example two STD_LOGIC type values are XNOR’ed together to form the final result.

```
library IEEE;
use IEEE . STD_logic_1164 . all;
entity xnor2 is
generic (delay : time);
port (A, B : in STD_LOGIC;
Y : out STD_LOGIC);
end entity xnor2;
architecture better of xnor2 is
begin
Y <= A xnor B after delay;
end architecture better;
```

2.33.4 Decoder

There are several ways to approach the design of decoders in VHDL. The most primitive approach would be two write a structural equivalent of a decoder logic circuit. **Figure 2.62** shows the inputs and outputs of 2-to-4 decoder and **Figure 2.63** shows the gate-level circuit. The input code word I_0, I_1 represents an integer in the range $0 - 3 [00, 01, 10, 11]$. The output code word Y_3, Y_2, Y_1, Y_0 has Y_i equal to 1 if and only if the input code word is the binary representation of ‘ i ’ and the enable input EN is 1. If EN = 0, all of the outputs are 0. The VHDL structural program for 2-to-4 decoder is given below. The components ‘and’ and ‘inv’ are assumed to already exist in the target technology.

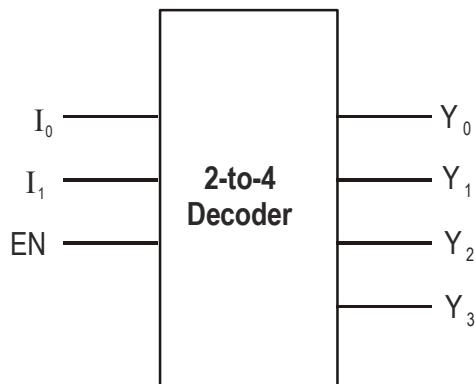


Fig. 2.62: 2 to 4 decoder

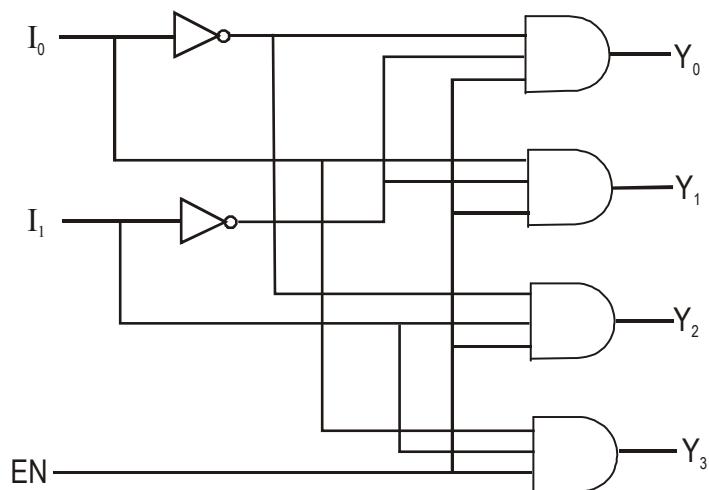


Fig. 2.63: Decoder gate circuit

```

library IEEE;
use IEEE . std_logic_1164 . all;

entity V2to4dec is
    port (I0, I1, EN : in STD_LOGIC;
          Y0, Y1, Y2, Y3 : out STD_LOGIC);
end V2to4dec;

architecture V2to4dec_S of V2to4dec is
begin
    signal NOTI0, NOTI1 : STD_LOGIC;
    component inv port (I: in STD_LOGIC; 0 : out STD_LOGIC); endcomponent;
    component and port (I0, I1, I2 : in STD_LOGIC; 0 : out STD_LOGIC); end component;
    begin
        u1 : inv port map (I0, NOTI0);
        u2 : inv port map (I1, NOTI1);
        u3 : and port map (NOTI0, NOTI1, EN, Y0);
        u4 : and port map (I0, NOTI1, EN, Y1);
        u5 : and port map (NOTI0, I1, EN, Y2);
        u6 : and port map (I0, I1, EN, Y3);
    end V2to4dec_S;

```

2.33.5 MULTIPLEXER

The logic symbol of 4-to-1 multiplexer is shown in **Figure 2.64** and the logic diagram is shown in **Figure 2.65**. The input lines are I_0, I_1, I_2 and I_3 . The select lines are S_0 and S_1 . The one output line is Y . The truth table of 4-to-1 multiplexer is given in **Table 3.10**. The boolean function of 4-to-1 multiplexer is,

$$Y = I_0 \overline{S_1} \overline{S_0} + I_1 \overline{S_1} S_0 + I_2 S_1 \overline{S_0} + I_3 S_1 S_0$$

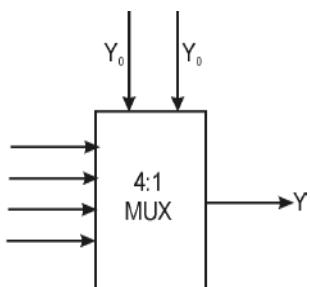


Fig. 2.64: Logic Symbol

Table: 3.10 Truth Table

Select Lines		
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

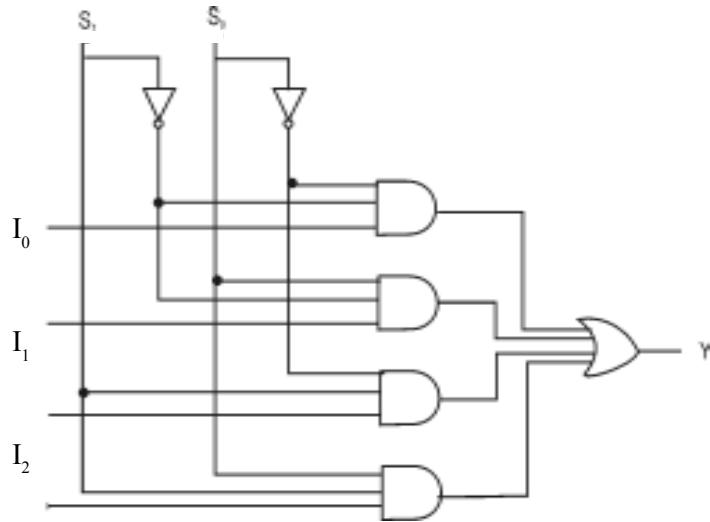


Fig. 3.65: Logic diagram of 4-to-1 Multiplexer

The VHDL program for 4-to-1 multiplexer (8 bit) is given below:

```

Library IEEE;
use IEEE . Std_logic_1164 . all;
entity mux is
port (
    S : in STD_LOGIC_VECTOR (1 downto 0); — Select inputs
    I0, I1, I2, I3 : in STD_LOGIC_VECTOR (1 to 8); — Data bus
    inputs
    Y : out STD_LOGIC_VECTOR (1 to 8) — Data bus output
);
end mux;

architecture better of mux is
begin
    with S select Y<=
        I0 when "00",
        I1 when "01",
        I2 when "10",
        I3 when "11",
        (others → 'u') when others; — 8 bit vector of 'u'
end better;
```

2.33.6 Parity Checker

We have already discussed that error-detecting codes use an extra bit, called a parity bit, to detect errors in the transmission and storage of data. In an even parity code, the parity bit is chosen so that the total number of 1 bit in a code word is even. The 74280 is a 9-bit parity generator/checker. Both even and odd parity enable inputs and parity outputs are available for generating or checking parity on 8 bits. **Figure 2.66** shows the logic diagram for 9 bit parity generator/checker (74280).

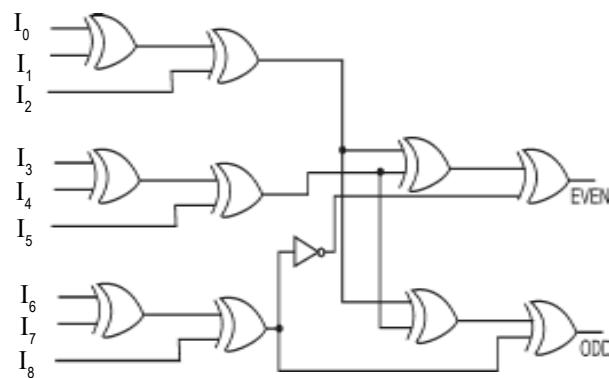


Fig. 2.66: 9-bit Parity Generator/Checker

The behavioural VHDL program for a 9-input parity checker is given below:

```

library IEEE;
use IEEE . Std_logic_1164 . all;
entity parity is
    port (I : in STD_LOGIC_VECTOR (1 to 9);
          EVEN, ODD : out STD_LOGIC );
architecture parity 1 of parity is
begin
process (I)
variable p : STD_LOGIC;
begin
    p := I(1)
    for j in 2 to 9 loop
        if I(j) = '1' then p := not p ; end if;
        end loop;
        ODD <= p ;
        EVEN <= not p ;
    end process;
end parity 1;

```

2.33.7 Comparator

The 8 bit comparator receives two 8bit numbers A and B as inputs and the outputs are $A = B$ and $A > B$. The logic symbol of 8-bit comparator (74682) is shown in **Figure 2.67**. The top half of the circuit checks the two 8 bit numbers for equality. The bottom half of the circuit compares the two input 8 bit numbers and asserts $A > B$ if $[A_7 - A_0] > [B_7 - B_0]$. The 74682 does not provide less than ($A < B$) output. However, any desired condition, including \leq and \geq can be obtained as shown in **Figure 2.68**.

VHDL has comparison operators for all of its built-in types. Equality ($=$) and inequality ($/=$) operators apply to equal size and structure and the operands are compared component by component. The other comparison operators ($>$, $<=$, $<=$) apply only to integer types, enumerated types (such as STD_LOGIC) and one dimensional arrays of integer type. The VHDL program that produces all of the comparison outputs for comparing two 8bit unsigned integers is given.

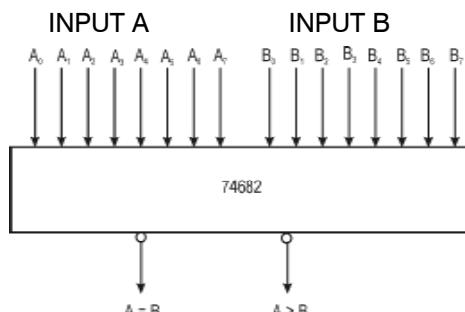


Fig. 2.67: 8 bit comparator

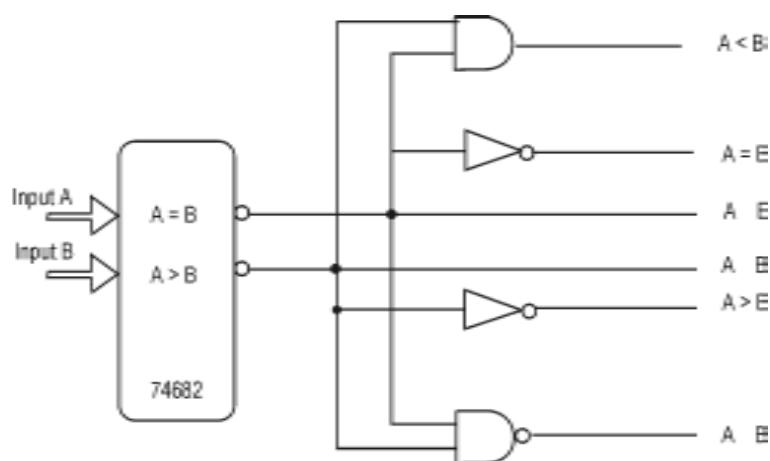


Fig. 2.68: 8-bit comparator's other outputs

```

library IEEE;
use IEEE . Std_logic_1164 . all
entity compare is
port (
    A, B : in STD_LOGIC_VECTOR (7 downto 0);
    EQ, NE, GT, GE, LT, LE : out STD_LOGIC
);
end compare;

architecture compare 1 of compare is
begin
process (A, B)
begin
    EQ <= '0' ; NE <= '0' ; GT <= '0' ;
    GE <= '0' ; LT <= '0' ; LE <= '0' ;
    if A = B then EQ <= '1' ; end if;
    if A/ = B then NE <= '1' ; end if;
    if A > B then NE <= '1'; end if;
    if A > = B then GE <= '1'; end if;
    if A < B then LT <= '1'; end if;
    if A < = B then LE <= '1'; end if;
end process
end compare 1;

```

2.33.8 Binary Adder-Subtractor

The addition and subtraction operations combined into one circuit is called as binary adder-subtractor. This is done by including an XOR gates with each full adder. 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 XOR gate receives input M and one of the inputs of $B(B_0 - B_3)$. When $M = 0$, $B \oplus 0 = B$. The full adders receive the value of B , the input carry is 0 and the circuit performs addition operation $(A + B)$, when $M = 1$, $B \oplus 1 = \bar{B}$ and $C_{in} 0 = 1$. The B inputs are all complemented

and 1 is added through the input carry. The circuit performs the operation $A + (2^{\text{'}}\text{'s complement of } B)$ i.e., $A - B$. The logic diagram of 4 bit binary adder-subtractor is shown in **Figure 2.69** and the logic symbol is shown in **Figure 2.70**.

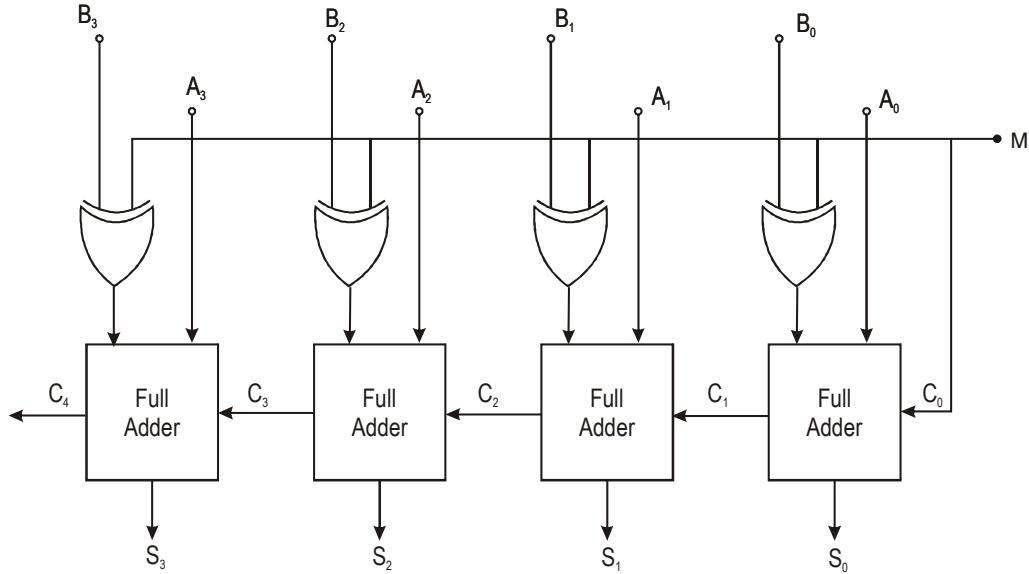


Fig. 2.69: 4 bit adder/subtractor

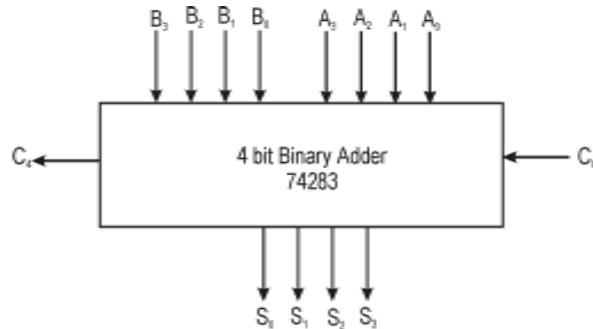


Fig. 2.70: 74283 logic symbol

The IEEE_std_logi_arith package defines two new array types, SIGNED and UNSIGNED, and a set of comparison functions for operands of type INTEGER, SIGNED or UNSIGNED. The package also defines addition and subtraction operations for the same kind of operands as well as STD_LOGIC and STD_ULOGIC for 1-bit operands.

The VHDL program for 8 bit adder-subtractor is given here.

$$S = 9 \text{ bit result of addition of } A \& B$$

$T = 9$ bit result of $A + C$

$U = 8$ bit result of $C + \text{SIGNED}(D)$

$V = 9$ bit result of $C - \text{UNSIGNED}(D)$

```
Library IEEE;
use IEEE . Std_logic_1164 . all;
use IEEE . Std_logic_orith . all;
entity addsub is
    port (
        A, B : in UNSIGNED (7 downto 0);
        C : in SIGNED (7 downto 0);
        D : in STD_LOGIC_VECTOR (7 downto 0);
        S : out UNSIGNED (8 downto 0);
        T : out SIGNED (8 downto 0);
        U : out SIGNED (7 downto 0);
        V : out STD_LOGIC_VECTOR (8 downto 0)
    );
end addsub;

architecture adds of addsub is
begin
    S <= ('0' & A) + ('0' & B);
    T <= A + C;
    U <= C + SIGNED (D);
    V <= C - UNSIGNED (D);
end adds;
```

ANNA UNIVERSITY QUESTIONS

COMBINATIONAL LOGIC

PART-A

1. What are half and full adders? (April 2003)
2. State the condition for $B = I_2$ in the Boolean expression

$$B = I_0 \bar{S}_0 \bar{S}_1 + I_1 \bar{S}_0 S_1 + I_2 S_0 \bar{S}_1 + I_3 S_0 S_1$$
 (April 2003)

What is the combinational logic circuit realised by the above boolean expression.
3. State the condition to check the equality of two n-bit binary numbers A and B. (April 2003)
4. Distinguish between combinational and sequential logic circuits. (April 2003, April 2004)
5. Using a single 7485 IC, draw the logic diagram of a 4 bit comparator. (April 2003)
6. Write down the truth table of a full adder. (April 2004)
7. Write down the truth table of a full subtractor. (Nov. 2004)
8. Describe the truth table of a half subtractor and write the Boolean expression corresponding to the difference and the borrow. (April 2005)
9. What is a combinational circuit? Give an example. (April 2005)
10. Draw the circuit of a half-adder. (Dec. 2005)
11. Write the truth table for a half-subtractor. (Dec. 2005)
12. Draw the flow diagram of gray to binary conversion. (Dec. 2005)
13. Draw a combinational logic circuit which can compare whether two bit binary numbers are same or not. (Dec. 2005)
14. Draw a parity checker circuit for 3 bit binary word $x_1 x_2 x_3$. (May 2006)
15. Represent a half adder in block diagram from and also its logic implementation. (May 2006)
16. Construct a 4-bit binary to gray code converter circuit and discuss its operation. (May 2006)
17. Mention any two uses of HDL. (May 2006)
18. What is logic synthesis? (Dec. 2006)
19. List the important features of HDL. (Dec. 2006)
20. Design a half adder. (May 2007)
21. What is a full adder? (May 2007)
22. What are the modeling techniques available to build HDL module? (May 2007)
23. Draw the circuit diagram for 3 bit parity generator. (Dec. 2007)
24. Define combinational circuit. (May 2009)

25. What is the need for code conversion? Give two commonly used codes. **(May 2009)**

PART-B

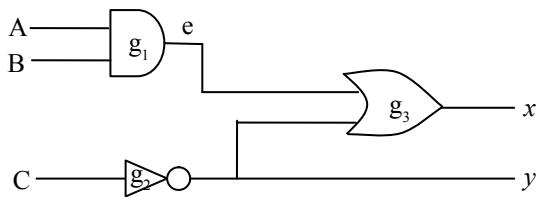
1. Design a parity generator to generate an odd parity bit for a 4-bit word. Use EX-OR and EX-NOR gates. **(8) (April 2003)**
2. Design a combinational circuit that compares two 4-bit numbers A and B to check if they are equal or not. **(8) (April 2003)**
3. (i) Design a full adder and a full subtractor. **(14)**
(ii) Draw the block diagram of a 2's complement adder/subtractor. **(2) (April 2003)**
4. Using a single 7483, draw the logic diagram of a 4 bit adder/subtractor.
5. Draw the Logic diagram of a 4 bit parallel adder/subtractor using full adders and explain. **(8) (Nov. 2003)**
6. Draw a full adder circuit using only NOR gates. **(8) (Nov. 2003)**
7. Design and explain the following circuits: (i) Full adder, (ii) Comparator. **(16) (Nov. 2003)**
8. (i) Explain how a full adder can be built using two half adders and an OR gate. **(6)**
(ii) Design a half adder using atmost three NOR gates. **(10) (April 2004)**
9. Design a look ahead carry generator. **(16) (April 2004)**
10. Design a circuit that converts 8421 BCD Code to excess 3 code. **(16) (April 2004)**
11. Implement the logic function $Y(A, B, C) = \sum m(1, 2, 7)$ using 74151A and 74153. **(8) (Nov. 04)**
12. Design a Binary to Gray Converter. **(16) (Nov. 2004)**
13. Design a BCD to Gray Code Converter. Use don't cares. **(16) (Nov. 2004)**
14. Explain carry look ahead adder circuit. **(16) (Nov. 2004)**
15. Explain with truth table and gate level circuit diagram for a full adder. **(12) (April 2005)**
16. Design a combinational circuit which accepts 3 bit binary number and converts its equivalent excess 3 code. **(10) (April 2005)**
17. Design and explain the working of Gray to BCD converter. **(16) (April 2005)**
18. Design a BCD adder to add two BCD digits. **(16) (Dec. 2005)**
19. (i) Design a 4-bit binary to BCD code converter. **(10) (Dec. 2005)**
(ii) Design a 4-bit binary to gray code converter. **(6) (Dec. 2005)**
20. Design and implement a binary to gray code converter. **(16) (Dec. 2005)**
21. Design a combinational circuit whose input is 4 bit binary number and whose output is 2's complement of input number. **(4) (Dec. 2005)**
22. Design a combinational logic circuit that will generate the square of all the combinations of a 3 bit binary number. **(8) (Dec. 2005)**
23. Design and explain: (i) 4 bit magnitude comparator. **(16) (May 2006)**

-
24. Design a BCD to 7 segment code converter. **(16) (May 2006)**
25. (i) Design a binary multiplexer, using half adders and/or full adders, to multiply two 2-bit numbers. **(8)**
(ii) Design a combinational logic circuit to compare two 2-bit binary numbers A and B and to generate the outputs $A < B$, $A = B$ and $A > B$. Is there a way to derive the third output from the first two outputs? **(8) (May 2006)**
26. Design a 4 bit magnitude comparator to compare two 4 bit numbers. **(16) (Dec. 2006)**
27. Construct a combinational circuit to convert given binary coded decimal number into an Excess-3 code. For example when the input to the gate is 0110 then the circuit should generate output as 1001. **(16) (Dec. 2006)**
28. Design a 4 bit comparator using logic gates. **(16) (May 2007)**
29. Design a 4 bit adder/subtractor using logic gates and explain its operation. **(16) (May 2007)**
30. Draw the block diagram of a BCD adder and explain its operation. **(10) (May 2007)**
31. (i) Design a combinational circuit to convert BCD to gray code. **(12) (May 2007)**
(ii) Design a 4 bit subtractor. **(4) (May 2007)**
32. (i) Design a combinational circuit to convert Excess-3 code to BCD code. **(10) (May 2007)**
(ii) Design a 2 bit x 2 bit multiplier. **(6) (May 2007)**
33. (i) Design a combinational circuit to convert gray code to BCD. **(12) (Dec. 2007)**
(ii) Design a Full adder circuit with a Decoder. **(4) (Dec. 2007)**
34. Design a 4 bit magnitude comparator to compare two 4 bit numbers. **(Dec. 2007)**
35. (i) Design a combinational circuit with three inputs and one output. The output is 1 when the binary value of the inputs is less than three. The output is 0 otherwise. **(6) (Dec. 2008)**
(ii) Design a code converter that converts a decimal digit from 8,-4,-2,-1 code to BCD. **(10) (Dec. 2008)**
36. (i) Design a 4 bit combinational circuit incrementer – A circuit that adds one to a 4-bit binary number? Use half-adders for this problem. **(8) (Dec. 2008)**
(ii) Design a combinational circuit that generates the 9's complement of a BCD digit. **(8) (Dec. 2008)**
37. Design a BCD to 7 segment decoder. **(16) (May 2009)**
38. With a suitable block diagram explain the operation of BCD adder. **(16) (May 2009)**

COMBINATIONAL LOGIC

PART-A

1. Distinguish between a decoder and a multiplexer. (Nov 2003, April 2004)
2. Draw a 1 to 2 demultiplexer circuit. (Nov 2003)
3. Draw a 1 to 2 multiplexer circuit. (Nov 2003)
4. What is a demux? (April 2004)
5. What is a decoder and obtain the relation between the number of inputs 'n' and outputs 'm' of a decoder? (April 2005)
6. Implement the logic function $f = AB + A'.B'$ using a suitable multiplexer. (Dec 2005)
7. How can a decoder be converted into a demultiplexer? (Dec 2005)
8. Distinguish between decoder and multiplexer. (Dec 2005)
9. Implement the logic function $f = \sum m(0,2,3,6)$ using a decoder. (May 2006)
10. How can a multiplexer used to convert 8-bit parallel data into serial form? (May 2006)
11. What are functions of encoders and decoders? (Dec. 2006)
12. What is a Multiplexer? (Dec. 2006)
13. Design a 2 input NAND gate using 2:1 multiplexer. (May 2007)
14. Implement the given function in 4:1 multiplexer $f = \sum m(0,1,3,5, 6)$. (May 2007)
15. What is a priority encoder? (May 2007)
16. Mention any two applications of multiplexers. (May 2007)
17. What is logic syntheses in HDL? (Dec. 2007)
18. Construct a 16x1 multiplexer with two 8x1 multiplexer and 2x1 multiplexer. (Dec. 2008)
19. Draw the logic diagram of 4 bit even parity checker. (Dec. 2008)
20. What is decoder? Draw the block diagram and truth table for 2 to 4 decoder. (May 2009)
21. Give some applications of multiplexer. (May 2009)
22. Compare the serial and parallel adder. (Dec 2010)
23. Define look ahead carry addition. (Dec 2010)
24. Define priority encoder (Dec 2010)
25. Write a dadflow description of a 2-to-1 line Mux uisng a condiiitonal operator (Dec 2010)
24. Draw the schematic of half-adder logic. (May 2011)
25. Determine the size and number of multiplexers required to implement a full adder. (May 2011)
26. Write the HDL description of the following circuit. (May 2011)



27. With a block diagram show how a full adder can be designed using two half adder and one OR gate. (Dec 2011)
28. List the modelling techniques available in HDL. (Dec 2011)
29. Define decoder. Draw the block diagram and truth table for 2 to 4 decoder. (Dec 2011)
30. Define Tri-state gates. (May 2012)
31. Define Logic Synthesis and Simulation. (May 2012)
32. Write the stimulus for 2-to-1 line multiplexer. (May 2012)
33. Implement a Full adder with two half adders. (Dec 2012)
34. Implement a 4-bit even parity checker. (Dec 2012)
35. Construct a 4×16 decoder using 3×8 decoders. (Dec 2012)
36. Write down the truth table of a full subtractor. (May 2013)
37. What is meant by Test Bench? (May 2013)
38. Distinguish between a decoder and a demultiplexer. (May 2013)

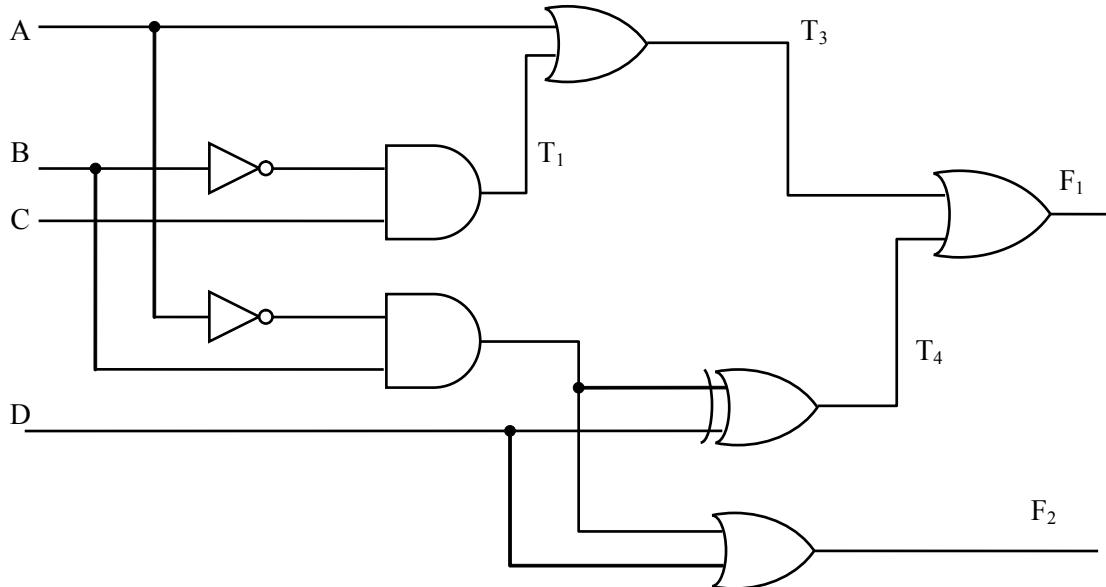
PART – B

- Implement the function $F(X_1, X_2, X_3, X_4) = \sum (0, 1, 3, 4, 8, 9, 15)$ with an 8×1 multiplexer where the following variables are connected in the specified order to selection lines S_2, S_1 and S_0 respectively
(i) X_1, X_2, X_3 (ii) X_2, X_3, X_4 (8) (April 2003)
- Design and explain the working of a 1 to 8 demultiplexer. (10) (April 2003)
- Explain how you will build a 64 input MUX using nine 8 input MUXs. (8) (April 2003)
- Implement the following Boolean function using an 8 to 1 multiplexer and 4 to 1 multiplexer.
 $F(A, B, C) = \sum (0, 1, 5, 7)$. (12) (Nov 2003)
- Design and explain the working of a 4 to 1 multiplexer. (8) (Nov 2003)
- Realize $F(w, x, y, z) = \sum (1, 4, 6, 7, 8, 9, 10, 11, 15)$ using 4 to 1 MUX. (16) (April 2004)
- (i) Implement a 3 to 8 line decoder. (8)

- (ii) Implement the logic function $Y(A,B,C) = \sum m(1,2,7)$ using 74151A and 74153. (8) (Nov 2004)
8. Design a seven segment decoder circuit to display the numbers 0 to 3. (16) (Nov 2004)
9. Implement the switching function $F = \sum (3, 6-8, 10, 13-15)$ using an 8 input multiplexer. (10) (Nov 2004)
10. Show that when two 2 input multiplexers drive another 2 input MUX, the result is a 4 input multiplexer. (6) (Nov 2004)
11. Show that when a 3 line to 8 line DEMUX drives eight 3 line to 8 line DEMUXs, the result is a 6 line to 64 line DEMUX. (6) (Nov 2004)
12. Implement the following function with a multiplexer, $f(a,b,c,d) = \sum (0,1,3,4,8,9,15)$. (10) (April 2005)
13. Implement the following Boolean expression using an 4:1 MUX $F = \sum (0,1,2,4,6,9,12,14)$. (8) (April 2005)
14. Design and explain the working of full adder and a decoder. (12) (April 2005)
15. What is the simplest logic circuit for a decoder that produces a 1 output when the BCD input is 0000 ? (8) (Dec 2005)
16. Implement the following function with a multiplexer, $f(A,B,C,D) = \sum (0,1,3,4,8,9,15)$. (6) (Dec 2005)
17. Write the structural VHDL description for a 2 to 4 decoder and explain it in detail. (16) (Dec 2005)
18. Implement the following Boolean function using an 8:1 MUX $f(a,b,c,d) = a'b'd' + acd + b'cd + a'c'd$ (8) (Dec 2005)
19. Design and explain the priority encoder. (8) (May 2006)
20. Describe the structural verilog description of 4 to 1 multiplexer. Also draw the internal diagram of the multiplexer. (16) (May 2006)
21. Construct a full adder circuit and write a HDL program module for the same. (16) (Dec. 2006)
22. (i) Implement the following with a multiplexer
 $F(A,B,C) = \sum(1,2,4,5)$ (8) (Dec. 2006)
23. Implement the Boolean function using 8:1 multiplexer.
 $F(A,B,C,D) = A'BD + ACD + B'CD + A'C'D.$ (8) (May 2007)

24. Construct a full adder circuit and write a HDL program module for the same. **(16) (May 2007)**
25. Implement the Boolean function using 8:1 multiplexer.
 $F(A,B,C,D) = AB'D + A'C'D + B'CD' + AC'D.$ **(16) (Dec. 2007)**
26. (i) A 8x1 multiplexer has inputs A,B and C connected to the selection inputs S2, S1 and S0 respectively. The data inputs 10 to 17 are as follows 11=12=17=0; 13=15=15=1; 10=14=D and 16=D'. Determine the Boolean function that the multiplexer implements. **(10) (Dec. 2008)**
(ii) Write the HDL dataflow description of a quadrupul -2 to -1 line multiplexers with enable. **(6) (Dec. 2008)**
27. (i) using the conditional operator (?;), write a HDL, dataflow description of a 4-bit adder subtractor of unsigned numbers. **(8) (Dec. 2008)**
(ii) Implement the Boolean function $F = \sum(0,1,3,4,8,9,15)$ using a multiplexer. **(8) (Dec. 2008)**
28. Define decoder. Design a 3 to 8 decoder. With suitable block diagram explain how a 4 to 16 decoder can be formed by using the same. **(16) (May 2009)**
29. Relative a BCD to Excess-3 code conversion circuit starting from its truth table. **(16) (Dec 2010)**
30. Design a full adder and subtractor using NAND and NOR gates respectively. **16 (Dec. 2010)**
31. (i) Define multiplexer
(ii) Implemment the following Boolean function using 8 : 1 MUX.
 $F(A, B, C, D) = ABC + ACD + BCD + ACD$ **16 (Dec. 2010)**
32. (i) Design a combinational circuit that comprises only of NOR gates for the following expression giving the input output relation. **(10)**
(ii) Draw the schematic of a full adder circuit and give its truth table. **(6) (May 2011)**
33. (i) Design a BCD to Excess - 3 code converter using truth table and K-Map simplification **(10) (May 2011)**
(ii) Draw the schematic of a magnitude comparator and give its truth table. **(6) (May 2011)**

34. (i) Design a combinational logic using a suitable multiplexer to realize the following Boolean expression. (10)
- (ii) Compare and contrast between encoders and multiplexers. (6) (May 2011)
35. Consider the combinational circuit shown in figure.

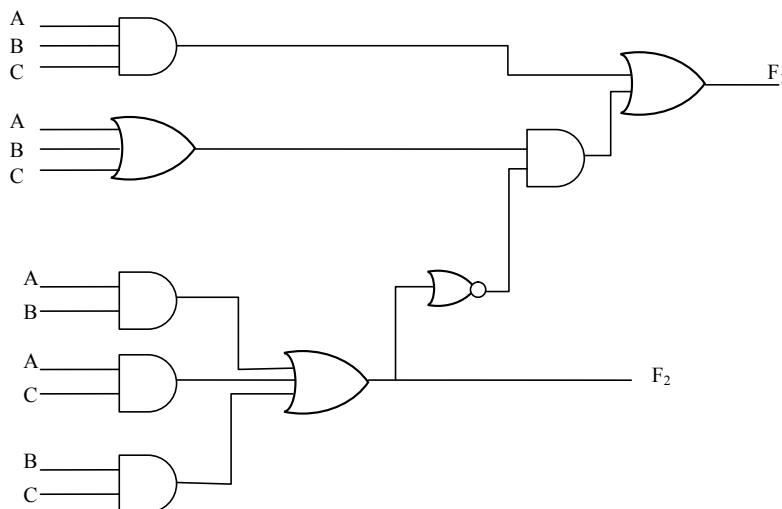


- (i) Derive the Boolean expressions for T₁ through T₄. Evaluate the outputs F₁ and F₂ as a function of the four inputs. (4)
- (ii) List the truth table with 16 binary combinations of the four input variables. Then list the binary values for T₁ through T₄ and outputs F₁ and F₂ in the table. (4)
- (iii) Plot the output Boolean function obtained in part (ii) on maps and show that simplified Boolean expressions are equivalent to the ones obtained in Part (i) (8) (Dec. 2011)
36. (i) With suitable block diagram explain Binary multiplier. (8)
- (ii) Write a detailed note on carry-propagation. (8) (Dec. 2011)
37. Construct a 5 to 32 line decoder with four 3 to 8 line decoders with enable and a 2 to 4 line decoder. Use block diagrams for components. (16) (Dec. 2011)
38. (i) Implement the following Boolean function with 16 x 1 multiplexer : (6)

$$F(A, B, C, D) = \sum(0, 1, 3, 4, 8, 9, 15)$$

Use block diagram representation.

- (ii) Write HDL gate level description for 3 to 8 line decoder. (4) (Dec. 2011)
39. Design Half and Full Substractor circuits. (16) (May 2012)
40. Design a circuit that converts 8421 BCD code to Excess-3 code. (16) (May 2012)
41. Implement a full adder with two 4×1 multiplexers. (16) (May 2012)
42. i) Analyse the combinational circuit shown in figure, determine the truth table and the Boolean expressions governing the outputs of the circuit. (10)



- ii) Explain BCD adder with a neat block diagram. (6) (Dec 2012)
43. i) Design a BCD to excess-3 code converter using logic gates. (12)
ii) Draw the diagram of a 4-bit adder subtractor using full adder. (4) (Dec 2012)
44. Implement $F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$ using 8×1 multiplexer. (8) (Dec 2012)
45. Design a 4-input priority encoder. (6) (Dec 2012)
46. Design a full adder using 2 half adders. (16) (May 2013)
47. Design a combinational circuit to convert binary to gray code. (16) (May 2013)
48. Implement the switching function $F = \sum m(0, 1, 3, 4, 12, 14, 15)$ using an 8 input MUX. (16) (May 2013)

UNIT – III

SYNCHRONOUS SEQUENTIAL LOGIC

- Sequential Circuits
- Latches and Flip Flops
- Analysis and Design Procedures
- State Reduction and State Assignment
- Shift Registers
- Counters
- HDL for Sequential Logic Circuits

SYNCHRONOUS SEQUENTIAL LOGIC

3.1 INTRODUCTION

In combinational logic circuits, the outputs at any instant of time depend only on the input signals present at that time as shown in **Figure 3.1**.

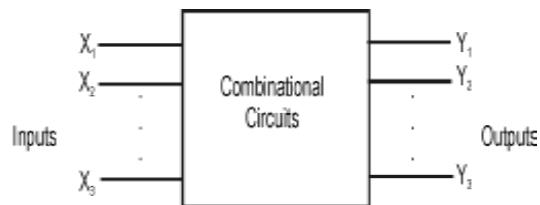


Fig. 3.1: Combinational Circuit

The logic circuits whose outputs at any instant of time depend not only on the present inputs but also on the past outputs are called sequential logic circuits. **Figure 3.2** shows the block diagram of sequential logic circuit.

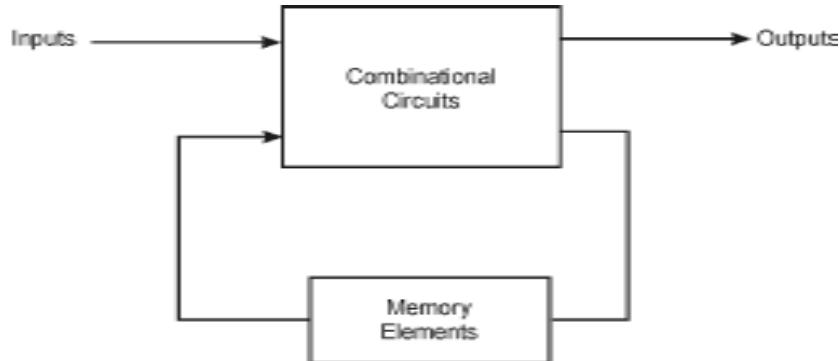


Fig. 3.2: Sequential Circuit-Block Diagram

The comparison between combinational and sequential circuits is given in **Table 3.1**.

TABLE 3.1 : Comparison between combinational and sequential circuits

<i>Combinational Circuits</i>	<i>Sequential Circuits</i>
Output depends on the present state input.	Output depends not only on present input but also on the past output.
Faster than sequential circuit.	Low speed.
Memory unit is not required.	Memory unit is required.
Easy to design.	Design is not easy.

The rotary channel selected knob on an old-fashioned TV is like a combinational circuit. Its output selects a channel based only on its current input – the position of the knob. The channel-up and channel-down push buttons on a TV is like a sequential circuit. The channel selection depends on the past sequence of up/down pushes.

3.2 CLASSIFICATION OF LOGIC CIRCUITS

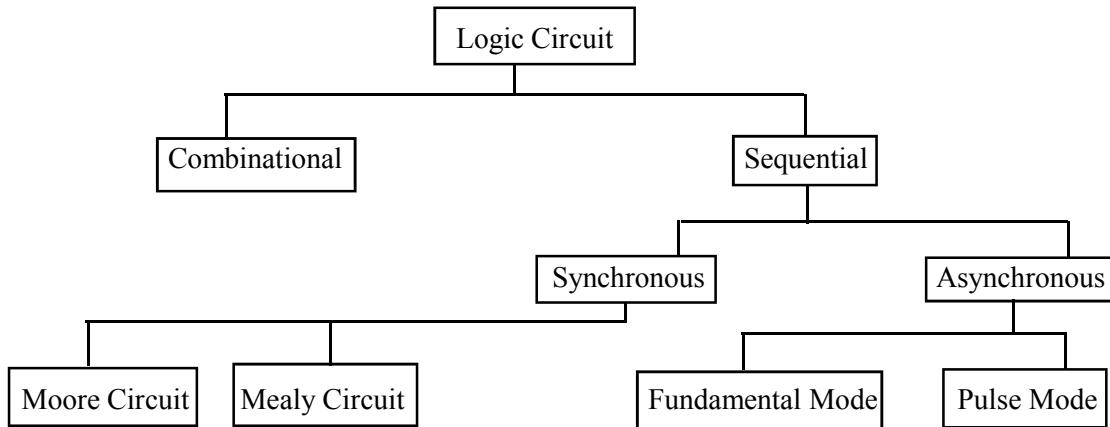


Fig. 3.3: Classification of Logic Circuits

In the synchronous or clocked sequential circuits, signals can affect the memory elements only at discrete instants of time. In the asynchronous or unclocked sequential circuits, change in input signals can affect memory element at any instant of time.

3.3 FLIP-FLOPS

The memory elements used in synchronous sequential circuits are called flipflops. A flip-flop circuit has two outputs one for the normal value and another for the complement value of the bit stored in it. Flip-flops flip from one state to another and then flop back. They are known as bistable multivibrators.

The output of the flip-flop is either logic 0 (0 volt) or logic 1 (+5 V). The flip-flop output will remain 0 or 1 until the trigger pulse is given to change the state. This means that it can store 1 bit information. The block diagram of a flip-flop is shown in **Figure 3.4**.

The Types of flip-flops are:

- (i) SR flip-flop
- (ii) JK flip-flop
- (iii) D flip-flop
- (iv) T flip-flop

The Applications of flipflops are:

- (i) Counters
- (ii) Frequency dividers
- (iii) Shift registers
- (iv) Storage registers

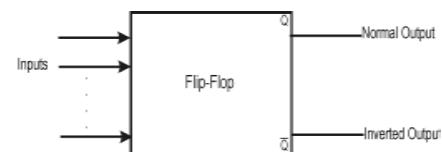


Fig. 3.4: Block Diagram of a Flip-Flop

3.4 SR FLIP-FLOP

The SR flip-flop has two inputs: S(set) and R(reset) and two outputs: Q(normal output) and \bar{Q} (inverted output). The symbol of SR flip-flop is shown in **Figure 3.5**. The NOR SR flip-flop is shown in **Figure 3.6**. The cross coupled connections from the output of one NOR gate to the input of the other NOR gate form a feedback path.

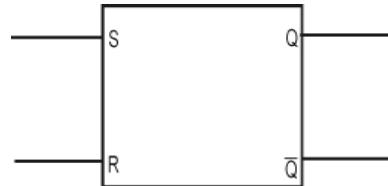


Fig. 3.5: Symbol of SR flip-flop

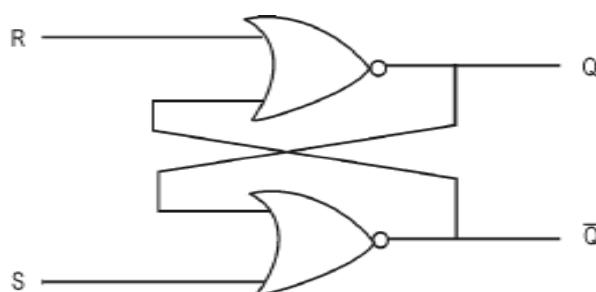


Fig. 3.6: S - R flip-flop

When $S = 0$ and $R = 0$, the output, Q_{n+1} remains in its present state, Q_n .

When $S = 0$ and $R = 1$, the flipflop reset to 0.

When $S = 1$ and $R = 0$, the flipflop set to 1.

When $S = 1$ and $R = 1$, the output of both gates will produce 0. $Q_{n+1} = \bar{Q}_{n+1} = 0$.

The truth table of NOR based SR flip-flop is shown in **Table 3.2**.

TABLE 3.2: Truth Table of SR flip-flop

Inputs		Outputs		State
S	R	Q_{n+1}	\bar{Q}_{n+1}	
0	0	Q_n	\bar{Q}_n	No change
0	1	0	1	Reset
1	0	1	0	Set
1	1	X	X	Forbidden

$Q_n \Rightarrow$ Present State ; $Q_{n+1} \Rightarrow$ Next State

The SR flip-flop can be also implemented using NAND gates. The inputs of this flip-flop are \bar{S} and \bar{R} .

When $\bar{S} = \bar{R} = 0$, Outputs $Q_{n+1} = \bar{Q}_{n+1} = 1$.

When $\bar{S} = 0, \bar{R} = 1$, the flip-flop output $Q_{n+1} = 1$ (set).

When $\bar{S} = 1, \bar{R} = 0$, the flip-flop output $Q_{n+1} = 0$ (reset).

When $\bar{S} = \bar{R} = 1$, the output remains in its prior state.

The NAND based SR flip-flop is shown in **Figure 3.7** and the truth table is given in **Table 3.3**.

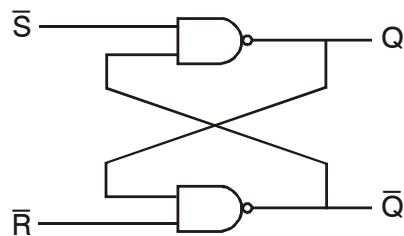


Fig. 3.7: NAND based SR flip-flop

TABLE 3.3: Truth Table of $\bar{S} \bar{R}$ flipflop

Inputs		Outputs		State
\bar{S}	\bar{R}	Q_{n+1}	\bar{Q}_{n+1}	
0	0	X	X	Forbidden
0	1	1	0	Set
1	0	0	1	Reset
1	1	Q_n	\bar{Q}_n	No change

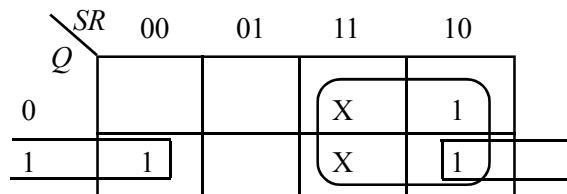
3.4.1 Characteristic Table and Characteristic Equation

The characteristic table shows the operation of the flip-flop in tabular form. Q is the present state (Q_n) and Q_{n+1} stands for next state. The characteristic table of SR flip-flop is shown in **Table 3.4**.

The characteristic equation is an algebraic expression for the binary information of the characteristic table. This equation is derived from K-map. This equation specifies the value of the next state as a function of the present state and the inputs.

TABLE 3.4: Characteristic Table

<i>Q</i>	<i>S</i>	<i>R</i>	<i>Q_{n+1}</i>
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	Forbidden
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	Forbidden



Characteristic equation:
$$Q_{n+1} = S + Q\bar{R}$$

3.5 CLOCKED SR FLIP-FLOP

Synchronous circuits change their states only when clock pulses are present. The clocked SR flip-flop is shown in **Figure 3.8**. It consists of a basic SR flipflop circuit and two additional NAND gates. The pulse input acts as an enable signal (EN) for the other two inputs. The outputs of NAND gates 3 and 4 stay at the logic level 1 as long as the clock pulse input remains 0. When the pulse input goes to 1, information from the S or R input is allowed to reach the output.

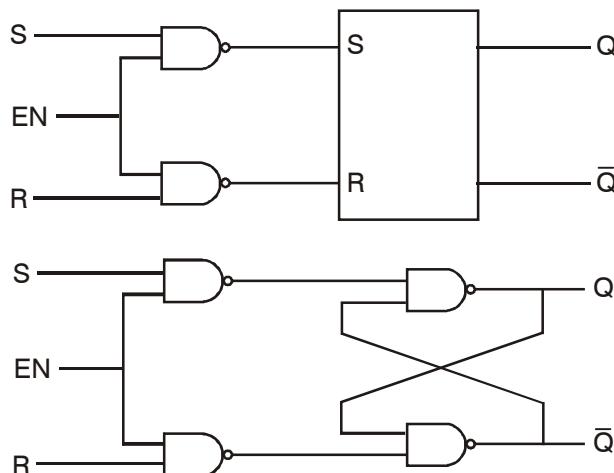


Fig. 3.8 : Clocked S-R flip-flop

The characteristic table and characteristic equation of clocked SR flip-flop are same as the characteristic table (**Table 3.4**) and characteristic equation of SR flip-flop. Note that the CP input is not included in **Table 3.4**. The table must be interpreted as: Given the present state Q and the inputs S and R, the application of a single pulse in the EN input causes the flip-flop to go to the next state Q_{n+1} .

3.6 TRIGGERING OF FLIP-FLOPS

The state of a flip-flop is switched by a momentary change in the input signal. This momentary change is called a trigger and the transition it causes is said to trigger the flip-flop. Clocked flip-flops are triggered by pulses. A clock pulse starts from an initial value of 0, goes momentarily to 1, and after a short time, returns to its initial 0 value. Based on the specific interval or point in the clock during or at which triggering of flip-flop takes place, it can be classified into two different types:

- ♦ Level triggering,
- ♦ Edge triggering.

3.6.1 Level Triggering

In Level triggering, the flip-flops are enabled in HIGH (+ve level) or LOW (−ve level) level. The flip-flop action is dependent on the entire period of the pulse.

Positive Level Triggering: If the flip-flop changes its state when the clock is positive, it is called as positive level triggering.

Negative Level Triggering: If the flip-flop changes its state when the clock is negative, it is called as negative level triggering.

3.6.2 Edge Triggering

A clock pulse may be either negative or positive. A positive clock source remains at 0 during the interval between pulses and goes to 1 during the occurrence of a pulse. The pulse goes through two signal transitions: from 0 to 1 and returns from 1 to 0. The positive transition is defined as the positive edge and the negative transition is defined as the negative edge. **Figure 3.9 (a)** shows the positive pulse and **Figure 3.9(b)** shows the negative pulse with +ve edge and -ve edge.

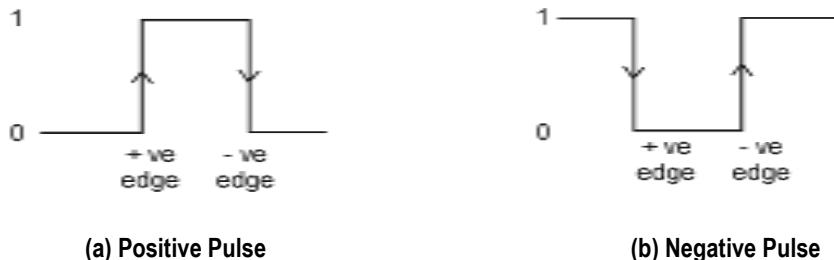


Fig. 3.9: Edge Triggering

The flip-flop changes its state either at the positive edge or at the negative edge of the clock pulse and is sensitive to its inputs only at this transition of the clock.

To make the flip-flop respond only to a pulse transition is to use capacitive coupling. An RC (resistor-capacitor) circuit is inserted in the clock input of the flipflop. The RC circuit is shown in **Figure 3.10**. This circuit generates a spike in response to a momentary change of input signal.

A positive edge emerges from the RC circuit with a positive spike and a negative edge emerges with a negative spike. Edge triggering is achieved by designing the flip-flop to neglect one spike and trigger on the occurrence of the other spike.

The graphic symbols of level triggered and edge triggered flip-flops are given in **Figure 3.11**.

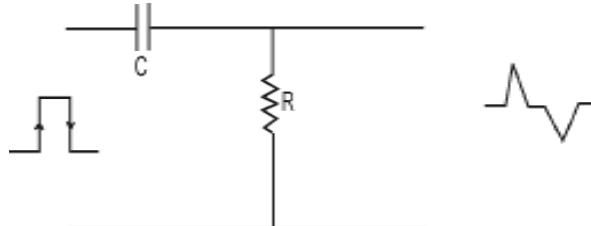
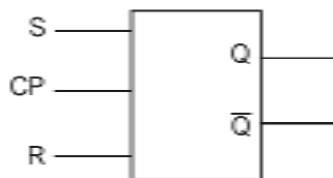
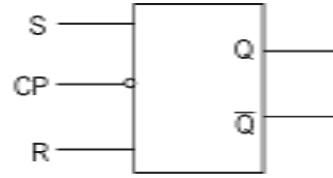


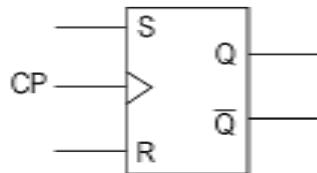
Fig. 3.10: RC circuit



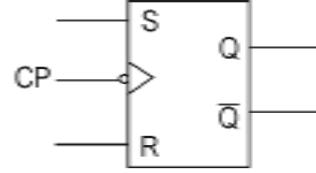
(a) Positive level triggered SR flipflop



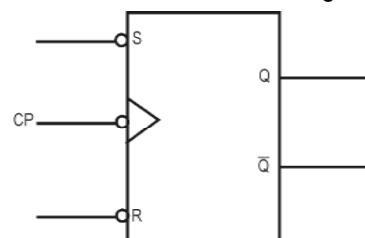
(b) Negative level triggered SR flipflop



(c) Positive edge triggered SR flipflop with active high inputs



(d) Negative edge triggered SR flip flop with active high inputs



(e) Negative edge triggered with active low inputs

Fig. 3.11 : Graphic Symbols

3.7 D FLIP-FLOP

To eliminate the undesirable condition of the indeterminate state in the RS flip-flop is to ensure the inputs S and R are never made equal to 1 at the same time. This is done by D flip-flop. The D (delay) flip-flop has one input called delay input and clock pulse input. The D flip-flop using SR flip-flop is shown in **Figure 3.12(a)** and the graphic symbol is shown in **Figure 3.12(b)**.



(a) Using SR flipflop

(b) Graphic symbol

Fig. 3.12 : D flip-flop

The truth table of D flip-flop is given in **Table 3.5**.

TABLE 3.5: Truth Table

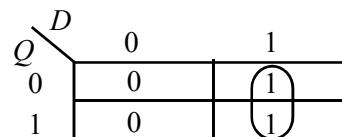
Clock	Input <i>D</i>	Output <i>Q_{n+1}</i>	State
1	0	0	Reset
1	1	1	Set
0	X	Q _n	No change

3.7.1 Characteristic Table and Characteristic Equation

The characteristic table for the *D* flip-flop is shown in **Table 3.6**. It shows that the next state of the flip-flop is independent of the present state since *Q_{n+1}* is equal to *D*, whether *Q* is equal to 0 or 1. This means that an input pulse will transfer the value of input *D* into the output of the flipflop independent of the value of the output before the pulse was applied. The characteristic equation is derived from *K*-map.

TABLE 3.6: Characteristic Table

<i>Q</i>	<i>D</i>	<i>Q_{n+1}</i>
0	0	0
0	1	1
1	0	0
1	1	1



Characteristic equation: $Q_{n+1} = D$

3.8 JK FLIP-FLOP

JK means Jack Kilby, Texas Instrument (TI) Engineer, who invented IC in 1958. JK flipflop has two inputs J(set) and K(reset). A JK flip-flop can be obtained from the clocked SR flipflop by augmenting two AND gates as shown in **Figure 3.13**. The data input J and the output \bar{Q} are applied to the first AND gate and its ouput ($J\bar{Q}$) is applied to the S input of SR flipflop. Similarly, the data input K and the output Q are applied to the second AND gate and its output (KQ) is applied to the R input of SR flip-flop.

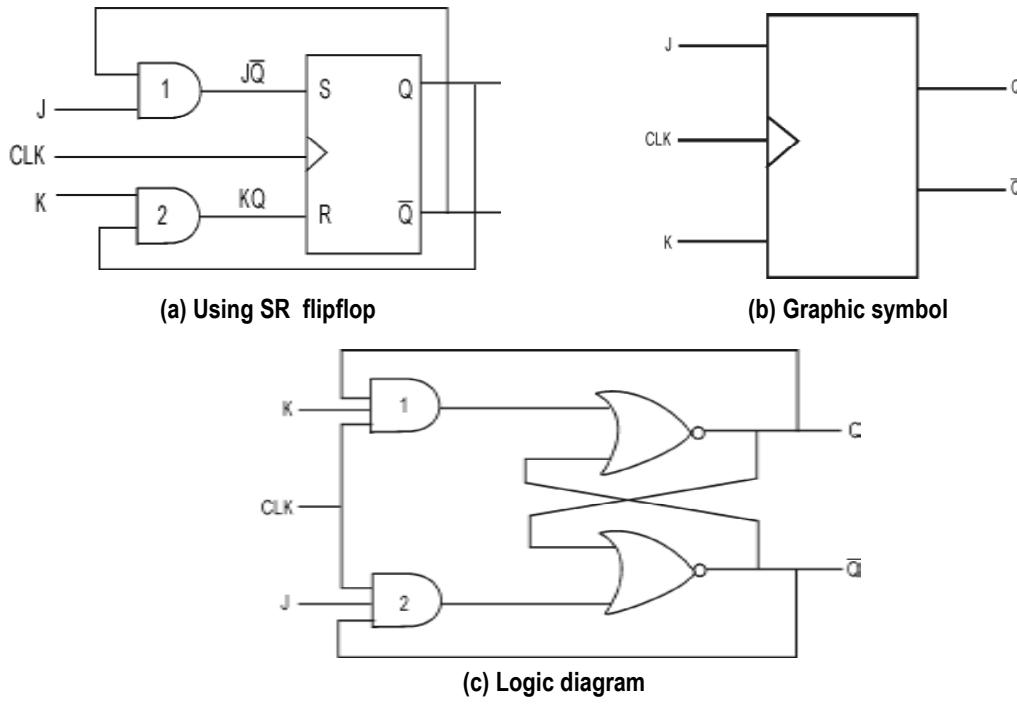


Fig. 3.13: JK flip-flop

$J = K = 0$

When $J = K = 0$, both AND gates are disabled. Therefore clock pulse have no effect, hence the flip-flop output is same as the previous output.

$J = 0, K = 1$

When $J = 0$ and $K = 1$, AND gate 1 is disabled i.e., $S = 0$ and $R = 1$. This condition will reset the flipflop to 0.

$J = 1, K = 0$

When $J = 1$ and $K = 0$, AND gate 2 is disabled i.e., $S = 1$ and $R = 0$. Therefore the flip-flop will set on the application of a clock pulse.

$J = K = 1$

When $J = K = 1$, it is possible to set or reset the flip-flop. If Q is high, AND gate 2 passes on a reset pulse to the next clock. When Q is low, AND gate 1 passes on a set pulse to the next clock. Eitherway, Q changes to the complement of the last state i.e., toggle. Toggle means to switch to the opposite state.

The truth table of JK flip-flop is given in **Table 3.7**.

TABLE 3.7: Truth Table

CLK	Inputs		Output Q_{n+1}	State
	J	K		
1	0	0	Q_n	No change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	\bar{Q}	Toggle

3.8.1 Characteristic Table and Characteristic Equation

The characteristic table for the JK flip-flop is shown in **Table 3.8**. From the **Table 3.8**, K-map for the next state transition (Q_{n+1}) can be drawn and the simplified logic expression which represents the characteristic equation of JK flip-flop can be found.

TABLE 3.8: Characteristic Table

Q	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

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

Characteristic equation:
$$Q_{n+1} = J\bar{Q} + \bar{K}Q$$

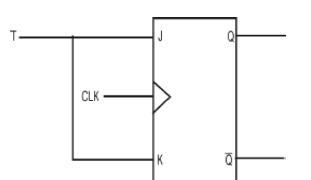
3.9 T FLIP-FLOP

The T(Toggle) flipflop is a modification of the JK flipflop. It is obtained from JK flip-flop by connecting both inputs J and K together, i.e., single input. Regardless of the present state, the flip flop complements its output when the clock pulse occurs while input $T = 1$.

When $T = 0$, $Q_{n+1} = Q_n$, i.e., the next state is the same as the present state and no change occurs.

When $T = 1$, $Q_{n+1} = \bar{Q}_n$, i.e., the next state is the complement of the present state.

The symbol and truth table of T flip-flop is shown in **Figure 3.14**.



(a) Graphic symbol

T	Q_{n+1}	State
0	Q_n	No change
1	\bar{Q}	Complement

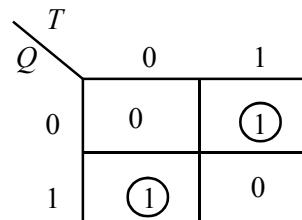
(b) Truth Table

Fig. 3.14 : T flipflop

The characteristic table is shown in **Table 3.9** and characteristic equation is derived using K-map.

TABLE 3.9 : Characteristic table.

Q	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0



Characteristic Equation:

$$Q_{n+1} = T\bar{Q} + \bar{T}Q$$

3.10 MASTER-SLAVE J-K FLIPFLOP

A master-slave flip-flop is constructed using two separate JK flipflops. The first flip-flop is called the master. It is driven by the positive edge of the clock pulse. The second flip-flop is called the slave. It is driven by the negative edge of the clock pulse. The logic diagram of a master-slave JK flipflop is shown in **Figure 3.15**.

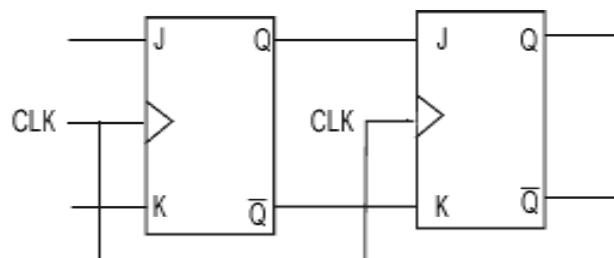


Fig. 3.15 : Logic diagram of Master-slave JK flipflop

When the clock pulse has a positive edge, the master acts according to its J - K inputs, but the slave does not respond, since it requires a negative edge at the clock input.

When the clock input has a negative edge, the slave flip-flop copies the master outputs. But the master does not respond since it requires a positive edge at its clock input.

The clocked master-slave J - K flipflop using NAND gates is shown in **Figure 3.16**.

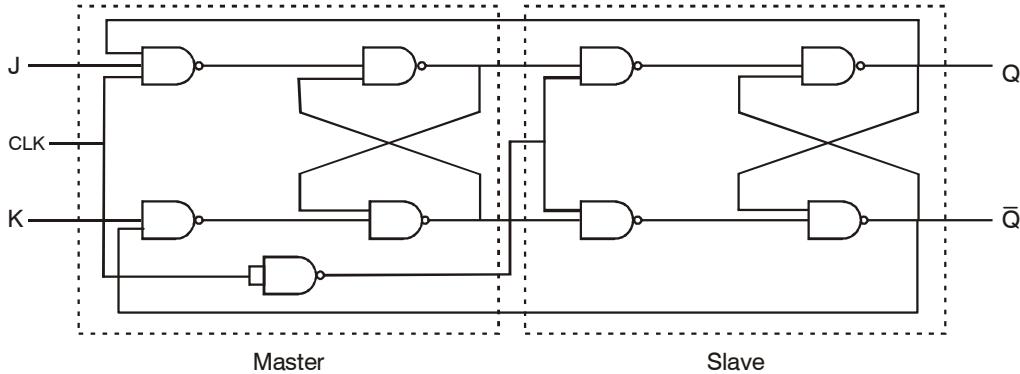


Fig. 3.16 : Master Slave JK flipflop

3.11 APPLICATION TABLE (or) EXCITATION TABLE

The characteristic table is useful for analysis defining the operation of the flip-flop. It specifies the next state (Q_{n+1}) when the inputs and present state are known.

The excitation or application table is useful for design process. It is used to find the flip-flop input conditions that will cause the required transition, when the present state (Q_n) and next state (Q_{n+1}) are known.

3.11.1 SR Flipflop

TABLE 3.10 (a) : Characteristic Table

Present State	Inputs		Next State
Q_n	S	R	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

TABLE 3.10 (b) : Modified Table

Present State	Next State	Inputs		Inputs	
		S	R	S	R
0	0	0	0	0	X
0	0	0	1	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	1	0	0	X	0
1	1	1	0	0	X

The excitation table for SR flipflop is derived from the characteristic table. **Table 3.10(a)** shows the characteristic table of SR flipflop and rearrangement of columns is shown in **Table 3.10(b)**.

Table 3.11 presents the excitation table for SR flipflop. It consists of present state (Q_n), next state (Q_{n+1}) and a column for each input to show how the required transition is achieved. There are 4 possible transitions from present state to next state. The required input conditions for each of the four transitions are derived from the information available in the characteristic table. The symbol X denotes the don't care condition, it does not matter whether the input is 1 or 0.

TABLE 3.11: Excitation Table

Present State	Next State	Inputs	
		S	R
Q_n	Q_{n+1}		
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

3.11.2 J - K flipflop

Table 3.12(a) represents the characteristic table of JK flipflop and **Table 3.12(b)** represents the rearrangement of the characteristic table. The required input conditions for each of the four transitions are derived from the information available in the characteristic table. The excitation table for JK flipflop is shown in **Table 3.13**.

TABLE 3.12 (a) : Characteristic Table

Present State	Inputs		Next State
Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

TABLE 3.12 (b) : Modified Table

Present State	Next State	Inputs		Inputs	
		J	K	J	K
Q_n	Q_{n+1}				
0	0	0	0	0	X
0	0	0	1	1	X
0	1	1	0	1	X
0	1	1	1	X	1
1	0	0	1	X	1
1	0	1	1	1	0
1	1	0	0	X	0
1	1	1	0	0	X

TABLE 3.13: Excitation Table

Present State	Next State	Inputs	
		J	K
Q_n	Q_{n+1}		
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

3.11.3 D flip-flop

The characteristic table and excitation table for the D flipflop are shown in **Table 3.14** and **Table 3.15** respectively.

Table 3.14: Characteristic Table

Present State	Delay input	Next State
Q_n	D	Q_{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

Table 3.15: Excitation Table

Present State	Next State	Input
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

3.11.4 T flipflop

The characteristic table and excitation table for the T flipflop are shown in **Table 3.16** and **Table 3.17** respectively.

TABLE 3.16: Characteristic Table

Present State	Input	Next State
Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

TABLE 3.17: Excitation Table

Present State	Next State	Input
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

3.12 REALIZATION OF ONE FLIP-FLOP USING OTHER FLIP-FLOPS

It is possible to implement a flip-flop circuit using any other flip-flop. The realization of one flipflop using other flipflops is implemented by the use of characteristic tables and excitation tables. The examples are:

- D flipflop using SR flipflop
- D flipflop using JK flipflop
- D flipflop using T flipflop
- T flipflop using SR flipflop
- T flipflop using JK flipflop
- T flipflop using D flipflop
- JK flipflop using SR flipflop
- JK flipflop using D flipflop

3.12.1 Realization of D flip-flop using SR flipflop

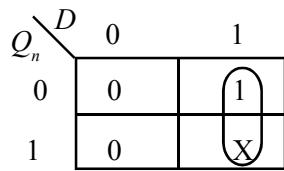
(April 2004)

- Write the characteristic table for required flipflop. (D flip-flop)
- Write the excitation table for given flipflop. (SR flip-flop)
- Determine the expression for the given flip-flop inputs (S and R) by using K-map.
- Draw the flip-flop conversion logic diagram to obtain the required flip-flop (D flipflop) by using the above obtained expression.

TABLE 3.18 : Excitation Table for D flipflop realization using SR flipflop

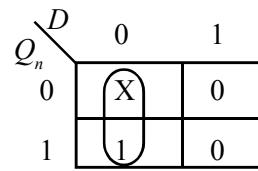
Required Flip-flop (D)			Given Flip-flop (SR)	
Present State Q_n	Input D	Next State Q_{n+1}	Excitation S	Inputs R
0	0	0	0	X
0	1	1	1	0
1	0	0	0	1
1	1	1	X	0

Expression for S



$$S = D$$

Expression for R



$$R = \overline{D}$$

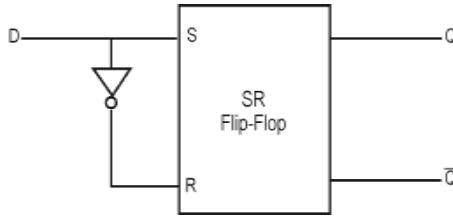


Fig. 3.17: D flip-flop using SR flipflop

3.12.2 Realization of J-K flipflop using SR flipflop

(April 2004)

- Write the characteristic table for JK flipflop.
- Write the excitation table for SR flipflop.
- Determine the expression for inputs S and R by using K-map.
- Draw the flip-flop conversion diagram.

TABLE 3.19 : Excitation table for JK flipflop realization using SR flipflop

Required flipflop (JK)				Given flipflop (SR) Excitation Inputs	
Q_n	J	K	Q_{n+1}	S	R
0	0	0	0	0	X
0	0	1	0	0	X
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	1	X	0
1	0	1	0	0	1
1	1	0	1	X	0
1	1	1	0	0	1

Expression for S Expression for R

Q_n	JK	00	01	11	10
0		0	0	(1)	(1)
1		X	0	0	X

$S = J\bar{Q}_n$

Q_n	JK	00	01	11	10
0		X	X	0	0
1		0	(1)	(1)	0

$R = KQ_n$

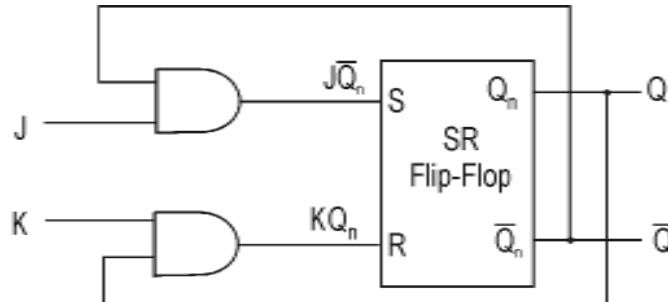


Fig. 3.18 : J-K flipflop using SR flipflop

3.12.3 Realization of T flipflop using D flipflop

- Write the characteristic table for T flipflop.
- Write the excitation table for D flipflop.
- Determine the expression for input D using K-map.
- Draw the flipflop conversion logic diagram.

TABLE 3.20: Excitation table for realization of T flipflop using D flipflop

Required flipflop (T)			Given flipflop (D)
Q_n	T	Q_{n+1}	Input (D)
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Expression for D

Q_n	0	1
0	0	(1)
1	(1)	0

$$D = \bar{T}Q_n + T\bar{Q}_n$$

$$= T \oplus Q_n$$

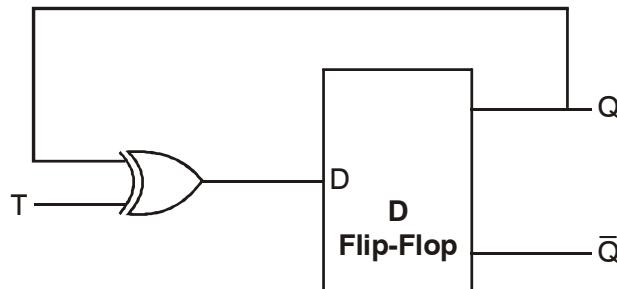


Fig. 3.19: T flip flop using D flip flop

3.12.4 Realization of D flip flop using JK flip flop

- Write the characteristic table for D flip flop.
- Write the excitation table for JK flip flop.
- Determine the expression of inputs J and K using K-map
- Draw the flip flop conversion logic diagram.

Table 3.21: Excitation table for realization of D flip flop using JK flip flop

Required flip flop (D)			Given flip flop (JK)	
Q_n	D	Q_{n+1}	J	K
0	0	0	0	X
0	1	1	1	X
1	0	0	X	1
1	1	1	X	0

Expression for J

Expression for K

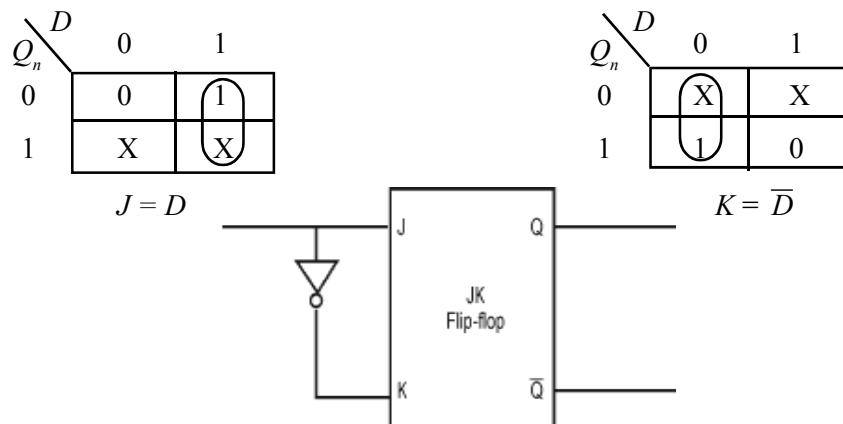


Fig. 3.20: D flip flop using JK flip flop

3.13 CLASSIFICATION OF SYNCHRONOUS SEQUENTIAL CIRCUITS

In synchronous or clocked sequential circuits, clocked flip flops are used as memory elements, which change their individual states in synchronism with the periodic clock signal. Therefore, the change in states of flip flops and change in state of the entire circuits occurs at the transition of the clock signal. The synchronous sequential circuits are represented by two models:

- ◆ Moore Model
- ◆ Mealy Model

3.13.1 Moore Model

In the Moore Model, the outputs are a function of the present state of the flip flops only.

The block diagram of Moore model is shown in **Figure 3.21**. The output depends only on present state of flip flops, it appears only after the clock pulse is applied, ie., it varies in synchronism with the clock input.

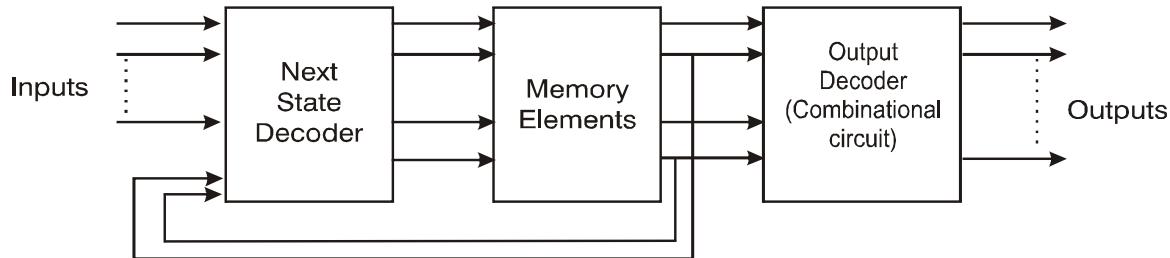


Fig. 3.21: Moore Model

3.13.2 Mealy Model

In the Mealy model, the outputs are functions of both the present state of the flip flops and inputs. The block diagram of Mealy model is shown in **Figure 3.22**.

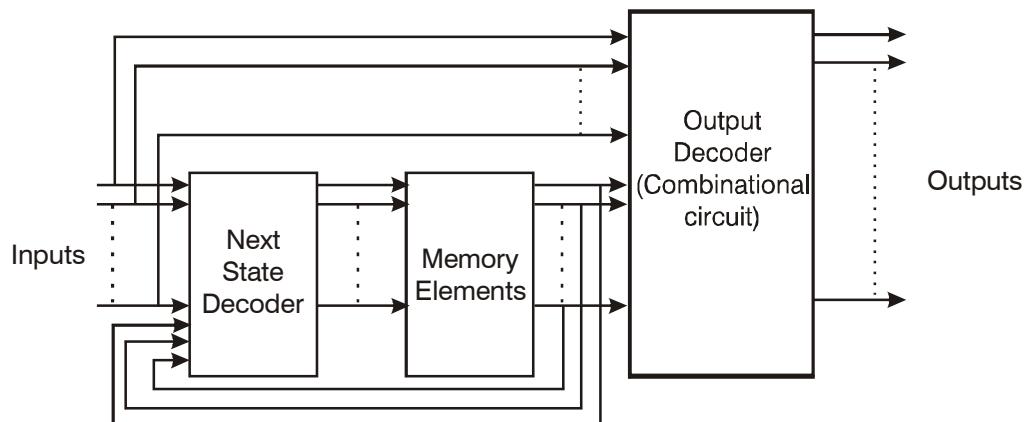


Fig. 3.22 : Block diagram of Mealy Model

3.13.3 Difference between Moore and Mealy Model

Sl.No.	Moore Model	Mealy Model
1.	Its output is a function of present state only.	Its output is a function of present state and present input.
2.	It requires more number of states for implementing same function	It requires less number of states for implementing same function.
3.	Input changes do not affect the output of the circuit	Input changes may affect the output of the circuit.

3.14 STATE EQUATION

It is an algebraic expression that specifies the condition for a flip flop state transition.

Some state equation examples are given below:

$$A(t+1) = Ax + Bx$$

$$B(t+1) = \bar{A}x$$

$$JA = B + \bar{x}$$

$$KA = \bar{B}C + BC\bar{x}$$

3.15 STATE TABLE

The time sequence of inputs, outputs and flip flop states can be enumerated in a state table.

State table consists of 4 sections: Present state, Next state, Input, Output.

The present state section shows the states of flipflops at any given time ' t '. The next state section shows the states of the flip flops one clock period later at time ($t+1$). The input section gives the value of inputs for each possible present state. The output section gives the value of outputs for each present state.

3.16 STATE DIAGRAM

State diagram is a graphical representation of a state table.

In the state diagram, a state is represented by a circle and the transition between states is indicated by directed lines connecting the circles. There is no difference between a state table and a state diagram except in the manner of representation.

3.17 ANALYSIS OF SYNCHRONOUS SEQUENTIAL CIRCUIT

The behaviour of a sequential circuit is determined from the inputs, outputs and the state of its flip flops. The outputs and the next state are both a function of the inputs and the present state. The analysis of a sequential circuit consists of obtaining a table or a diagram for the time sequence of inputs, outputs and internal states. It is also possible to write Boolean expressions that describe the behaviour of the sequential circuit. A logic diagram is recognized as a synchronous (clocked) sequential circuit if it includes flip-flops. The flip flops may be of any type and the logic diagram may or may not include combinational circuit gates.

3.17.1 Analysis of Mealy Model

Example 3.1

A sequential circuit with two ‘D’ flip flops A and B, one input (x); and one output (y). The flip flop input functions are:

$$DA = Ax + Bx$$

$$DB = \bar{A}x$$

and the circuit output function is,

$$y = (A + B)\bar{x}$$

- (a) Draw the logic diagram of the circuit
- (b) Tabulate the state table
- (c) Draw the state diagram

Solution

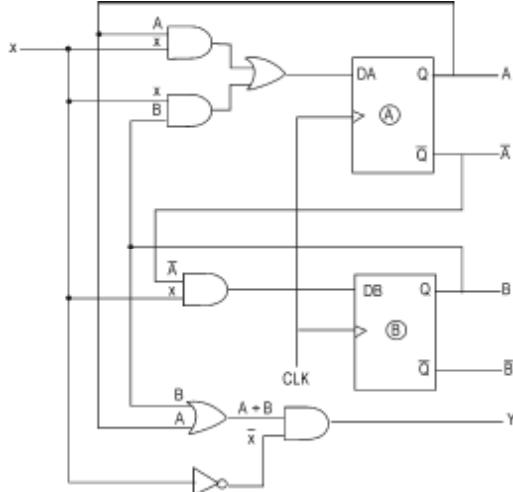


Fig. 3.23 : Mealy circuit

Figure 3.23 shows the Mealy synchronous sequential circuit for the given input and output functions. It consists of two D flip flops A and B, an input x and an output y.

D inputs determine the flipflop's next state; $D = Q(t+1)$.

Therefore the next-state equations are:

$$A(t+1) = A(t)x(t) + B(t)x(t)$$

$$B(t+1) = B(t+1) = \bar{A}(t)x(t)$$

The LHS of equation denotes the next state of the flip flop and the RHS of equation is a Boolean expression that specifies the present state and input conditions that make the next state equal to 1.

State Table

The state table of the circuit of the circuit is obtained by the following procedure

- A circuit with ‘m’ flip flops and ‘n’ inputs needs 2^{m+n} rows in the state table. In this example, $m = 2$, $n = 1$ and $2^{m+n} = 2^3 = 8$ rows are needed. Eight binary combinations from 000 to 111 are listed under the present state and input columns.
- The next state section has $m(2)$ columns, one for each flip flop. The binary values for the next state are derived directly from the state equations. The next state of flip flop A must satisfy the state equation, $A(t+1) = Ax + Bx$

The next state of flip flop B must satisfy the state equation, $B(t+1) = \bar{A}x$

- The output section has as many columns as there are output variables. This example has one output (y). Therefore one column is needed. Its binary value is derived from the circuit or from the output equation, $y = (A+B)\bar{x}$

The state table is given in **Table 3.22**.

Table 3.22: State Table

Present State		Input	Next State		Output
A	B	x	$A(t+1) = Ax + Bx$	$B(t+1) = \bar{A}x$	$Y = (A+B)\bar{x}$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

The above state is slightly changed with only three sections: present state, next state and output. The input conditions are enumerated under the next state and output sections. The state table of **Table 3.22** is repeated in **Table 3.23** using the second form.

Table 3.23: Second form of the state table

Present state	Next state				Output	
	$x = 0$		$x = 1$			
	A	B	A	B		
0 0	0 0		0 1		0	
0 1	0 0		1 1		1	
1 0	0 0		1 0		1	
1 1	0 0		1 0		1	

State Diagram

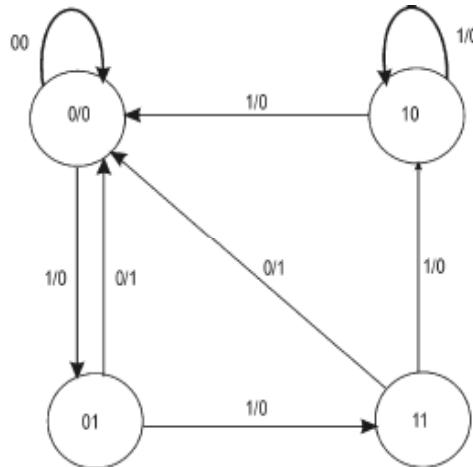


Fig. 3.24 : State diagram

The state diagram provides the same information as the state table and is obtained directly from table 3.23. The binary number inside each circle identifies the state of the flip-flops. The directed lines are labelled with two binary numbers separated by a slash. The input value during the present state is labelled first and the number after slash gives the output during the present state. For example, the directed line from state 00 to 01 is labelled 1/0, meaning that when the sequential circuit is in the present state 00 and the input is 1, the output is 0. A directed line connecting a circle with itself indicates that no change of state occurs. The state diagram is shown in **Figure 3.24**.

Example 3.2

Analyze the synchronous Mealy machine in **Figure 3.25** to obtain its state diagram.

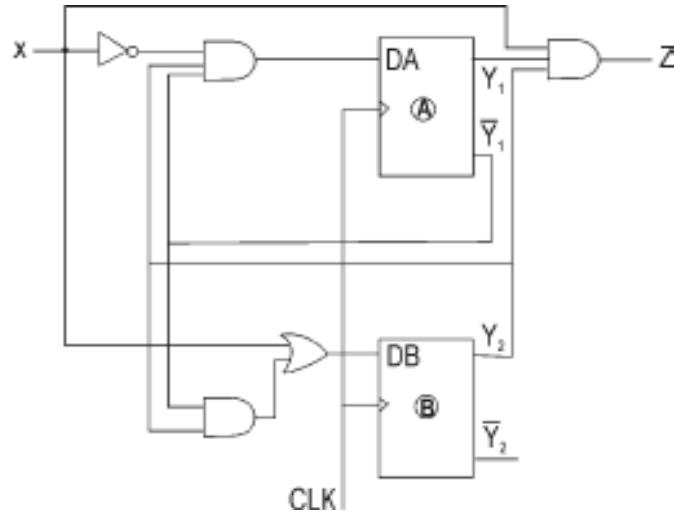


Fig. 3.25 : Mealy Model

Solution

The given synchronous Mealy machine consists of two D flip flops, one input and one output.

The flip flop input functions are,

$$DA = \bar{Y}_1 Y_2 \bar{x}$$

$$DB = X + Y_1 \bar{Y}_2$$

The circuit output function is,

$$Z = Y_1 \cdot Y_2 \cdot X$$

State Equations

The next state equation for a D flip flop is,

$$D = Q(t+1)$$

Therefore the next-state equations are

$$Y_1(t+1) = Y_1 Y_2 \bar{x}$$

$$Y_2(t+1) = x + Y_1 Y_2$$

State Table

Table 3.24 : State Table

Present state		Input	Next State		Output
y_1	y_2	x	$Y_1(t+1) = \bar{Y}_1 Y_2 \bar{X}$	$Y_2(t+1) = x + \bar{y}_1 y_2$	$Z = Y_1 Y_2 x$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	1	1

Table 3.25 : Second form of the state table

Present state		Next state				Output	
		$x = 0$		$x = 1$		$x = 0$	$x = 1$
Y_1	Y_2	Y_1	Y_2	Y_1	Y_2	Z	Z
0	0	0	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	0
1	1	0	0	0	1	0	1

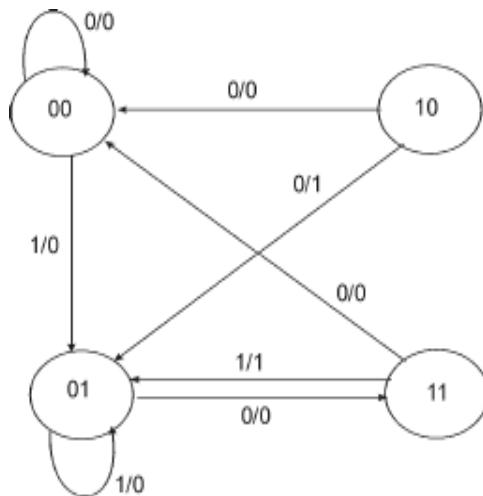
State Diagram

Fig. 3.26 : State Diagram

3.17.2 Analysis with JK and other flip flops

The next state values of a sequential circuit with D flip flop can be derived directly from the next-state equations. When other types of flip flops are used, it is necessary to refer the characteristics table. The next-state values of a sequential circuit that use any other type of flip flop such as JK, SR or T can be derived by following procedure.

1. Obtain the binary values of each flip flop input function in terms of the present state and input variables.
2. Use the corresponding flip-flop characteristic table to determine the next state. (*Refer Examples 3.3 to 3.5*)

3.17.3 Analysis of Moore Model

Example 4.3: A sequential circuit has two JK flip flop A and B. The flip flop input functions are:

$$JA = B$$

$$JB = \bar{x}$$

$$KA = B\bar{x}$$

$$KB = A \oplus x$$

- (a) Draw the logic diagram of the circuit
- (b) Tabulate the state table
- (c) Draw the state diagram

Solution: The output function is not given in the problem. The output of the flip flops may be considered as the output of the circuit. The logic diagram is shown in **Figure 3.27** for the given state equations.

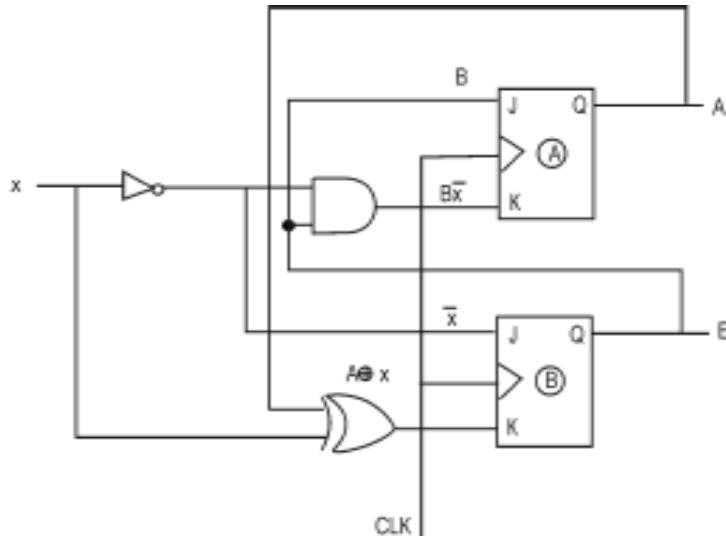


Fig. 3.27 : Sequential circuit with JK flip-flops

State Table

To obtain the next-state values of a sequential circuit with JK flip flops, use the JK flip flop characteristic table, given in **Table 3.26**.

Table 3.26 : Characteristic Table for J-K flip flop

Q	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Here, $Q = A$ or B and $Q_{n+1} = A(t+1)$ or $B(t+1)$

Table 3.27 : State Table

Present state	Input	Flip flop Inputs				Next state				
		A	B	x	$JA = B$	$KA = B\bar{x}$	$JB = \bar{x}$	$KB = A \oplus x$	$A(t+1)$	$B(t+1)$
0	0	0	0	0	0	1	0	0	0	1
0	0	1	0	0	0	0	1	1	0	0
0	1	0	1	1	1	1	0	1	1	1
0	1	1	1	0	0	0	1	1	1	0
1	0	0	0	0	0	1	1	1	1	1
1	0	1	0	0	0	0	0	1	1	0
1	1	0	1	1	1	1	1	0	0	0
1	1	1	1	0	0	0	0	1	1	1

Table 3.28 : Second form of the State table

Present State	Next State				
	$x = 0$		$x = 1$		
A	B	A	B	A	B
0	0	0	1	0	0
0	1	1	1	1	0
1	0	1	1	1	0
1	1	0	0	1	1

State Diagram

The state diagram of the Moore Model sequential circuit is shown in **Figure 3.28**. Since the circuit has no outputs, the directed lines out of the circles are marked with one binary number only to designate the value of input x_1

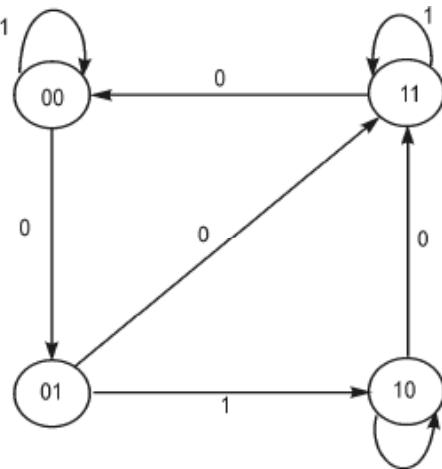


Fig. 3.28 : State Diagram

Example 4.4

Derive the state table and state diagram for the given sequential circuit, **Figure 3.29**.

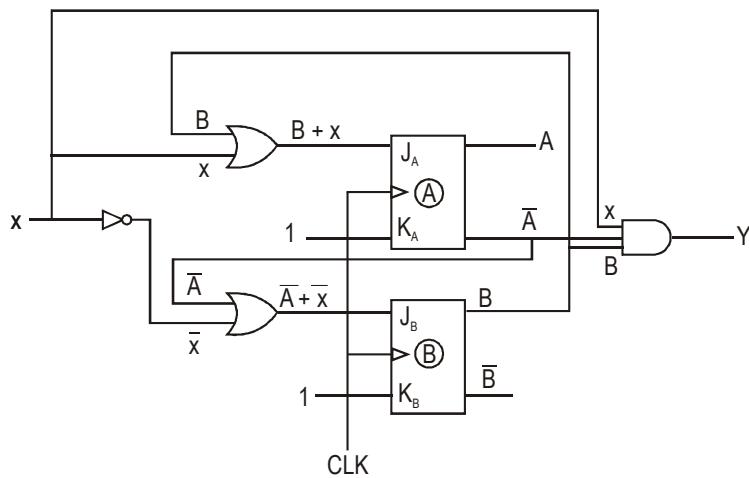


Fig. 3.29 : Sequential Circuit

Solution: Input functions:

$$JA = B + x$$

$$KA = 1$$

$$JB = \bar{A} + \bar{x}$$

$$KB = 1$$

Output equation : $Y = x \bar{A} B$

State Table: To obtain the next-state values of a sequential circuit with J-K flip flops, use the JK flip flop characteristics table given in **Table 3.29**.

Table 3.29: State Table

Present Input State			Flip Flop Inputs				Next state		Output
A	B	x	$JA = B + x$	$KA = 1$	$JB = \bar{A} + \bar{x}$	$KB = 1$	$A(t+1)$	$B(t+1)$	$Y = x \bar{A}B$
0	0	0	0	1	1	1	0	1	0
0	0	1	1	1	1	1	1	1	0
0	1	0	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	1	1	0	1	0
1	0	1	1	1	0	1	0	0	0
1	1	0	1	1	1	1	0	0	0
1	1	1	1	1	0	1	0	0	0

Table 3.30: Second form of State Table

Present state		Next state				Output	
		$x = 0$		$x = 1$		$x = 0$	$x = 1$
A	B	A	B	A	B	y	Y
0	0	0	1	1	1	0	0
0	1	1	0	1	0	0	1
1	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0

State diagram

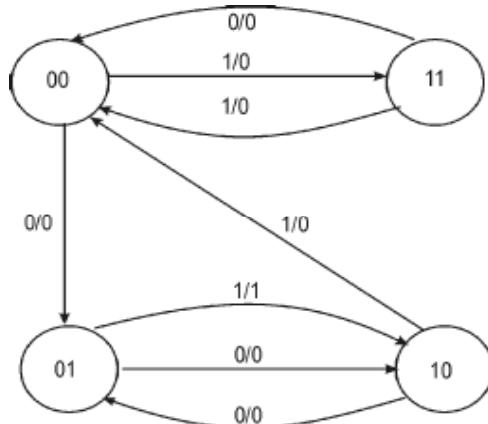


Fig. 3.30: State diagram

Example 3.5

Analyze the synchronous Moore machine shown in **Figure 3.31** to obtain its state diagram.

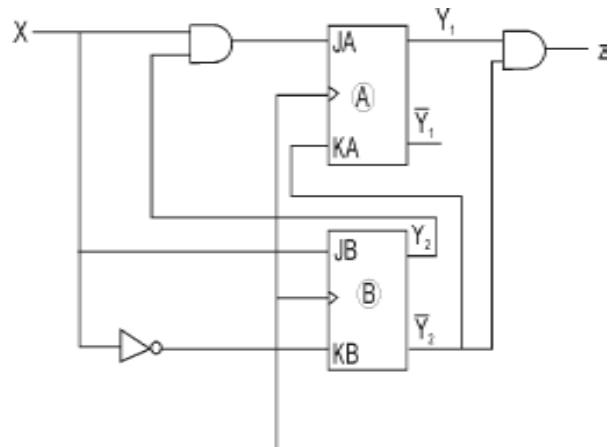


Fig. 3.31 : Moore Model

Solution: Using the assigned variable Y_1 and Y_2 for the two JK flip flops, we can write the four excitation input equations and the single Moore output equation as follows:

$$\begin{aligned} JA &= Y_2 x \quad ; \quad KA = \bar{Y}_2 \quad ; \quad JB = x \\ KB &= \bar{x} \quad ; \quad Z = y_1 \bar{y}_2 \end{aligned}$$

State Table:

To obtain the next-state values of a sequential circuit with J-K flip flops, use the JK flip flop characteristic table given in **Table 3.31**.

Table 3.31: State Table

Present State	Input	Flip Flop inputs				Next State		Output	
		y_1	y_2	x	$JA = Y_2 x$	$KA = \bar{y}_2$	$JB = x$	$KB = \bar{x}$	
0 0	0	0	0	0	1	0	0	1	0
0 0	1	0	0	1	1	1	0	0	0
0 1	0	0	1	0	0	0	1	0	0
0 1	1	0	1	1	0	1	0	1	0
1 0	0	1	0	0	1	0	1	0	1
1 0	1	1	0	1	1	1	0	0	1
1 1	0	1	1	0	0	0	0	1	0
1 1	1	1	1	1	0	0	0	1	1

Table 3.32 : Second form of state table

Present state		Next state		Output	
		$x = 0$	$x = 1$	$x = 0$	$x = 1$
y_1	y_2	y_1	y_2	Z	Z
0	0	0	0	0	0
0	1	0	0	0	0
1	0	0	0	1	1
1	1	1	0	1	1

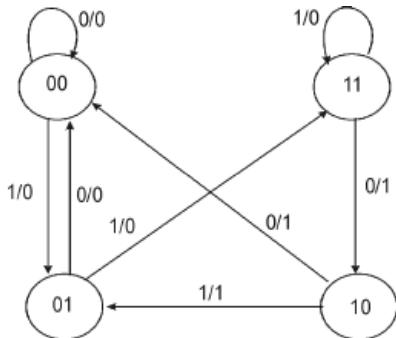


Fig. 3.32 : State diagram

3.17.4 Analysis Procedure

The synchronous sequential circuit analysis procedure is summarized as given below:

Step 1: Assign a state variable to each flip flop in the synchronous sequential logic circuit.

Step 2: Write the excitation input equations for each flip flop and also write the Moore and/or Mealy output equations.

Step 3: Substitute the excitation input equations into the bistable equations for the flip flops to obtain the next state output equations.

Step 4: Obtain the state table and reduced form of the state table.

Step 5: Draw the state diagram by using the second form of the state table.

3.18 STATE MINIMIZATION

The state reduction is used to avoid the redundant states in the sequential circuits. The reduction in redundant states reduce the number of required flip flops and logic gates, thus reducing the cost of the circuit.

The two states are said to be redundant or equivalent, if every possible set of inputs generate exactly same output and same next state. When two states are equivalent, one of them can be removed without altering the input-output relationship.

3.32

Digital Principles and System Design

Since ' n ' flip flops produced 2^n states, a reduction in the number of states may result in a reduction in the number of flip flops.

The need for state reduction or state minimization is explained with one example. In the given state diagram of **Figure 3.33**, only the input-output sequences are important; the internal states are used merely to provide the required sequences. For this reason, the states marked inside the circles are denoted by letter symbols (a, b, c, \dots) instead of by their binary values. The state table of the state diagram is shown in **Table 3.33**.

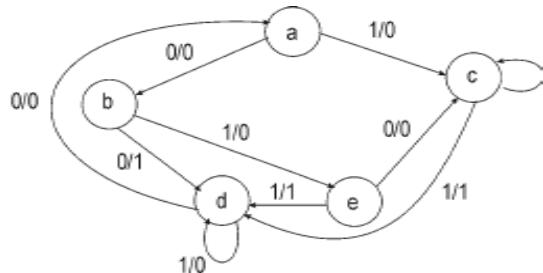


Fig. 3.33 : State Diagram

Table 3.33 : State Table

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	b	c	0	0
b	d	e	1	0
c	c	d	0	1
d	a	d	0	0
e	c	d	0	1

From the state **Table 3.33**, the states c and e generate exactly same next state and same output for every possible set of inputs. The state c and e go to next states c and d and have outputs of 0 and 1 for $x = 0$ and $x = 1$ respectively. Therefore state e can be removed and replaced by c . The final reduced state table is given in **Table 3.33**. The state diagram for the reduced state table consists of only four states and is shown in **Table 3.34**.

Table 3.34: Reduced state Table

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	b	c	0	0
b	d	c	1	0
c	c	d	0	1
d	a	d	0	0

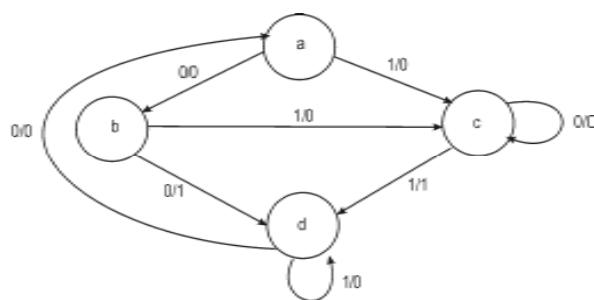


Fig. 3.34: Reduced State Table

Example 3.6

Reduce the number of states in the following state table and tabulate the reduced state table.

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	0
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

Solution

(i) g and e states go to the same next state (a, f) and have same output for both input combinations (0,1). Therefore states g and e are equivalent and one can be removed; g is replaced by e. **Table 3.35** shows the reduced state table 1.

Table 3.35: Reduced state table 1

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	e	f	0	1

(ii) Now states d and f are equivalent. Both states go to the same next state (e,f) and have same output (0,1). Therefore one state can be removed; f is replaced by d. The reduced state table-2 is shown in **Table 3.36**.

Table 3.36 : Reduced state Table 2

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

Thus 7 states are reduced into 5 states.

3.19 STATE ASSIGNMENT

State Assignment is a procedure to minimize the combinational gates in a circuit. State assignment procedure is assigning binary values to states in such a way as to reduce the cost of the combinational circuit that drives the flip-flops.

When a sequential circuit is viewed from its external input-output terminals, it may follow a sequence of internal states. But the binary values of the individual states (state assignment) may be of no consequence as long as the circuit produces the required sequence of outputs for any given sequence of inputs.

Two examples of possible binary assignments are shown in **Table 3.37** for the five states a,b,c,d and e. Assignment 1 is a straight binary assignment. Assignment 2 is chosen arbitrarily. Infact there are 140 different distinct assignment for these states.

Table 3.37 : Two possible Binary state Assignment

State	Assignment 1	Assignment 2
a	001	000
b	010	100
c	011	011
d	100	111
e	101	001

Binary Assignment 1 is substituted to **Table 3.37** for the letter symbols a,b,c,d and e. The binary form of the state table is used to derive the combinational circuit part of the sequential circuit. **Table 3.38** is the reduced state table with binary assignment 1.

Table 3.38 : Reduced state table with binary assignment 1

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
001	001	010	0	0
010	011	100	0	0
011	001	100	0	0
100	101	100	0	1
101	001	100	0	1

Rules for 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.

Figure 3.35 (a) shows the example for this rule. There are 4 states whose next state is same (000). Thus state assignments for these states are 100, 101, 110 and 111, which can be grouped into logically adjacent cells in a K-map.

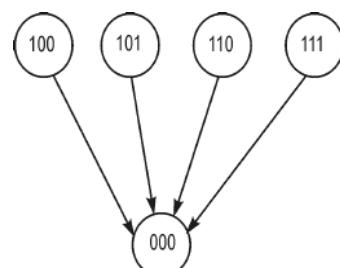


Fig. 3.35 (a) : Rule 1-Example

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.

Figure 3.35 (b) shows the example for rule 2. 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.

Rule 3: Adjacent assignments should be given to states that have the same outputs.

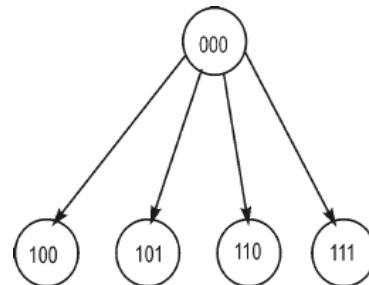


Fig. 3.35 (b) : Rule 2-Example

3.20 DESIGN OF SYNCHRONOUS SEQUENTIAL CIRCUITS

A synchronous sequential circuit is made up of number of flip flops and combinational gates. The design of the circuit consists of choosing the flip flops and then finding a combinational gate structure that together with the flip flops. The number of flip flops is determined from the number of states needed in the circuit. The combinational circuit is derived from the state table.

Design procedure

1. The given problem is determined with a state diagram
2. From the state diagram, obtain the state table.
3. The number of states may be reduced by state reduction methods, if applicable.
4. Assign binary values to each state (Binary Assignment) if the state table contains letter symbols.
5. Determine the number of flip flops needed and assign a letter symbol (A,B,C ...) to each.
6. Choose the type of flip flop (D,T,JK, SR) to be used.
7. From the state table, derive the circuit excitation and output tables.
8. Using K-map or any other simplification method, derive the circuit output functions and the flip-flop input functions.
9. Draw the logic diagram.

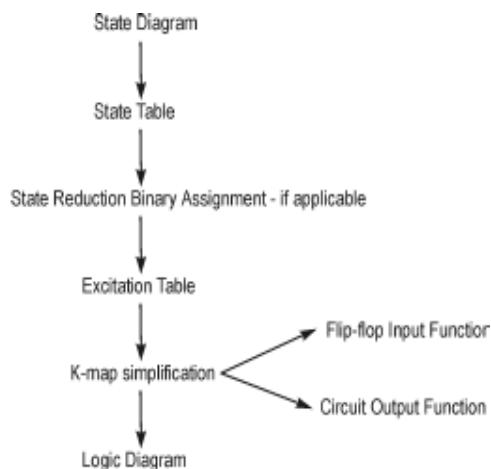


Fig. 3.36 : Design Procedure

The type of flip flop to be used may be included in the design specifications or may depend on what is available to the designer. Many digital systems are constructed with JK flip flops because they are the most versatile available. The selection of flip flops is given as follows:

Flip Flop	Application
JK	General Applications
D	Applications requiring transfer of data (Ex: Shift Registers)
T	Applications involving complementation (Ex: Binary Counters)

3.20.1 DESIGN WITH J-K FLIP FLOPS

Example 3.7

Design the synchronous sequential circuit specified by the state diagram of **Figure 3.37** using J-K flip flops.

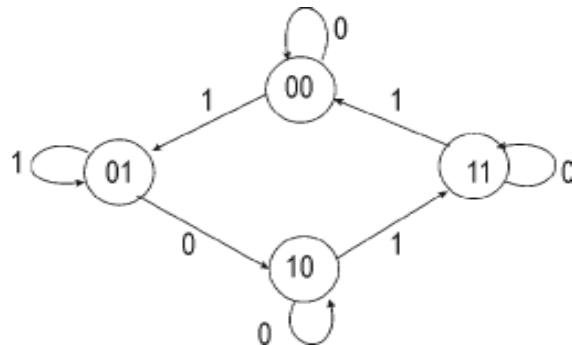


Fig. 3.37 : State Diagram

Solution

The state diagram of **Table 3.39** consists of 4 states with binary values already assigned. Since the directed lines are marked with a single binary digit without a slash, there is one input variable and no output variables. To represent these four states, two flip flops are needed and designated as A and B. The input variable is designated as x. **Table 3.39** shows the state table for the state diagram.

Table 3.39 : State Table

Present State		Next state			
		x = 0		x = 1	
A	B	A	B	A	B
0	0	0	0	0	1
0	1	1	0	0	1
1	0	1	0	1	1
1	1	1	1	0	0

The present state and input variables are arranged in the form of a truth table in table 3.43. We have already discussed about the excitation tables in section 3.11. Again the excitation table for JK flip flop is given in **Table 3.40**. Using the excitation input values, fill the values of J and K inputs of flip flop A(JA and KA) and flip flop B (JB and KB) in the **Table 3.41**.

Table 3.40: Excitation Table for JK flip flop

Present State		Next state		Inputs	
$Q(t)$		$Q(t+1)$		J	K
0		0		0	X
0		1		1	X
1		0		X	1
1		1		X	0

Table 3.41: Excitation Table

Inputs of combinational circuit			Next State	Outputs of combinational circuit				
Present State	Input			Flip Flop Inputs				
A	B	X	$A(t+1)$	$B(t+1)$	JA	KA	JB	KB
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

Flip Flop Input Functions Equations

The flip flop input functions (JA , KA , JB , KB) are derived from excitation table by using K-map as follows:

Expression for JA

A	Bx	00	01	11	10
0					$\textcircled{1}$
1		X,	X	X	\textcircled{X}

$$JA = B\bar{x}$$

Expression for KA

A	Bx	00	01	11	10
0		X	X	\textcircled{X}	X
1				$\textcircled{1}$	

$$KA = Bx$$

Expression for JB

Bx	00	01	11	10
0		1	X	X
1		1	X	X

$$JB = x$$

Expression for KB

Bx	00	01	11	10
0	X			1
1	X	(X)		

$$KB = Ax + \bar{A}\bar{x} = \bar{A} \oplus x$$

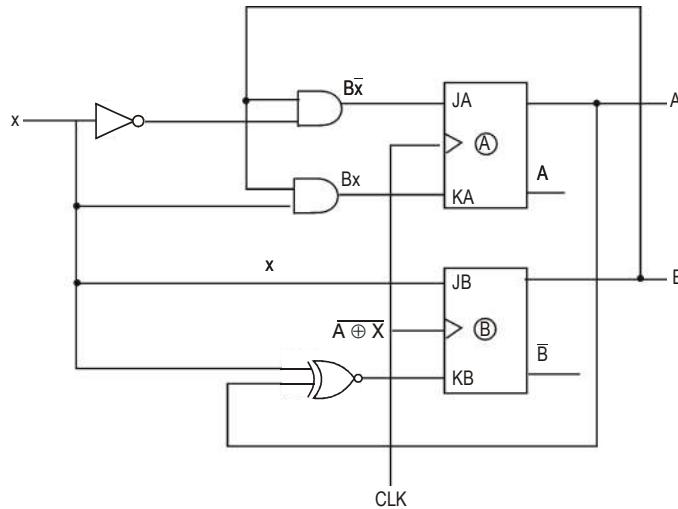
Logic diagram

Fig. 3.38: Logic diagram with JK flipflop

3.20.2 DESIGN WITH D FLIP FLOPS**Example 3.8**

Design the synchronous (clocked) sequential circuit for Mealy state diagram of **Figure 3.39** using D flip flops.

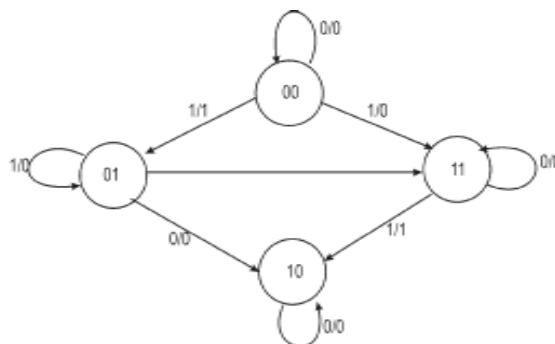


Fig. 3.39 : Mealy State Diagram

Solution**Step 1: State Table**

Table 3.42: State Table

Present State		Next State		Output	
		$x = 0$	$x = 1$	$x = 0$	$x = 1$
A	B	A	B	Y	y
0	0	0	0	0	1
0	1	1	0	0	0
1	0	1	0	0	1
1	1	1	1	0	0

Step 2: Excitation table for design with D flip flops:

Table 3.43 : Excitation table for D flip flop

Present State	Next State	Input
$Q(t)$	$Q(t+1)$	D
0	0	0
0	1	1
1	0	0
1	1	1

Table 3.44: Excitation table for design with D flip flops

Present State	Next State		Flip Flop Inputs	Output	
A	B	x	$A(t+1) \ B(t+1)$	$DA \ DB$	Y
0	0	0	0 0	0 0	0
0	0	1	0 1	0 1	1
0	1	0	1 0	1 0	0
0	1	1	0 1	0 1	0
1	0	0	1 0	1 0	0
1	0	1	1 1	1 1	1
1	1	0	1 1	1 1	0
1	1	1	0 0	0 0	0

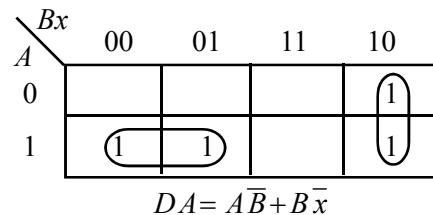
$$D = Q(t+1)$$

Therefore, $DA = A(t+1)$

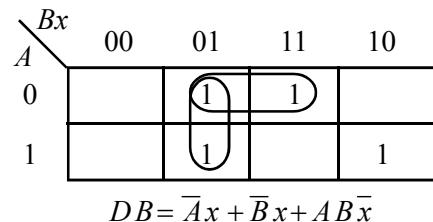
$$DB = B(t+1)$$

Step 3: K map simplification for flip flop input functions and circuit output function

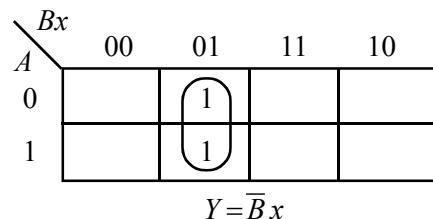
Expression for DA



Expression for DB



Expression for y



Step 4: Logic Diagram

The simplified functions are:

$$DA = A\bar{B} + B\bar{x}$$

$$DB = \bar{A}x + \bar{B}x + AB\bar{x}$$

$$Y = \bar{B}x$$

The logic diagram of sequential circuit with D flip flops is shown in **Figure 3.40**

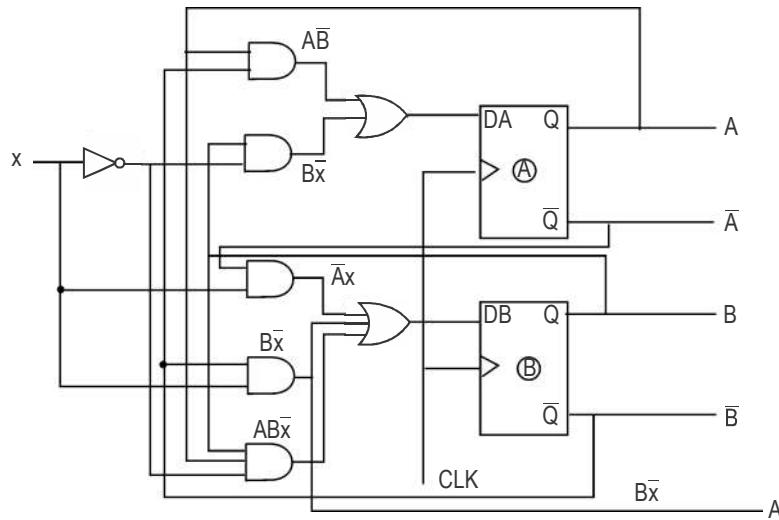


Fig. 3.40 : Logic diagram with D flip flops

3.20.3 DESIGN WITH T FLIP FLOPS

Example 3.9

Design the synchronous sequential circuit for the Moore state diagram of **Figure 3.41** using T flip flops.

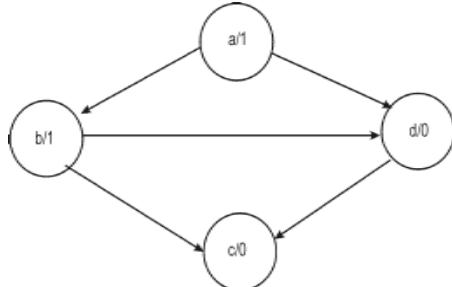


Fig. 3.41 : Moore state diagram

Solution

Step 1: State Table: The state table for the given Moore state diagram is shown in **Table 3.47**.

Table 3.45: State Table

Present State	Next State	Output
a	b	1
b	c	1
c	d	0
d	a	0

Step 2: Binary Assignment

Assign the binary values 00, 01, 11, 10 to the states a, b, c, d respectively. **Table 3.46** shows the state table with binary assignment.

Table 3.46 : State Table with Binary assignment

Present State		Next State		Output
A	B	A(t+1)	B(t+1)	Y
0	0	0	1	1
0	1	1	1	1
1	1	1	0	0
1	0	0	0	0

Step 3: Excitation table for design with T flip flops.

Table 3.47 shows the excitation table for T flip flop and **Table 3.48** shows the excitation table for given problem.

Table 3.47 : Excitation table for T flip flop

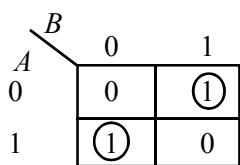
$Q(T)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

Table 3.48 : Excitation Table

Present state		Next State		Inputs		Output
A	B	A(t+1)	B(t+1)	TA	TB	Y
0	0	0	1	0	1	1
0	1	1	1	1	0	1
1	1	1	0	0	1	0
1	0	0	0	1	0	0

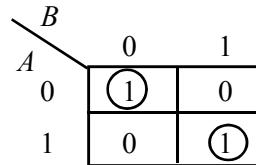
Step 4: K-map simplification for flip flop input functions (TA, TB) and output function (Y).

Expression for TA

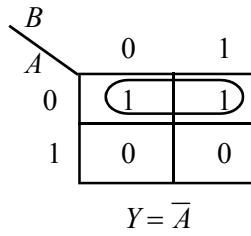


$$TA = \overline{A}B + A\overline{B} = A \oplus B$$

Expression for TB



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

Expression for Y

Step 5: Logic diagram of Moore sequential circuit with T flip flops.

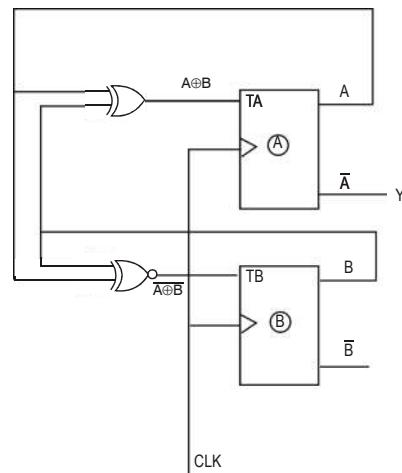


Fig. 3.42 : Moore sequential circuit with T flip flops

3.21 ALGORITHM STATE MACHINE (ASM) Chart

A special flow chart that has been developed specifically to define digital hardware algorithms is called an Algorithmic State Machine (ASM) chart. A state machine is another term for a sequential circuit, which is the basic structure of a digital system.

The ASM chart resembles a conventional flow chart, but is interpreted differently. A conventional flow chart describes the sequence of procedural steps and decision paths for an algorithm without concern for their time relationship. The ASM chart describes the sequence of event as well as timing relationship between the states of a sequential controller and the events that occur while going from one state to the next. It is specifically adapted to specify accurately the control sequence and data processing operations in a digital system, taking into consideration the constraints of digital hardware.

The ASM chart is a special type of flow chart suitable for describing the sequential operations in a digital system. The chart is composed of three basic elements:

1. a state symbol
2. an input condition symbol (decision symbol)
3. a conditional output symbol

3.21.1 State Symbol

The state symbol is a rectangle with an arrow entering the top and leaving the bottom. A state name is written to the side or on top of the box. Inside the box is written a list of unconditional output(s). If no unconditional output is required, the box interior is left blank. **Figure 3.43** illustrates the ASM state symbol.

3.21.2 Decision Symbol

The condition or decision symbol is a diamond or modified diamond. One arrow enters the top and two leave the sides or a side and the bottom. Written into the decision symbol are the logic conditions for the decision to be either true or false. The condition(s) can be a single variable or a Boolean expression. **Figure 3.44** shows the decision symbols.

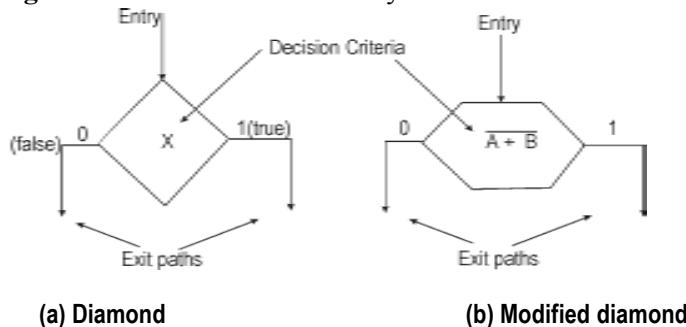


Fig. 3.44: Decision symbols

3.21.3 Conditional output symbol

It is a modified ellipse, as indicated in **Figure 3.45**. Output variables are listed internally to the symbol. There may be a single variable or multiple variables.

3.21.4 ASM Block

An ASM block consists of a state symbol, and/or possibly a state symbol with decision block(s) and/or conditional output symbol(s). The block symbols indicate a process involving a present state, the decisions, and conditional outputs and the next state.

The Moore sequential circuit model is represented without conditional output symbols and the Mealy model includes conditional output symbols.

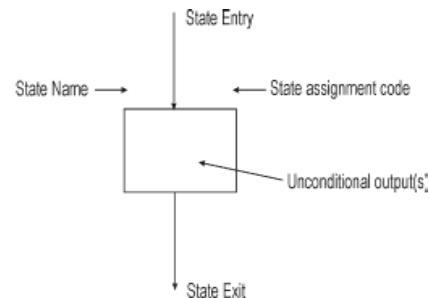


Fig. 3.43: State symbol

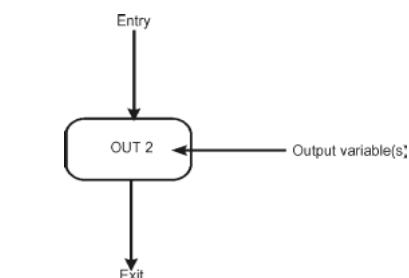


Fig. 3.45: Conditional output symbol

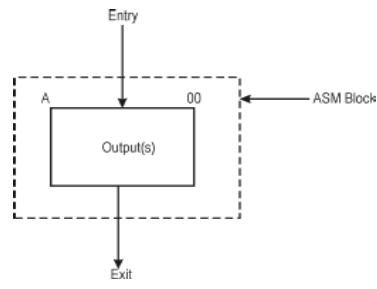


Fig. 3.46: Simple ASM Block

Figure 3.47(a) illustrates a Moore model with an unconditional output in State A. **Figure 3.47(b)** shows a Moore/Mealy mix with both unconditional (output variable OUT 1 in State A) and conditional (Output variable OUT 2) outputs.

Figure 3.47(b) shows state A, which consists of the state symbol, the decision symbol and the conditional output symbol. If input variables $X\bar{Y}=1$, then output OUT 2 is active until next state B is reached. If input variables $X\bar{Y}=0$, the conditional output is inactive and c is the next state. Output OUT 1 remains active as long as the machine remains in state A. The standard state diagram notation equivalent of the ASM diagram in **Figure 3.47** is shown in **Figure 3.48**

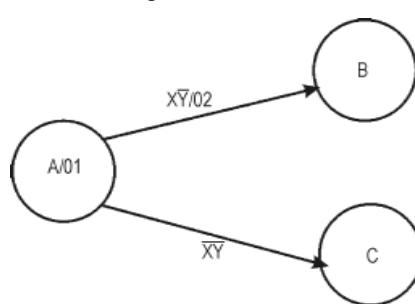
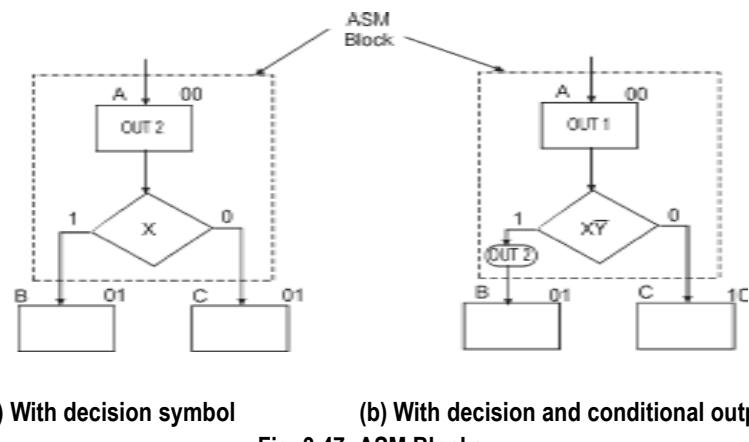


Fig. 3.48: Standard State diagram notation for the ASM block in Fig. 3.47(b)

Each block in the ASM chart describes the state of the system during one clock pulse interval. The operations within the state and conditional boxes in **Figure 3.49** are executed with a common clock pulse while the system is in state T_1 . The same clock pulse also transfers the system controller to one of the next states, T_2 , T_3 or T_4 as dictated by the binary values of E and F. The equivalent state diagram is shown in **Figure 3.50**.

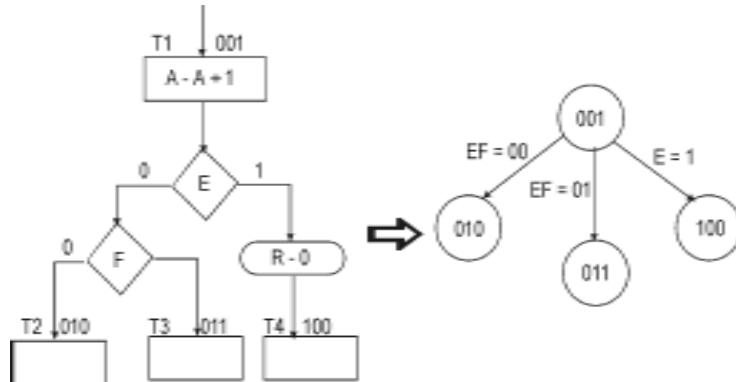


Fig. 3.49: ASM Block

Fig. 3.50: State Diagram equivalent

3.22 SHIFT REGISTERS

3.22.1 Types of Shift Registers

A register is simply a group of flip-flops that can be used to store a binary number. There must be one flip-flop for each bit in the binary number. For instance, a register used to store an 8-bit binary number must have eight flip-flops. Naturally the flip-flops must be connected such that the binary number can be entered (shifted) into the register and possibly shifted out. A group of flip-flops connected to provide either or both of these functions is called a shift register.

The bits in a binary number (data) can be removed from one place to another in either of two ways. The first method involves shifting the data 1 bit at a time in a serial fashion, beginning with either the most significant bit (MSB) or the least significant bit (LSB). This technique is referred to as serial shifting. The second method involves shifting all the data bits simultaneously and is referred to as parallel shifting.

There are two ways to shift data into a register (serial or parallel) and similarly two ways to shift the data out of the register. This leads to the construction of four basic register types as shown in **Figure 3.51** serial in- serial out, serial in-parallel out, parallel in-serial out, and parallel in- parallel out. All of these configuration are commercially available as TTL, MSI/LSI circuits. For instance

Serial in-serial out - 54/74LS91, 8 bits

Serial in-parallel out - 54/74164, 8 bits

Parallel in-serial out - 54/74165, 8 bits

Parallel in-parallel out - 54/74194, 4 bits

Parallel in-parallel out - 54/74188, 8 bits

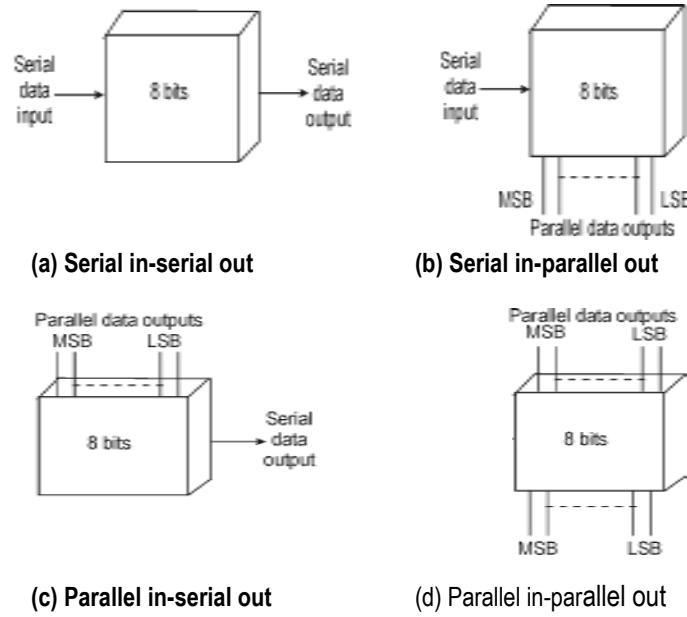


Fig. 3.51: Shift register types

Data shifting techniques and methods for constructing the four different types of registers are discussed in the following sections.

3.22.2 Serial-In Serial-Out Shift Register

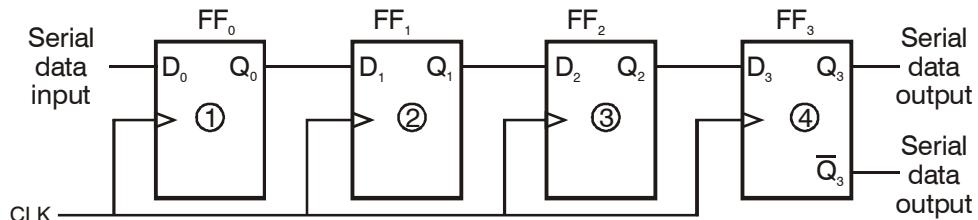


Fig. 3.52: Serial in/serial out shift register

Figure 3.52 illustrates entry of the four bits 1010 into the register, beginning with the right most bit. The register is initially clear. The 0 is put onto the data input line, making $D = 0$ for FF_0 . When the first clock pulse is applied, FF_0 is reset, thus storing the 0.

Next the second bit, which is a 1, is applied to the data input, making $D = 1$ for FF_0 and $D = 0$ for FF_1 because the D input of FF_1 is connected to the Q_0 output. When the second clock pulse occurs, the 1 on the data input is shifted into FF_0 , causing FF_0 to set; and the 0 that was in FF_0 is shifted into FF_1 .

The third bit, a 0 is now put onto the data-input line, and a clock pulse is applied. The 0 is entered into FF_0 , the 1 stored in FF_0 is shifted into FF_1 , and the 0 stored in FF_1 is shifted into FF_2 .

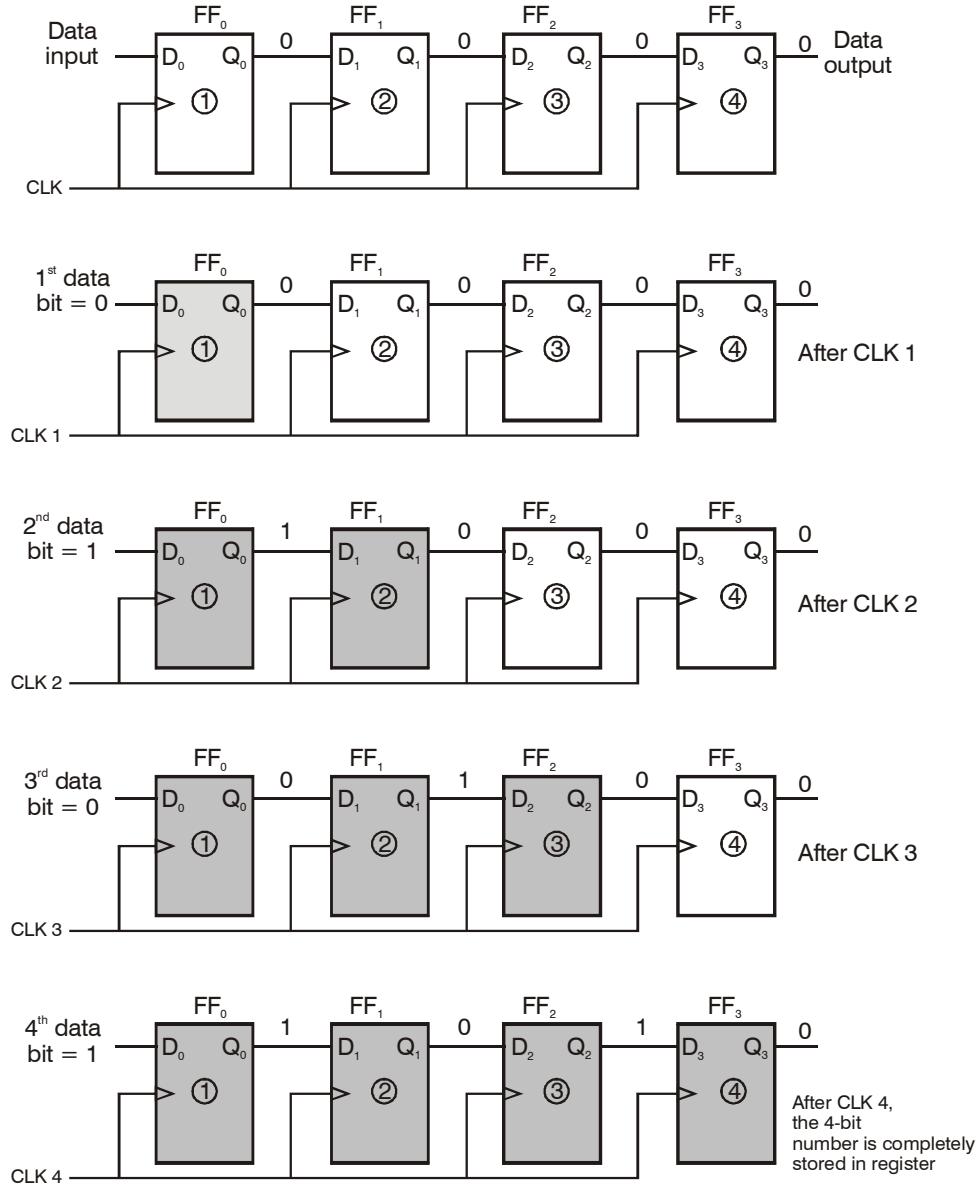


Fig. 3.53: Four bits (1010) being entered serially into the register

The last bit, a 1, is now applied to the data input, and a clock pulse is applied. This time the 1 is entered into FF_0 , the 0 stored in FF_0 is shifted into FF_1 , the 1 stored in FF_1 is shifted into FF_2 , and the 0 stored in FF_2 is shifted into FF_3 . This completes the serial entry of the four bits into the shift register, where they can be stored for any length of time as long as the flip-flops have dc power.

To get the data out of the register, the bits must be shifted out serially and taken off the Q_3 output as **Figure 3.54** illustrates. After $CLK4$ in the data-entry operations just described, the right-most bit 0, appears on the Q_3 output. When clock-pulse $CLK5$ is applied the second bit appears on the Q_3 output. Clock pulse $CLK6$ shifts the third bit to be output, and $CLK7$ shifts the fourth bit to the output. Notice that while the original four bits are being shifted out, more bits can be shifted in. All zeros are shown being shifted in.

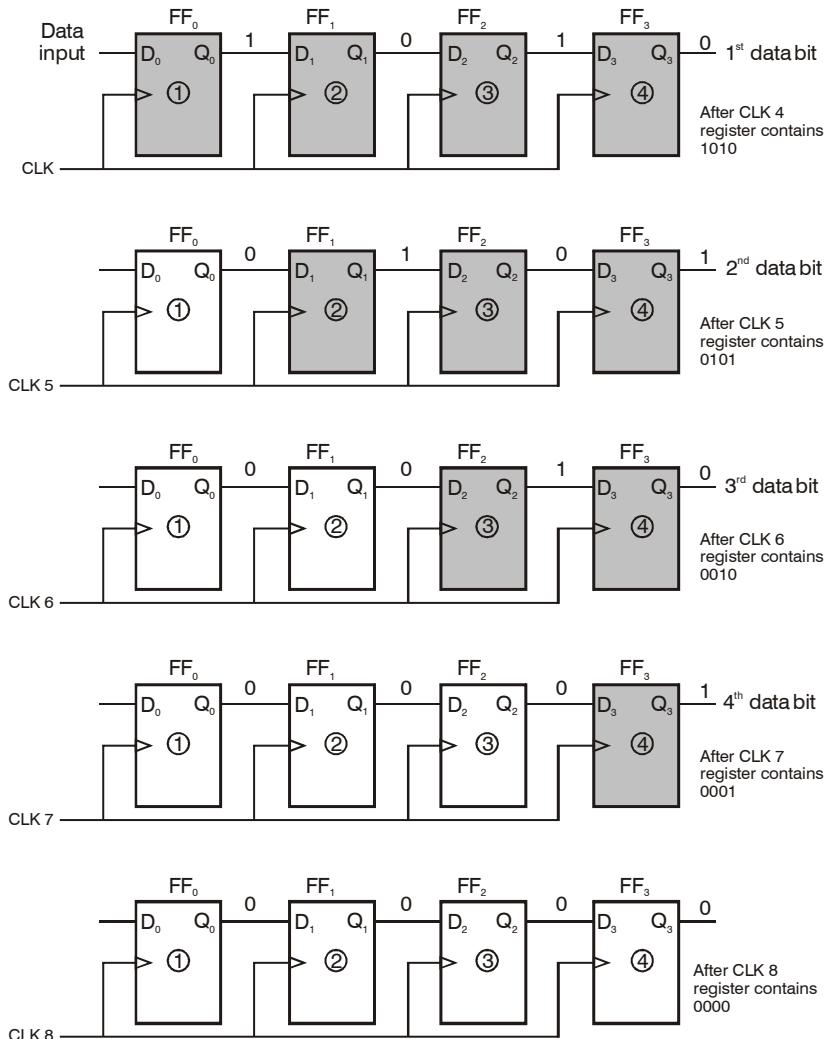


Fig. 3.54 : Four bits (1010) being serially-shifted out of the register and replaced by all zeros.

3.22.3 Serial In Parallel Out shift Register

In this shift register, data bits are entered into the register in the same as serial-in serial-out shift register. 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.

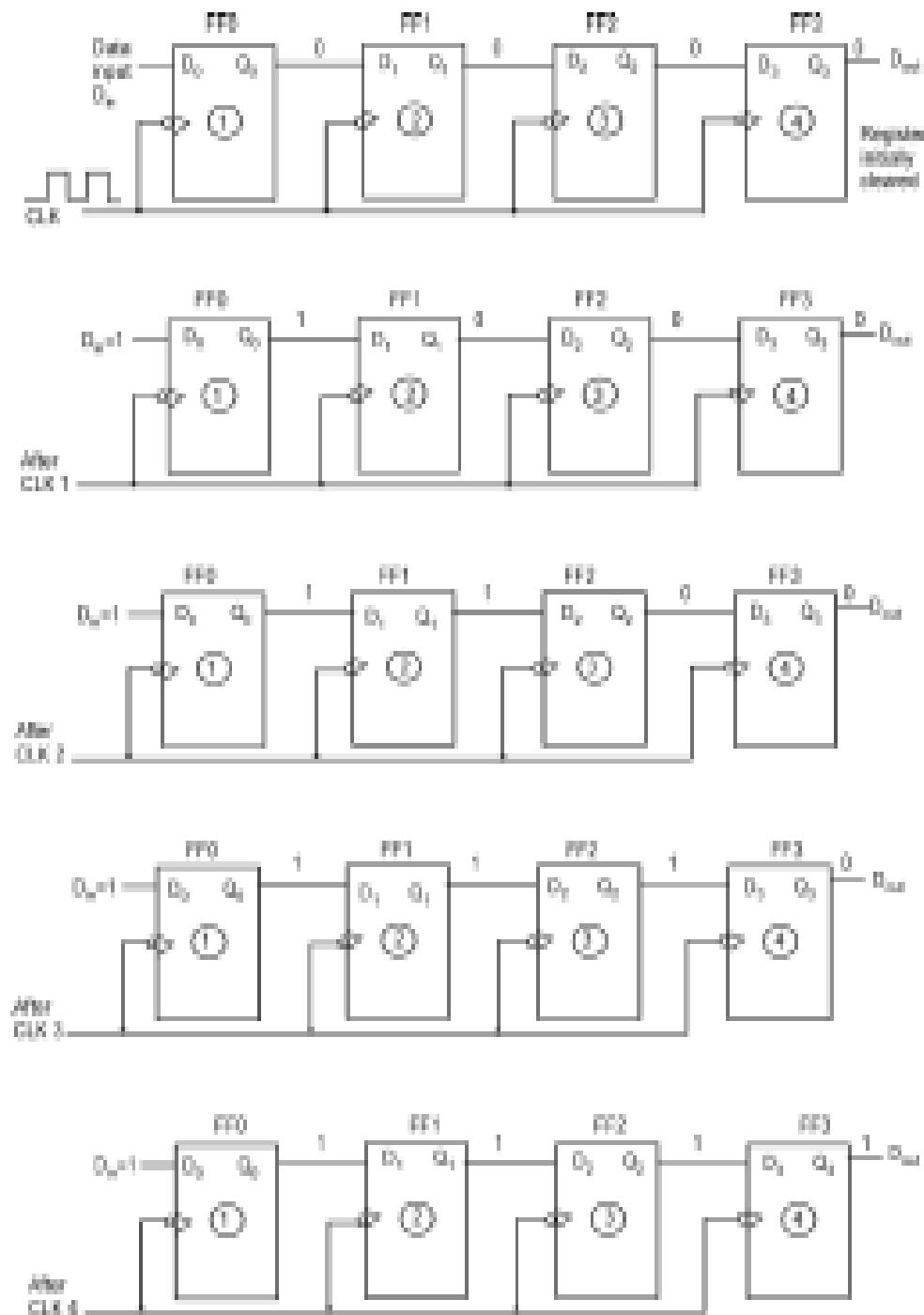
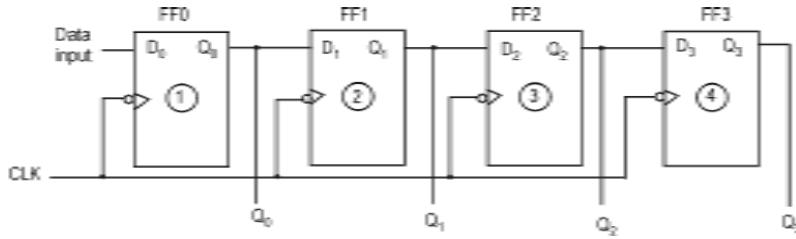


Fig. 3.55: Four Bits 1111 being serially entered into shift-right register



3.56: A serial-in parallel out shift register

3.22.4 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.

Figure 3.57 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/ \overline{LOAD} is the control input which allows shift or loading data operation of the register. When SHIFT/ \overline{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.

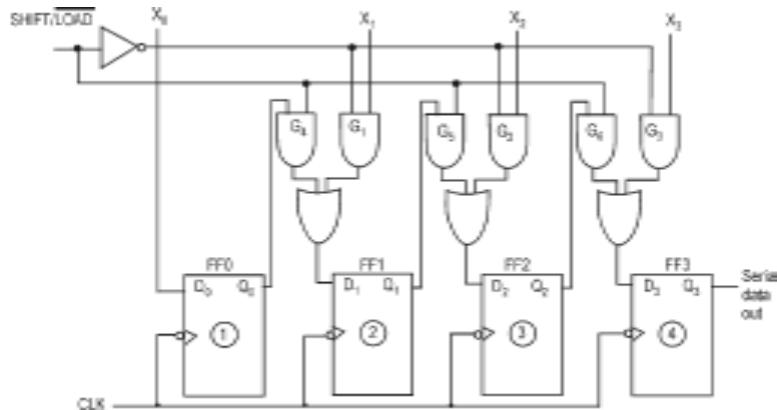


Fig. 3.57 : Parallel in serial out shift register

When SHIFT/ \overline{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/ \overline{LOAD} input.

3.22.5 Parallel in Parallel Out Register

From the third and second type of registers, it is cleared that how to enter the data in parallel i.e., all bits simultaneously into the register and how to take data out in parallel from the register. In

parallel in parallel out register, there is simultaneous entry of all data bits and the bits appear on parallel outputs simultaneously. **Figure 3.58** shows this type of register.

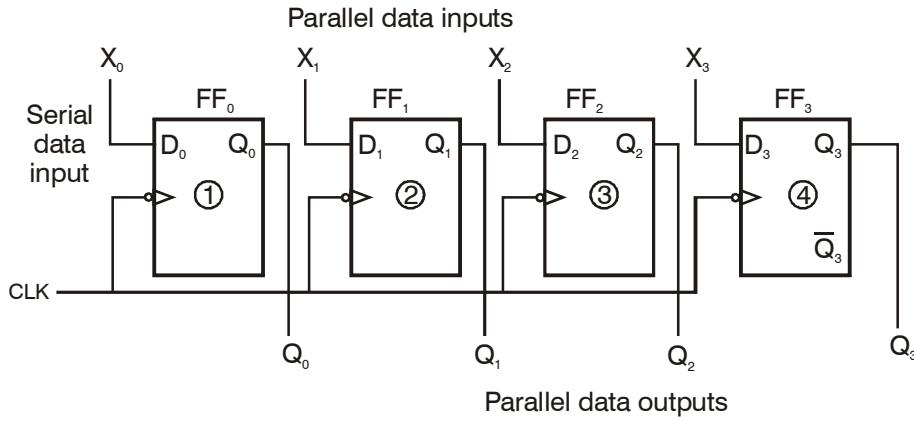


Fig. 3.58 : Parallel in parallel out shift register

3.22.6 Universal Shift Registers

A register which is capable of shifting data both to the right and left is called a bi-directional shift register. A register that can shift in only one direction is called a uni-directional shift register. If the register has shift and parallel load capabilities, then it is called a shift register with parallel load or Universal Shift Register.

Shift registers can be used for converting serial data to parallel data, and vice-versa. If a parallel load capability is added to a shift register, then data entered in parallel can be taken out in serial fashion by shifting the data stroed in the register.

The functions of shift register are:

1. A clear control to clear the register to 0.
2. A *CLK* input for clock pulses to synchronise all operations.
3. A shift-right control to enable the shift-right operation and serial input and output lines associated with the shift right.
4. A shift-left control to enable the shift-left operation and the serial input and output lines associated with the shift left.
5. A parallel load control to enable a parallel transfer and the *n* input lines associated with the parallel transfer.
6. *n* parallel output lines.
7. A control line that leaves the information in the register unchanged even though clock pulses are continuously applied.

Figure 3.59 illustrates a 4 bit universal shift register. It consists of four D flip-flops and four 4 input multiplexers (MUX). S_1 and S_0 be the two selection inputs connected to all the four multiplexers. These two selection inputs are used to select one of the four inputs of each multiplexer. Input 0 in each MUX is selected when $S_1 S_0 = 00$ and input 1 is selected when $S_1 S_0 = 01$. Similarly inputs 2 and 3 are selected when $S_1 S_0 = 10$ and $S_1 S_0 = 11$ respectively. The inputs S_1 and S_0 control the mode of operation of the register. When $S_1 S_0 = 00$, the present value of the register is applied to the D inputs of the flip-flops. This is done by connecting the output of each flip-flop to the 0 input of the respective multiplexer. The next clock pulse transfers into each flip-flop, the binary value it held previously, and hence no change of state occurs. When $S_1 S_0 = 01$, terminal 1 of the multiplexer inputs has a path to the D inputs of the flip-flops. This causes a shift-right operation with the left serial input transferred into flip-flop A_4 . When $S_1 S_0 = 10$, a shift-left operation results with the right serial input going into flip-flop A_1 . Finally when $S_1 S_0 = 11$ the binary information on the parallel input lines (I_1, I_2, I_3 , and I_4) are transferred into the register simultaneously during the next clock pulse. The function table of bi-directional shift register with parallel inputs and parallel outputs is shown in **Table 3.49**

Table 3.49 : Function table

Mode Control		Operation
S_1	S_0	
0	0	No change
0	1	Shift-right
1	0	Shift-left
1	1	Parallel load

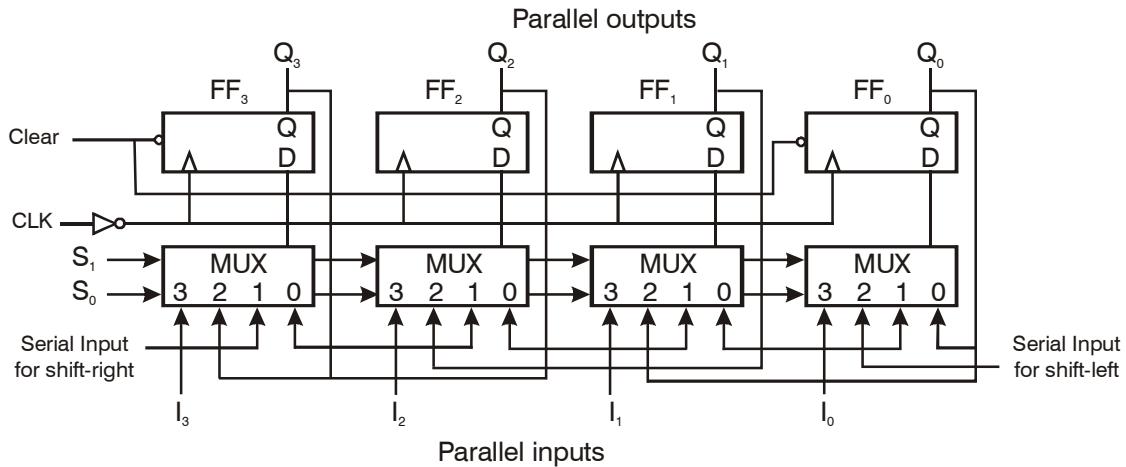


Fig. 3.59 : 4-bit Universal shift register

Bi-direction Shift Register: When mode control $M = 1$, the AND gates G_1 through G_4 are enabled and the data at D_8 is shifted to the right when the clock pulses are applied, and thus it acts as a shift-right register. When $M = 0$, the AND gates G_5 through G_8 are enabled allowing the data at D_1 to be shifted to the left, and thus it acts as a shift-left register. M should be changed only when $CLK = 0$, otherwise the data stored in the register may be altered. The 4-bit bi-directional shift register is shown in **Figure 3.60**.

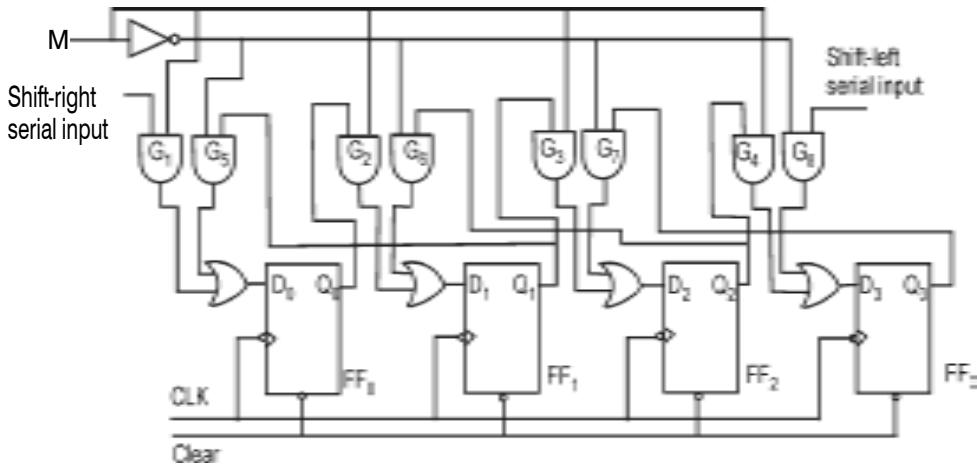


Fig. 3.60: 4-bit bi-directional shift register

3.23 COUNTERS

A counter is a sequential circuit consisting a set of flipflops to count the sequence of the input pulses presented to it in digital form. The number of flipflops used and the way in which they are connected determine the number of states (called the modulus) and also the specific sequence of states that the counter goes through during each complete cycle.

Counters are classified into two broad categories according to the way they are clocked:

- ♦ Asynchronous
- ♦ Synchronous

Asynchronous Counters: In asynchronous (Ripple) counters, the first flip-flop is clocked by the external clock pulse and then each successive pulse is clocked by the output of the preceding flipflop.

Synchronous Counters: In synchronous counters, the clock input is connected to all the flipflops so that they are clocked simultaneously.

ASYNCHRONOUS (RIPPLE) COUNTERS

3.23.1 2 Bit Asynchronous Binary Counter

In 2 bit asynchronous binary counter, the clock (CLK) is applied to the clock input of the first flipflop ($FF0$) only. The second flipflop, $FF1$, is triggered by the Q_0 output of $FF0$. Because of the inherent propagation delay time through a flipflop, a transition of the input clock pulse (CLK) and a transition of the Q_0 output of $FF0$ can never occur at exactly the same time. Therefore, the two flipflops are never simultaneously triggered, so the counter operation is asynchronous. **Figure 3.61** shows a 2 bit asynchronous binary counter.

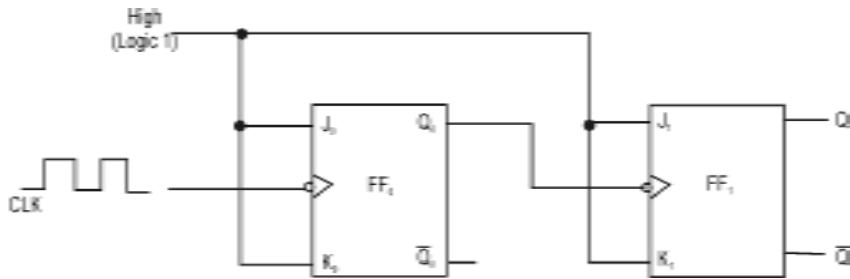


Fig. 3.61: 2 bit asynchronous binary counter

Figure 3.62 shows the timing diagram for 2 bit ripple counter. It illustrates the changes in the state of the flipflop outputs in response to the clock. The negative-going edge of $CLK\ 1$ causes the Q_0 output of $FF0$ to go HIGH. It has no effect on $FF1$ because a negative-going transition must occur to trigger the flipflop. After the leading edge of $CLK\ 1$, $Q_0 = 1$ and $Q_1 = 0$.

The negative-going edge of $CLK\ 2$ causes Q_0 go LOW, and it triggers $FF1$, causes Q_1 to go HIGH. After the leading edge of $CLK\ 2$, $Q_0 = 0$ and $Q_1 = 1$.

The negative-going edge of $CLK\ 3$ causes Q_0 to go HIGH again and it has no effect on $FF1$. Thus after the leading edge of $CLK\ 3$, $Q_0 = 1$ and $Q_1 = 1$.

The negative-going edge of $CLK\ 4$ causes Q_0 to go LOW, causing Q_1 to go LOW. After the leading edge of $CLK\ 4$, $Q_0 = 0$ and $Q_1 = 0$. The counter has now recycled to its original state.

Since it goes through a binary sequence (00, 01, 10, 11) this counter is a binary counter.

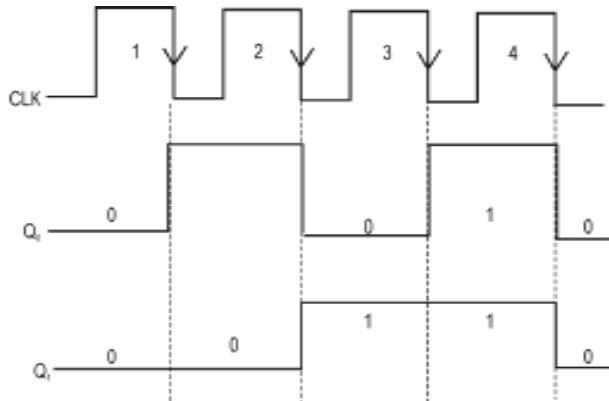


Fig. 3.62: Timing diagram for 2 bit counter

3.23.2 3 Bit Asynchronous Binary Counter

A 3 bit asynchronous binary counter is shown in **Figure 3.63**. The operation is the same as that of the 2 bit counter, except that the 3 bit counter has eight states, due to its 3 flipflops. The timing diagram for 3 bit asynchronous binary counter is shown in **Figure 3.64**.

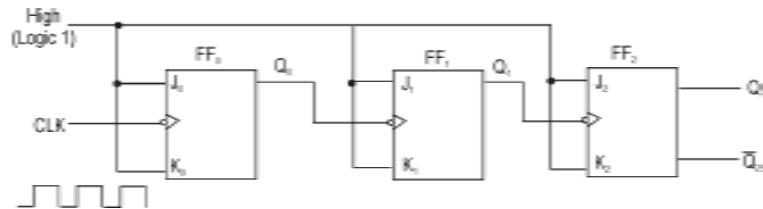


Fig. 3.63: 3 bit asynchronous binary counter

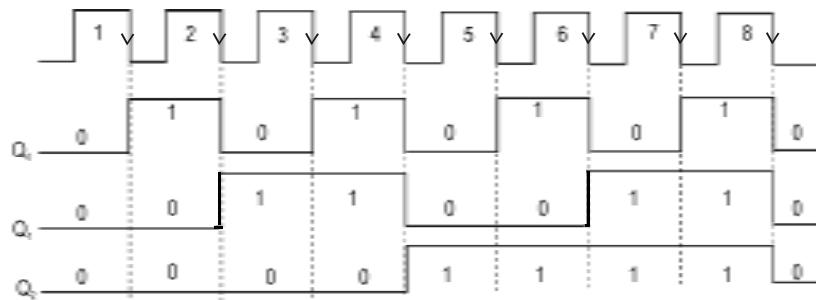


Fig. 3.64: Timing diagram

3.23.3 Propagation Delay

Asynchronous counters are commonly referred to as **Ripple Counters** for the following reason:

The effect of the input clock pulse is first felt by FF_0 . This effect cannot get to FF_1 immediately because of the propagation delay through FF_0 . Then there is the propagation delay through FF_1 before FF_2 can be triggered. Thus the effect of an input clock pulse “ripples” through the counter, taking same time, due to propagation delays, to reach the last flipflop.

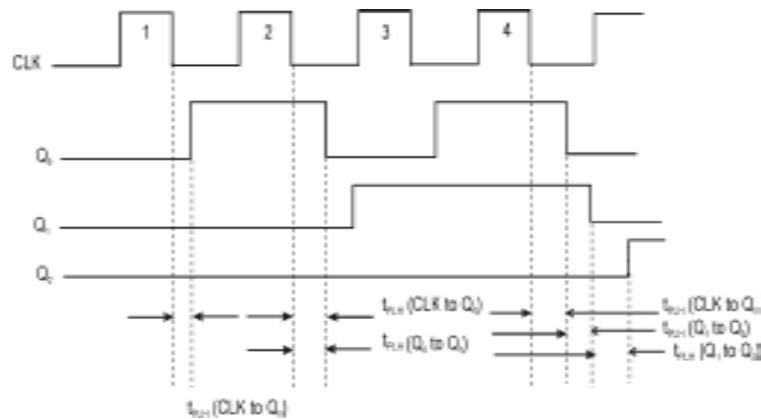


Fig. 3.65: Propagation delays

Figure 3.65 shows the ripple clocking effect for the first four clock pulses, with the propagation delays indicated. The propagation delay of the first stage is added in the propagation delay of second stage to decide the transition time for third stage. The cumulative delay of an asynchronous counter is

a major disadvantage in many applications because it limits the rate at which the counter clocked and creates decoding problems.

For example, each flipflop has a propagation delay for 10 ns, total delay time,

$$t_p = 3 \times 10 = 30 \text{ ns}$$

The maximum clock frequency,

$$f_{max} = \frac{1}{t_p} = \frac{1}{30 \times 10^{-9}} = 33.33 \text{ MHz.}$$

3.23.4 4 Bit Asynchronous Binary Counter

A 4 bit asynchronous binary counter is shown in **Figure 3.66**. This counter has 16 states, due to 4 flipflops. The timing diagram is shown in **Figure 3.67** for 16 clock pulses.

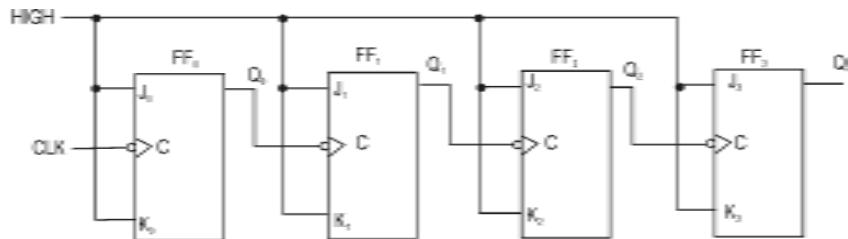


Fig. 3.66: 4 bit asynchronous counter

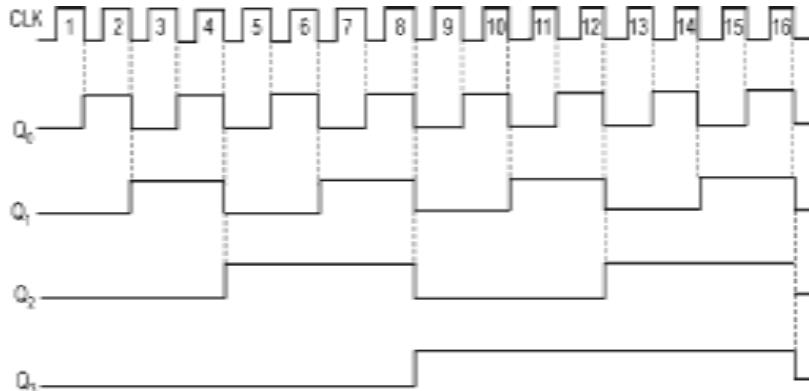


Fig. 3.67: Timing diagram

3.23.5 Asynchronous Decade (MOD 10) Counters

The modulus (N) of a counter is the number of unique states that the counter will sequence through. The maximum possible states (maximum modulus) of a counter is 2^n , where ' n ' is the number of flipflops in the counter.

Counters can also be designed to have a number of States in their sequence that is less than the maximum of 2^n . The resulting sequence is called a **truncated sequence**.

One common modulus for counters with truncated sequences is ten, called **MOD 10**.

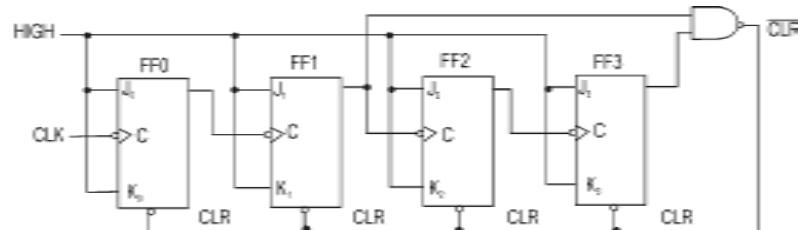


Fig. 3.68: Asynchronous Decade Counter

$$2^n \geq N$$

$$2^4 \geq 10$$

$$n = 4 = \text{Number of flipflops}$$

Counters with 10 states in their sequence are called **decade** counters. A decade counter with a count sequence of zero (0000) through nine (1001) is a BCD decade counter. It must recycle back to 0000 state after the 1001 state. To make the counter recycle after the count of nine (1001) is to decode count ten (1010) with a NAND gate and connect the output (0) of the NAND gate to the clear (CLR) inputs of the flipflops as shown in **Figure 3.68**. The timing diagram of decade counter is shown in **Figure 3.69**.

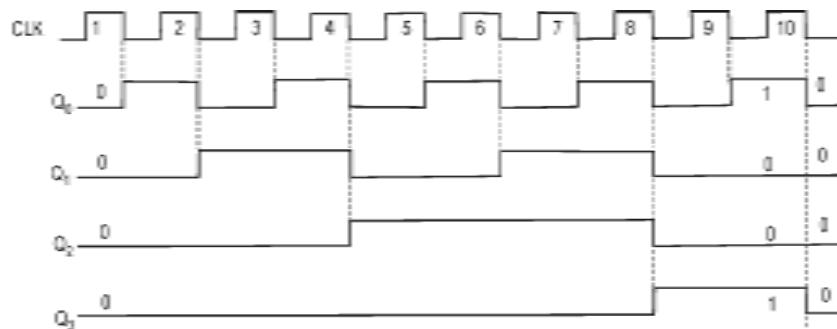


Fig. 3.69: Timing Diagram

3.24 SYNCHRONOUS COUNTERS

The term ‘synchronous’ refers to events that have a fixed time relationship with each other. In synchronous counter, the clock pulses are applied to all flipflops simultaneously. Hence there is minimum propagation delay. **Table 3.50** shows the comparison between synchronous and asynchronous counters.

TABLE 3.50 : Comparison between Synchronous and asynchronous counters

Sl. No.	Asynchronous Counters	Synchronous Counters
1.	All the flipflops are not clocked simultaneously.	All the flipflops are clocked simultaneously.
2.	The delay times of all flip-flops are added. Therefore there is considerable propagation delay.	There is minimum propagation delay.
3.	The maximum frequency depends on modulus.	The maximum frequency does not depend on modulus.
4.	Logic circuit is very simple.	Circuit is complex.
5.	Minimum number of logic devices are needed.	The number of logic devices is more than ripple counter.
6.	Cheaper than synchronous counters.	Costlier than ripple counters.

3.24.1 Two Bit Synchronous Counter

In this counter the clock signal is connected in parallel to clock inputs of both the flipflops $FF0$ and $FF1$. The output of $FF0$ (Q_0) is connected to J_1 and K_1 inputs of the second flipflop $FF1$. Assume that the counter is initially in the binary 0 states i.e., both flipflops are RESET. When positive edge of the first clock pulse is applied, $FF0$ will toggle because $J_0 = K_0 = 1$, whereas $FF1$ output will remains zero because $J_1 = K_1 = 0$. After first clock pulse $Q_0 = 1$ and $Q_1 = 0$.

When the leading edge of $CLK\ 2$ occurs, $FF0$ will toggle and Q_0 will go Low. Since $FF1$ has a HIGH ($Q_0 = 1$) on its J_1 and K_1 inputs at the triggering edge of this clock pulse, the flipflop toggles and Q_1 goes HIGH. Thus after $CLK2$, $Q_0=0$ and $Q_1=1$.

When the positive edge of $CLK3$ occurs, $FF0$ again toggles to the SET state ($Q_0=1$) and $FF1$ remains SET ($Q_1=0$) because its J_1 and K_1 inputs are both LOW ($Q_0 = 0$). After this triggering edge, $Q_0 = 1$ and $Q_1 = 1$.

Finally, at the positive edge of $CLK4$, Q_0 and Q_1 go LOW because they both have a toggle condition on their J and K inputs. i.e., $Q_0 = Q_1 = 0$. The counter has now recycled to its original state.

Figure 3.70 shows a 2 bit synchronous binary counter and **Figure 3.71** shows the timing diagram for the counter.

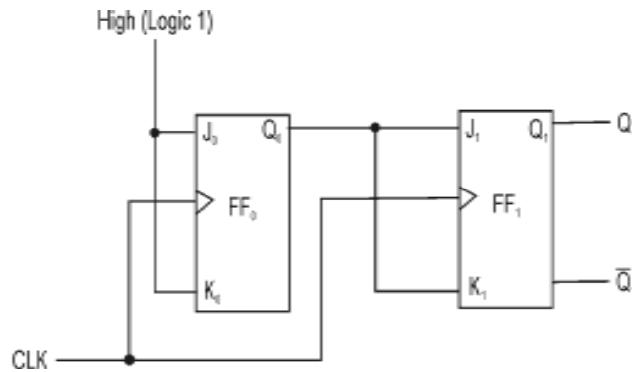


Fig. 3.70: 2 Bit synchronous binary counter

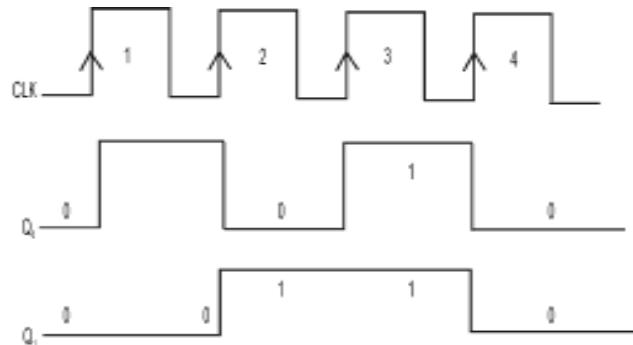


Fig. 3.71: Timing Diagram

3.24.2 3 Bit Synchronous Binary Counter

A 3 bit synchronous binary counter is constructed with three JK flipflops and an AND gate as shown in **Figure 3.72**. The output of FF_0 (Q_0) changes on each clock pulse as the counter progresses from its original state to its final state and then back to its original state as shown in the timing diagram of **Figure 3.73**. To produce this operation, FF_0 must be held in the toggle mode by constant HIGH, on its J_0 and K_0 inputs.

The output of FF_1 (Q_1) goes to the opposite state following each time $Q_0 = 1$. This change occurs at CLK 2, CLK 4, CLK 6 and CLK 8. To produce this operation, Q_0 is connected to the J_1 and K_1 inputs of FF_1 . When $Q_0 = 1$, and a clock pulse occurs, FF_1 is in the toggle mode and therefore changes state. When $Q_0 = 0$, FF_1 remains in its present state.

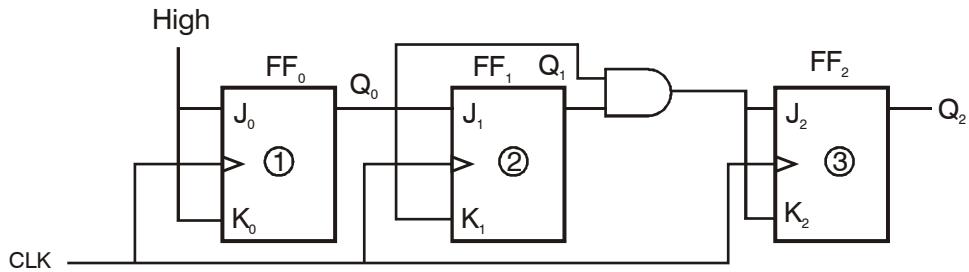
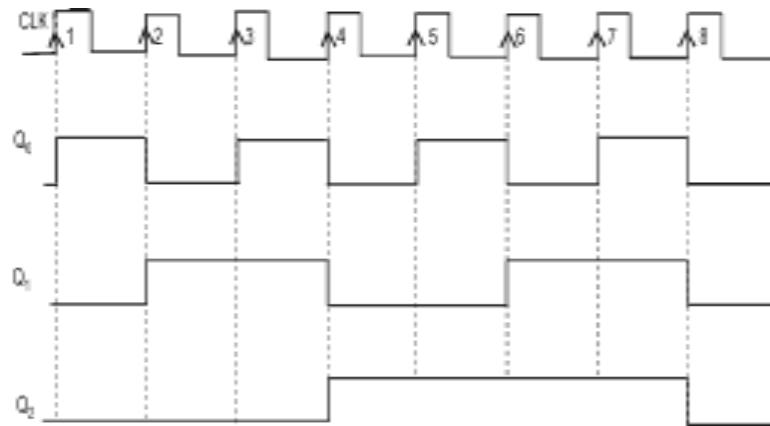


Fig. 3.72 : 3 bit synchronous binary counter



CLOCK Pulse	Q_2	Q_1	Q_0
Initially	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
8 (recycles)	0	0	0

Fig. 3.73: Timing Diagram

Q_2 changes state both times, it is preceded by the unique condition in which both Q_0 and Q_1 are HIGH. This condition is detected by the AND gate and applied to the J_2 and K_2 inputs of FF_2 . Whenever $Q_0 = Q_1 = 1$, the output of the AND gate makes the $J_2 = K_2 = 1$ and FF_2 toggles on the following clock pulse. Otherwise the J_2 and K_2 inputs of FF_2 are held LOW by the AND gate input and FF_2 does not change state.

3.24.3 4 Bit Synchronous Binary Counter

In this counter 4 JK flipflops and two AND gates are used as shown in **Figure 3.74**. The operation of first 3 flipflops is same as the 3 bit counter. For the fourth stage, flip-flop has to change the state, when $Q_0 = Q_1 = Q_2 = 1$. This condition is decoded by 3-input AND gate G_2 . Therefore, when $Q_0 = Q_1 = Q_2 = 1$, flipflop FF_3 toggles and for all other times it is in no change condition. The timing diagram for this counter is shown in **Figure 3.75**. Points where the AND gate outputs are HIGH are indicated by the shaded areas.

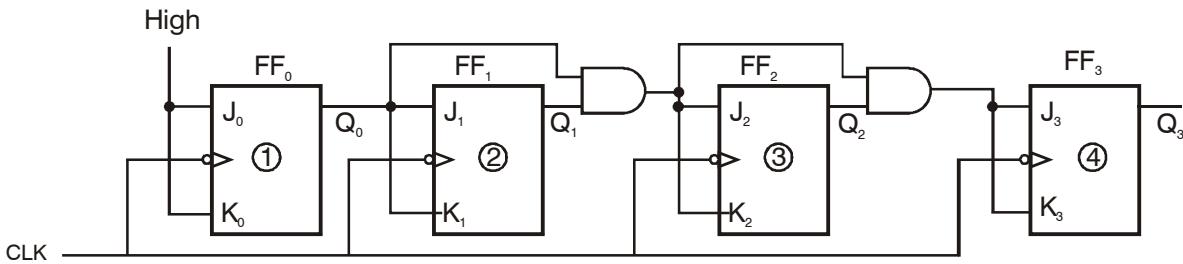


Fig. 3.74: 4 bit synchronous binary counter

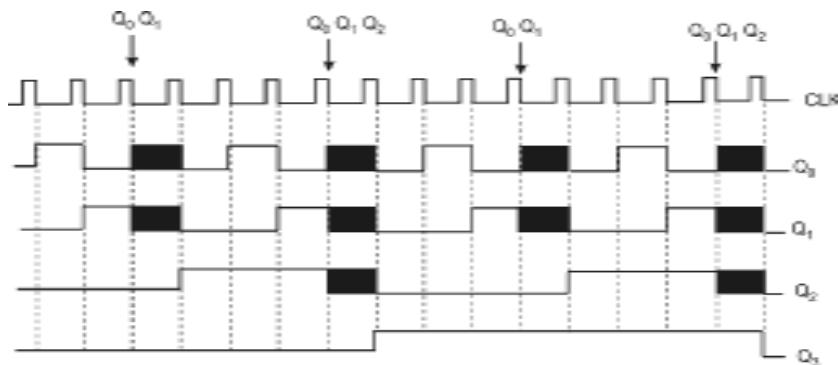


Fig. 3.75: Timing diagram

3.24.4 4 Bit Synchronous Decade Counter

BCD decade counter has a count sequence from 0000 to 1001 (9). After 1001 state it must recycle back to 0000 state. This counter requires four flipflops and AND/OR logic as shown in **Figure 3.76**. The timing diagram for the decade counter is shown in **Figure 3.77**.

$$\begin{aligned}
 J_0 &= K_0 = 1 \\
 J_1 &= K_1 = Q_0 \bar{Q}_3 \\
 J_2 &= K_2 = Q_0 Q_1 \\
 J_3 &= K_3 = Q_0 Q_1 Q_2 + Q_0 Q_3
 \end{aligned}$$

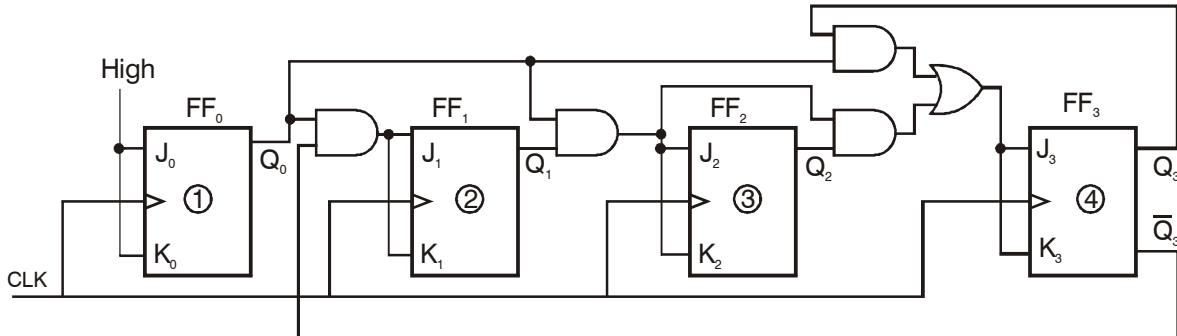


Fig. 3.76: 4 Bit Synchronous Decade Counter

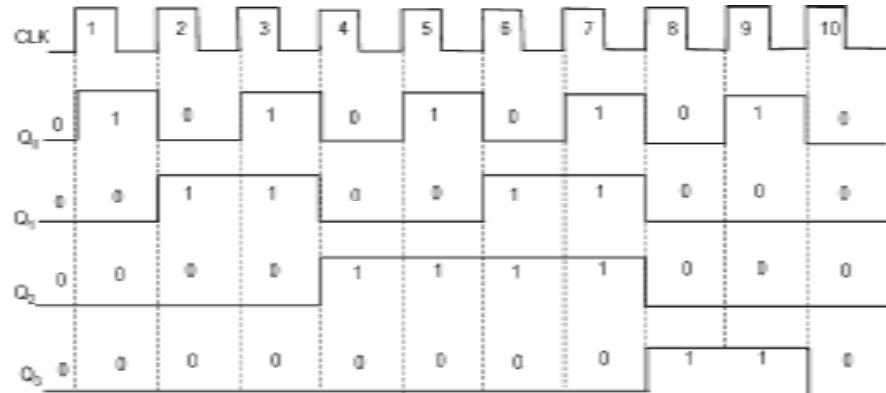


Fig. 3.77: Timing Diagram

3.24.5 Synchronous UP/DOWN Counter

An up/down counter is a bi-directional counter, capable of progressing in either direction through a certain sequence. A MOD 8 (3 bit) UP/DOWN counter that advances upward through its sequence (0, 1, 2, 3, 4, 5, 6, 7) and then can be reversed so that it goes through the sequence in the opposite direction (7, 6, 5, 4, 3, 2, 1, 0) is an illustration of up/down sequential operation.

To form a parallel (synchronous) UP/DOWN counter, the control input (UP/DOWN) is used to allow either the normal output or the inverted output of one flipflop to the J and K inputs of the next flipflop. When UP/DOWN = 1, the MOD 8 counter will count from 000 to 111 and UP/DOWN = 0, it will count from 111 to 000.

When UP/DOWN = 1, it will enable AND gates 1 and 3 and disable AND gates 2 and 4. This allows the Q_0 and Q_1 outputs through the AND gates to the J and K inputs of the following flipflops, so that the counter counts up as pulses are applied. When UP/DOWN = 0, the reverse action takes place.

$$J_1 = K_1 = (Q_0 \cdot UP) + (\bar{Q}_0 \cdot DOWN)$$

$$J_2 = K_2 = (Q_0 \cdot Q_1 \cdot UP) + (\bar{Q}_0 \cdot \bar{Q}_1 \cdot DOWN)$$

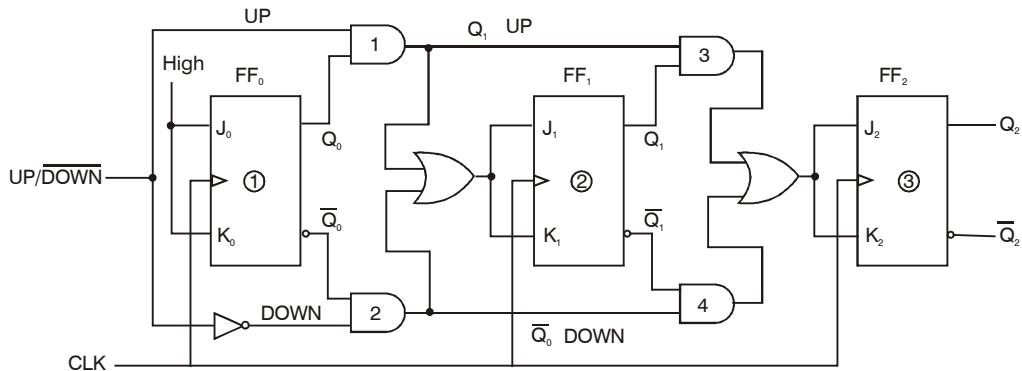


Fig. 3.78: 3 Bit (MOD 8) UP/DOWN Synchronous Counter

3.25 MODULUS-N-COUNTERS

The counter with ' n ' flipflops has maximum MOD number 2^n . Find the number of flipflops (n) required for the desired MOD number (N) using the equation,

$$2^n \geq N$$

(i) For example, a 3 bit binary counter is a **MOD 8 counter**. The basic counter can be modified to produce MOD numbers less than 2^n by allowing the counter to skip those are normally part of counting sequence.

$$n = 3$$

$$N = 8$$

$$2^n = 2^3 = 8 = N$$

(ii) **MOD 5 Counter:**

$$2^n = N$$

$$2^n = 5$$

$$2^2 = 4 \text{ less than } N.$$

$$2^3 = 8 > N(5)$$

\therefore 3 flipflops are required.

(iii) MOD 10 Counter:

$$2^n = N = 10$$

$$2^3 = 8, \text{ less than } N;$$

$$2^4 = 16 > N(10).$$

To construct any MOD- N counter, the following method can be used.

1. Find the number of flipflops (n) required for the desired MOD number (N) using the equation,
 $2^n \geq N$.
2. Connect all the flipflops as a required counter.
3. Find the binary number for N .
4. Connect all flipflop outputs for which $Q = 1$ when the count is N , as inputs to NAND gate.
5. Connect the NAND gate output to the CLR input of each flipflop.

When the counter reaches N^{th} state, the output of the NAND gate goes LOW, resetting all flipflops to 0. Therefore the counter counts from 0 through $N - 1$.

For example, MOD-10 counter reaches state 10 (1010). i.e., $Q_3 Q_2 Q_1 Q_0 = 1010$. The outputs Q_3 and Q_1 are connected to the NAND gate and the output of NAND gate goes LOW and resetting all flipflops to zero. Therefore MOD-10 counter counts from 0000 to 1001 and then recycles to the zero value. The MOD-10 counter circuit is shown in **Figure 3.79**.

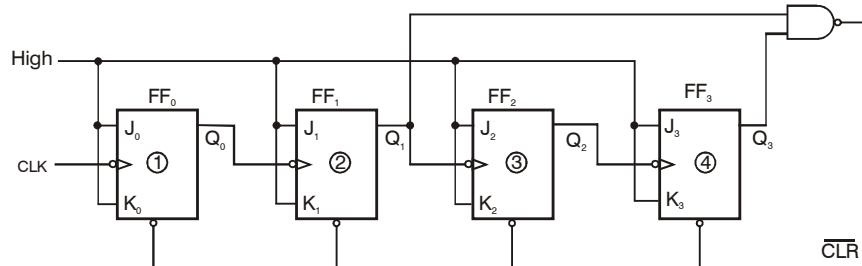


Fig. 3.79 : MOD-10 (Decade) Counter

3.26 SHIFT REGISTER COUNTERS

A shift register counter is a basically a shift register with the serial output connected back to the serial input to produce special sequences. Two of the most common types of shift register counters are:

- (i) Johnson Counter (Shift Counter)
- (ii) Ring Counter.

3.26.1 Johnson Counter (Shift Counter)

In a Johnson counter, the complement of the output of the last flipflop is connected back to the D input of the first flipflop. This feedback arrangement produces a characteristic sequence of states as shown in **Table 3.51** for a 4 bit Johnson counter. The 4 bit sequence has a total of eight states. In general, a Johnson counter will produce a modulus of $2n$, where ‘ n ’ is the number of stages in the counter. For example 5 bit Johnson counter has 10 states.

TABLE 3.51 : 4 Bit Johnson Sequence

Clock Pulse	Q_0	Q_1	Q_2	Q_3
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
0	0	0	0	0

Figure 3.80 shows the circuit of Johnson Counter. The Q output of each stage is connected to the D input of the next stage. The complement of the output of FF_3 (\bar{Q}_3) is connected back to the D input of FF_0 . The counter fills up with 1s from left to right and then fills up 0s again. The timing diagram of 4-bit Johnson Counter is shown in **Figure 3.81**.

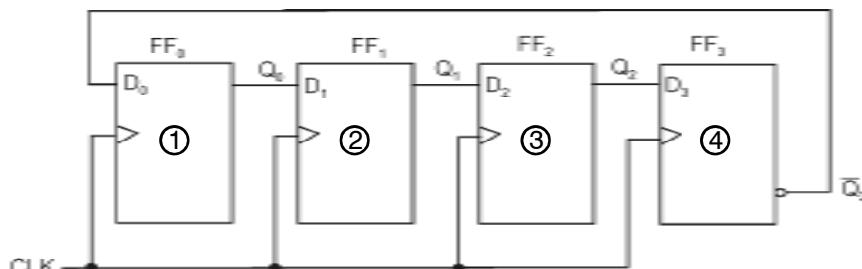


Fig. 3.80: Four-bit- Johnson counter

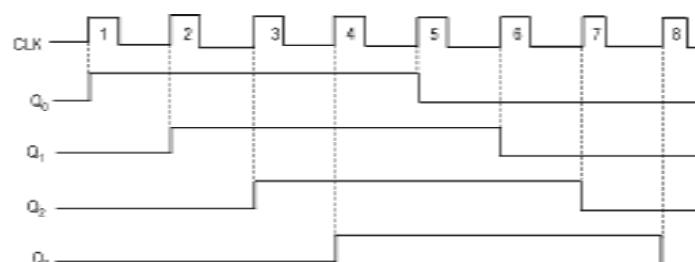


Fig. 3.81: Timing sequence for a 4-bit Johnson counter

3.26.2 Ring Counters

The ring counter utilizes one flipflop for each state in its sequence. It has the advantage that decoding gates are not required. In case of a 10 bit ring counter, there is a unique output for each decimal digit.

Figure 3.82 shows the circuit of a ring counter. The output Q_0 sets D_1 input, Q_1 sets D_2 , Q_2 sets D_3 and Q_3 is fed back to D_0 . Because of these connections, bits are shifted left one position per positive clock edge and fed back to the input. All the flipflops are clocked together. When CLR goes low then back to high, the output is 0000. The first positive clock edge shifts MSB to LSB position and other bits to one position left so that the output becomes $Q = 0010$. This process continues on second and third positive clock edge so that successive outputs are 0100 and 1000. The fourth positive clock edge starts the cycle all over again and output is 0001. Thus the stored 1 bits follow a circular path (i.e., the stored 1 bits move left through all flipflops and the final flipflop sends it back to the first flipflop). This action has given it the name of ring counter. **Figure 3.83**.

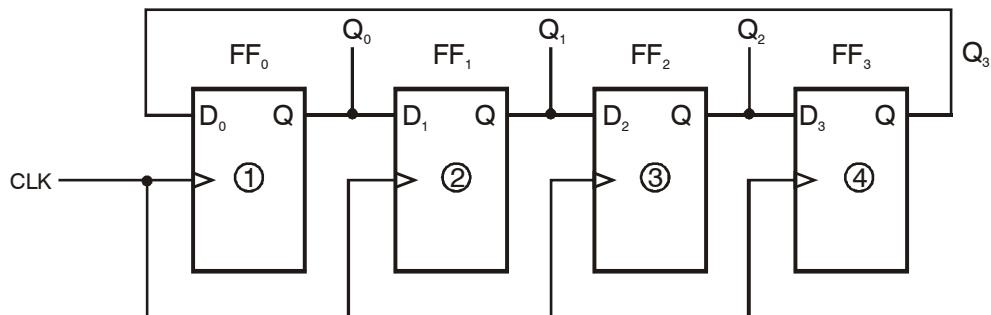


Fig. 3.82 : Ring Counter

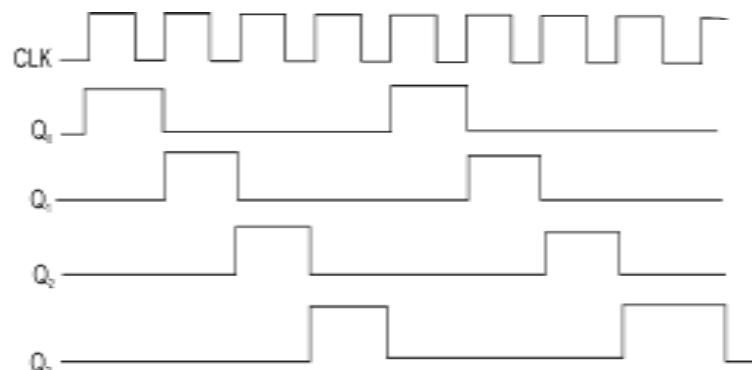


Fig. 3.83: Timing Diagram

3.27 DESIGN OF COUNTERS

The procedure for design of counters as follows:

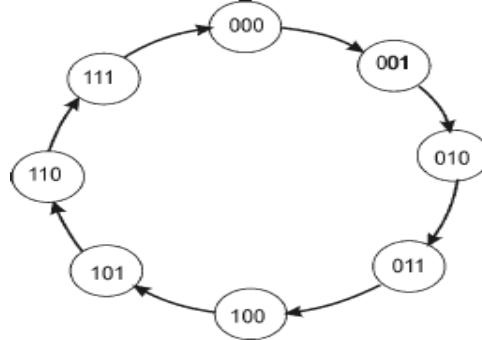
- Step 1:** Specify the counter sequence and draw a state diagram.
- Step 2:** Derive a next-state table from the state diagram.
- Step 3:** Make the state assignment and develop a transition table showing the flipflop inputs required.
- Step 4:** Draw the karnaugh maps for each input of each flipflop.
- Step 5:** Derive the logic expression for each flipflop input from the K-maps.
- Step 6:** Implement the expressions with combinational logic and combine with the flipflops to form the counter.

Example 3.10: Using JK flipflops, design a synchronous counter which counts in the sequence,

000, 001, 010, 011, 100, 101, 110, 111, 000.

Solution:

Step 1: State Diagram



Step 2: State Table

Present State	Next State
000	001
001	010
010	011
011	100
100	101
101	110
110	111
111	000

Step 3: Excitation Table for Counter**Excitation Table for JK flipflop:**

$Q(t)$	$Q(t+1)$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Excitation Table for Counter:

Present State			Next State			Flipflop Inputs					
q_2	q_1	q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	X	1	X	1	X	1

Step 4: K-map Simplification**For J_2**

		$Q_1 Q_0$	00	01	11	10
		Q_2	0	0	1	0
		1	X	X	X	X
0	0	0	0	0	1	0
0	1	1	X	X	X	X
1	0	0	X	X	X	X
1	1	1	X	X	X	X

$$J_2 = Q_1 Q_0$$

For K_2

		$Q_1 Q_0$	00	01	11	10	
		Q_2	0	X	X	(X)	X
		1	0	0	(1)	0	
0	0	0	X	X	(X)	X	
0	1	1	X	X	(1)	0	
1	0	0	0	0	(1)	0	
1	1	1	X	X	(1)	0	

$$K_2 = Q_1 Q_0$$

For J_1

		$Q_1 Q_0$	00	01	11	10	
		Q_2	0	0	1	X	X
		1	0	1	X	X	X
0	0	0	0	0	1	X	
0	1	1	1	X	X	X	
1	0	0	1	X	X	X	
1	1	1	X	X	X	X	

$$J_1 = Q_0$$

For K_1

		$Q_1 Q_0$	00	01	11	10	
		Q_2	0	X	(X)	1	0
		1	X	(X)	1	0	
0	0	0	X	(X)	1	0	
0	1	1	(X)	1	X	0	
1	0	0	(X)	1	X	0	
1	1	1	(X)	1	X	0	

$$K_1 = Q_0$$

For J_0				For K_0							
$Q_1 Q_0$		Q_2		$Q_1 Q_0$		Q_2					
		00	01	11	10			00	01	11	10
0	1	X	X	1		0	X	1	1	X	
	0	X	X	1			X	1	1	X	

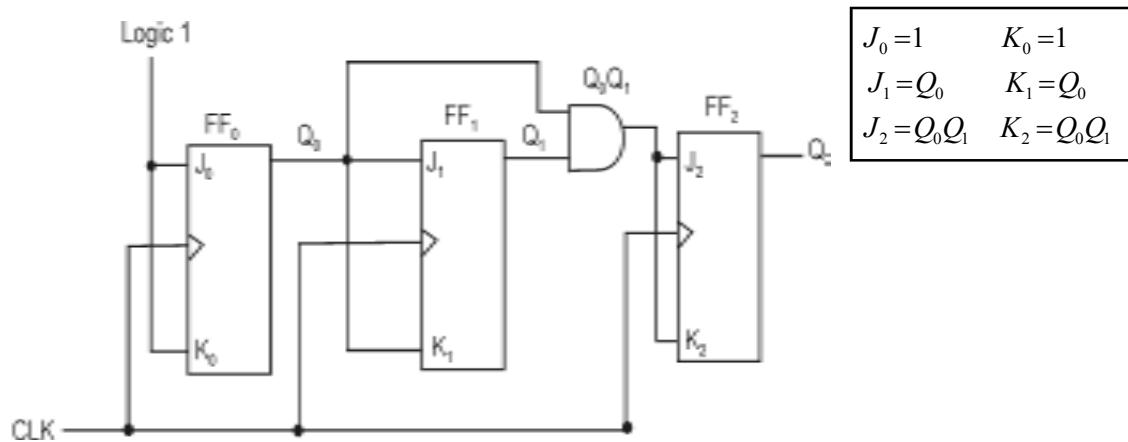
 $J_1 = Q_0$ $K_1 = Q_0$ **Step 5: Circuit Diagram**

Fig.3.84 : Synchronous Counter

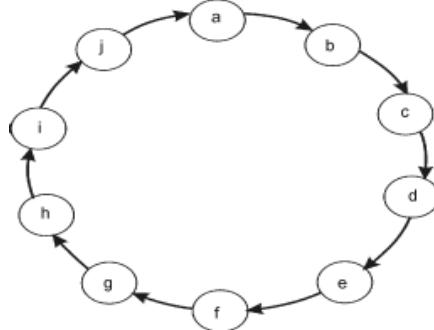
Example 3.11: Design a MOD-10 synchronous counter using JK flipflops. Write excitation table and state table.

Solution

$$2^n \geq N = 10$$

$$2^4 > 10$$

$\therefore n=4$; 4 flipflops are required.

Step 1: State Diagram

Step 2: State Table

Present State	Next State
a	b
b	c
c	d
d	e
e	f
f	g
g	h
h	i
i	j
j	a

Step 3: Transition Table

Present State				Next State			
q_3	q_2	q_1	q_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0

Step 4: Excitation Table

Present State				Next State				Excitation Inputs							
q_3	q_2	q_1	q_0	Q_3	Q_2	Q_1	Q_0	J_3	K_3	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	0	0	1	0	X	0	X	0	X	1	X
0	0	0	1	0	0	1	0	0	X	0	X	1	X	X	1
0	0	1	0	0	0	1	1	0	X	0	X	X	0	1	X
0	0	1	1	0	1	0	0	0	X	1	X	X	1	X	1
0	1	0	0	0	1	0	1	0	X	X	0	0	X	1	X
0	1	0	1	0	1	1	0	0	X	X	0	1	X	X	1
0	1	1	0	0	1	1	1	0	X	X	0	X	0	1	X
0	1	1	1	1	0	0	0	1	X	X	1	X	1	X	1
1	0	0	0	1	0	0	1	X	0	0	X	0	X	1	X
1	0	0	1	0	0	0	0	X	1	0	X	0	X	X	1

Step 5: K-map Simplification**For J_3 map**

$q_1 q_0$	00	01	11	10
$q_3 q_2$	00	0	0	0
	01	0	0	0
	11	X	X	(X)
	10	X	X	X

$$J_3 = q_2 q_1 q_0$$

For K_3 map

$q_1 q_0$	00	01	11	10
$q_3 q_2$	00	X	(X) X	X
	01	X	X X	X
	11	X	X X	X
	10	0	(1) X	X

$$K_3 = q_0$$

For J_2

$q_1 q_0$	00	01	11	10
$q_3 q_2$	00	0	0	(1) 0
	01	X	X	X
	11	X	X	X
	10	0	0	(X) X

$$J_3 = q_1 q_0$$

For K_2

$q_1 q_0$	00	01	11	10
$q_3 q_2$	00	X	X	(X) X
	01	0	0	1 0
	11	X	X	X
	10	X	X	(X) X

$$K_2 = q_1 q_0$$

For J_1

$q_1 q_0$	00	01	11	10
$q_3 q_2$	00	0	(1) X	X
	01	0	(1) X	X
	11	X	X	X
	10	0	0	X X

$$J_1 = \bar{q}_3 q_0$$

For K_1

$q_1 q_0$	00	01	11	10
$q_3 q_2$	X	(X) 1	0	
	X	X 1	0	
	X	X X	X	
	X	(X) X	X	

$$K_1 = q_0$$

For J_0				For K_0			
q_3q_2	00	01	11	00	01	11	10
00	1	X	X	1	X	1	X
01	1	X	X	1	X	1	X
11	X	X	X	X	X	X	X
10	1	X	X	X	X	X	X

$J_0 = 1$ $K_0 = 1$

Step 6: Circuit Diagram

$J_0 = 1$	$K_0 = 0$
$J_1 = q_3q_0$	$K_1 = q_0$
$J_2 = q_1q_0$	$K_2 = q_1q_0$
$J_3 = q_2q_1q_0$	$K_3 = q_0$

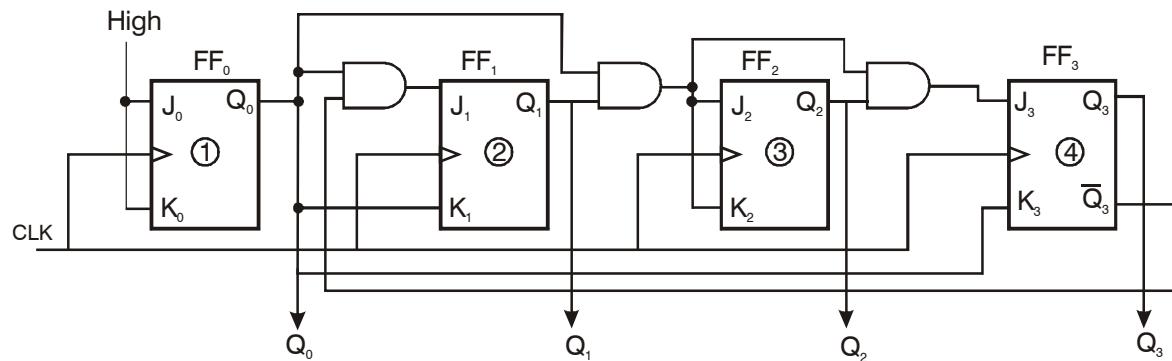


Fig. 3.85 : MOD 10 Synchronous Counter

Example 3.12: Design a synchronous 3-bit gray code up counter with the help of excitation table.

Solution: Gray code sequence: 000, 001, 011, 010, 110, 111, 101, 100.

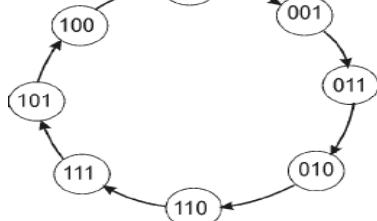
Step 1: State Diagram

Fig. 3.86

Step 2: State Table

Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	1	0	1	0
0	1	0	1	1	0
1	1	0	1	1	1
1	1	1	1	0	1
1	0	1	1	0	0
1	0	0	0	0	0

Step 3: Excitation Table

Excitation Table for JK flipflops.

$Q(t)$	$Q(t+1)$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Present State			Next State			Excitation Inputs					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	1	0	X	1	X	X	0
0	1	1	0	1	0	0	X	X	0	X	1
0	1	0	1	1	0	1	X	X	0	0	X
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	1	0	1	X	0	X	1	X	0
1	0	1	1	0	0	X	0	0	X	X	1
1	0	0	0	0	0	X	1	0	X	0	X

Step 4: K-map Simplification**For J_2**

$Q_2 Q_1$	0	1
00	0	0
01	1	0
11	X	X
10	X	X

$J_2 = Q_1 \bar{Q}_0$

For K_2

$Q_2 Q_1$	0	1
00	X	X
01	X	X
11	0	0
10	1	0

$K_2 = \bar{Q}_1 \bar{Q}_0$

For J_1

$Q_2 Q_1$	0	1
00	0	1
01	X	X
11	X	X
10	0	0

$J_1 = \bar{Q}_2 Q_0$

For K_1

$Q_2 Q_1$	0	1
00	X	X
01	0	0
11	0	1
10	X	X

$K_1 = Q_2 Q_0$

For J_0

$Q_2 Q_1$	0	1
00	1	X
01	X	0
11	1	X
10	X	0

$J_0 = \bar{Q}_2 \bar{Q}_1 + Q_2 Q_1$
 $= \bar{Q}_2 \oplus Q_1$

For K_0

$Q_2 Q_1$	0	1
00	X	0
01	1	X
11	X	0
10	1	X

$K_0 = \bar{Q}_2 Q_1 + Q_2 \bar{Q}_1$
 $= Q_2 \oplus Q_1$

Step 5: Circuit Diagram

$J_0 = \bar{Q}_2 \oplus Q_1$	$K_0 = Q_2 \oplus Q_1$
$J_1 = \bar{Q}_2 Q_0$	$K_1 = Q_2 Q_0$
$J_2 = Q_1 \bar{Q}_0$	$K_2 = \bar{Q}_1 \bar{Q}_0$

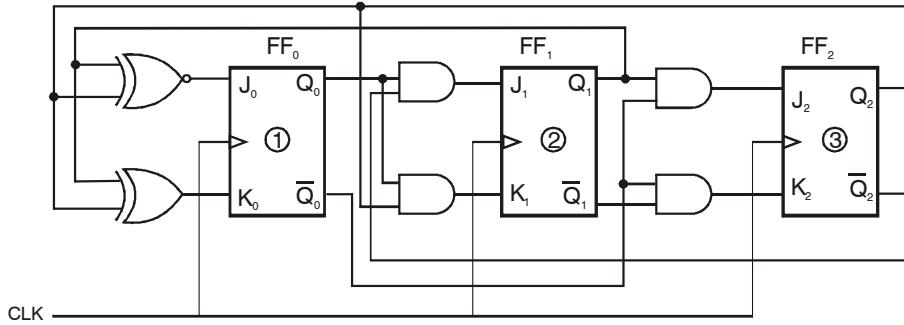


Fig. 3.87 : 3 bit Gray code up counter

Example 3.13: Design a 3 bit (MOD 8) synchronous UP-DOWN counter.

Solution: When UP/DOWN = 1, UP mode, UP/DOWN = 0, DOWN mode.

Step 1: State Diagram

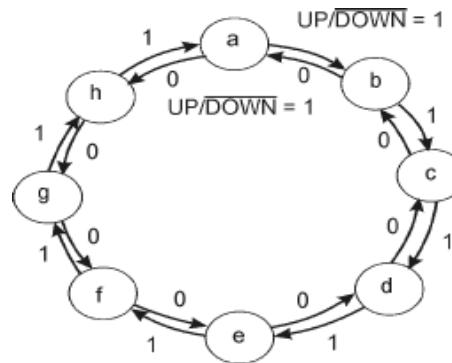


Fig. 3.88

Step 2: State Table

Control input UP/DOWN	Present State	Next State
0	a	h
0	b	a
0	c	b
0	d	c
0	e	d
0	f	e
0	g	f
0	h	g
1	a	b
1	b	c
1	c	d
1	d	e
1	e	f
1	f	g
1	g	h
1	h	a

Step 3: Excitation Table

Control Input	Pressure State			Next State			Excitation Inputs						
	UP/DOWN (C)	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	1	1	1	1	X	1	X	1	X	
0	0	0	1	0	0	0	0	X	0	X	X	1	
0	0	1	0	0	0	1	0	X	X	1	1	X	
0	0	1	1	0	1	0	0	X	X	0	X	1	
0	1	0	0	0	1	1	X	1	1	X	1	X	
0	1	0	1	1	0	0	X	0	0	X	X	1	
0	1	1	0	1	0	1	X	0	X	1	1	X	
1	1	1	1	1	1	0	X	0	X	0	X	1	
1	0	0	0	0	0	1	0	X	0	X	1	X	
1	0	0	1	0	1	0	0	X	1	X	X	1	
1	0	1	0	0	0	1	0	X	X	0	1	X	
1	0	1	1	1	0	0	1	X	X	1	X	1	
1	1	0	0	1	0	1	X	0	0	X	1	X	
1	1	0	1	1	1	0	X	0	1	X	X	1	
1	1	1	0	1	1	1	X	0	X	0	1	X	
1	1	1	1	0	0	0	X	1	X	1	X	1	

Step 4: K-map Simplification**For J_2**

		$Q_1 Q_0$	00	01	11	10
		CQ_2	00	1	0	0
		CQ_2	01	X	X	X
		CQ_2	11	X	X	(1)
		CQ_2	10	0	0	(1)

$$J_2 = \overline{C} \overline{Q}_1 Q_0 + C Q_1 Q_0$$

For K_2

		$Q_1 Q_0$	00	01	11	10
		CQ_2	00	X	X	X
		CQ_2	01	1	0	0
		CQ_2	11	0	0	(1)
		CQ_2	10	X	X	(X)

$$K_2 = \overline{C} \overline{Q}_1 Q_0 + C Q_1 Q_0$$

For J_1				For K_1							
CQ_2		00	01	11	10	CQ_2		00	01	11	10
00	1	0	X	X	X	00	X	X	0	1	
01	1	0	X	X	X	01	X	X	0	1	
11	0	1	X	X	X	11	X	X	1	0	
10	0	1	X	X	X	10	X	X	1	0	

$J_1 = CQ_0 + \bar{C}\bar{Q}_0$

For J_0				For K_0							
CQ_2		00	01	11	10	CQ_2		00	01	11	10
00	1	X	X	1		00	X	1	1	X	
01	1	X	X	1		01	X	1	1	X	
11	1	X	X	1		11	X	1	1	X	
10	1	X	X	1		10	X	1	1	X	

$J_0 = 1$

$K_0 = 1$

Step 5: Circuit Diagram

$C = UP/\bar{D}OWN$	
$J_0 = 1$	$K_0 = 1$
$J_1 = CQ_0 + \bar{C}\bar{Q}_0$	$K_1 = CQ_0 + \bar{C}\bar{Q}_0$
$J_2 = CQ_1Q_0 + \bar{C}\bar{Q}_1\bar{Q}_0$	$K_2 = CQ_1Q_0 + \bar{C}\bar{Q}_1\bar{Q}_0$

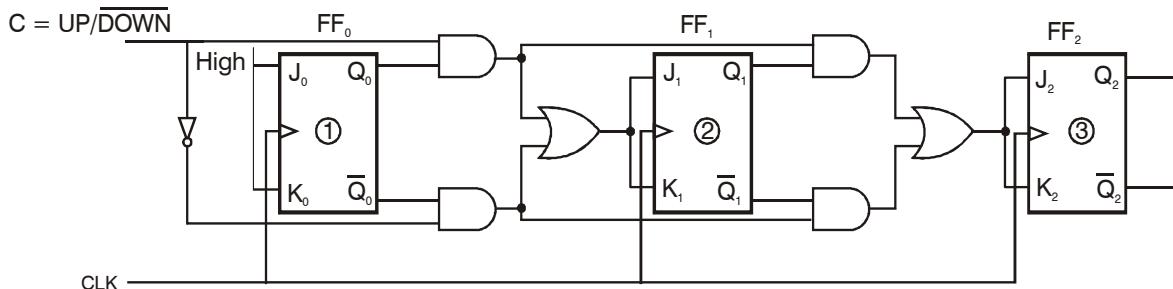


Fig. 3.89 : MOD 8 Synchronous UP-DOWN Counter

Example 3.14: Design a four state down counter using type T design procedure.

Solution

$$2^n \geq N = 4$$

$$2^2 = 4$$

$\therefore n = 2$; Two flipflops are required.

Step 1: State Diagram

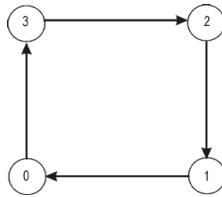


Fig. 3.90

Step 2: State Table

Present State	Next State
3	0
2	3
1	2
0	1

Step 3: Transition Table

Present State		Next State	
Q_1	Q_0	Q_1	Q_0
1	1	0	0
1	0	1	1
0	1	1	0
0	0	0	1

Step 4: Excitation Table

**Excitation Table for
T flipflop**

**Excitation Table for
down counter**

$Q(t)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

Present State		Next State		Excitation Inputs	
Q_1	Q_0	Q_1	Q_0	T_1	T_0
1	1	0	0	1	1
0	1	1	0	1	1
0	0	0	1	0	1

Step 5: K-map Simplification

Q_1	0	1
0	1	0
1	1	0

$$T_1 = \bar{Q}_0$$

Q_1	0	1
0	1	1
1	1	1

$$T_0 = 1$$

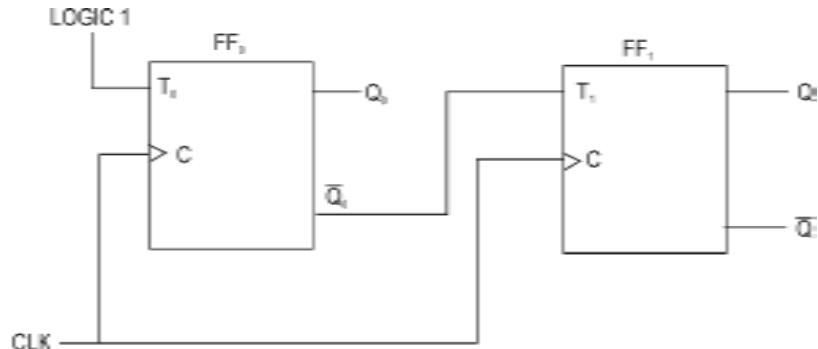
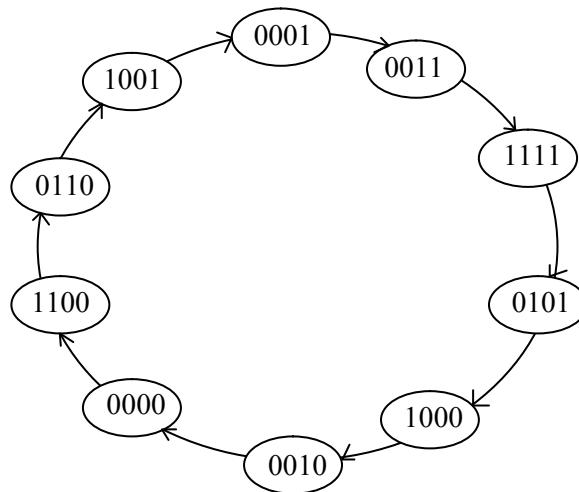
Step 6: Circuit Diagram

Fig. 3.91 : DOWN Counter

Example 3.15: Design a synchronous counter using JK flip-flops to count the following sequence:
“1 - 3 - 15 - 5 - 8 - 2 - 0 - 12 - 6 - 9”. (May 2011)

Step 1: State Diagram

Step 2: State Table

Present State	Next State
0001	0011
0011	1111
1111	0101
0202	1000
1000	0010
0010	0000
0000	1100
1100	0110
0110	1001
1001	0001

Step 3: Excitation Table for JK Flip-Flop

Present State	Next State	Inputs	
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Excitation Table for Counter:

Q_n				Q_{n+1}				FF Inputs							
q_3	q_2	q_1	q_0	Q_3	Q_2	Q_1	Q_0	J_3	J_2	J_1	J_0	K_3	K_2	K_1	K_0
0	0	0	1	0	0	1	1	0	0	1	X	X	X	X	0
0	0	1	1	1	1	1	1	1	1	X	X	X	X	0	0
1	1	1	1	0	1	0	1	X	X	X	X	1	0	1	0
0	1	0	1	1	0	0	0	1	X	0	X	X	1	X	1
1	0	0	0	0	0	1	0	X	0	1	0	1	X	X	X
0	0	1	0	0	0	0	0	0	0	X	0	X	X	1	X
0	0	0	0	1	1	0	0	1	1	0	0	X	X	X	X
1	1	0	0	0	1	1	0	X	X	1	0	1	0	X	X
0	1	1	0	1	0	0	1	1	X	X	1	X	1	1	X
1	0	0	1	0	0	0	1	X	0	0	X	1	X	X	0

Step 4: K - Map simplification**For J_3**

$Q_3 Q_2$	00	01	11	10
00	1	0	1	0
01	X	1	X	1
11	X	X	X	X
10	X	X	X	X

$$J_3 = \overline{Q_0} \overline{Q_1} + Q_0 Q_1 + Q_2 \overline{Q_3}$$

For K_3

$Q_3 Q_2$	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	1	X	1	X
10	1	1	X	X

$$K_3 = Q_3$$

For J_2

$Q_3 Q_2$	00	01	11	10
00	1	0	1	0
01	X	X	X	X
11	X	X	X	X
10	0	0	X	X

$$J_2 = Q_0 Q_1 + \overline{Q_0} \overline{Q_1} Q_3$$

For K_2

$Q_3 Q_2$	00	01	11	10
00	X	X	X	X
01	X	1	X	1
11	0	X	0	X
10	X	X	X	X

$$K_2 = \overline{Q}_3$$

For J_1

$Q_3 Q_2$	00	01	11	10
00	0	1	X	X
01	X	0	X	X
11	1	X	X	X
10	1	0	X	X

$$J_1 = Q_2 Q_3 + \overline{Q_0} \overline{Q_1} Q_3 + Q_0 Q_2 Q_3$$

For K_1

$Q_3 Q_2$	00	01	11	10
00	X	X	0	1
01	X	X	X	1
11	X	X	1	X
10	X	X	X	X

$$K_1 = Q_3 + \overline{Q_0} Q_1$$

For J_0

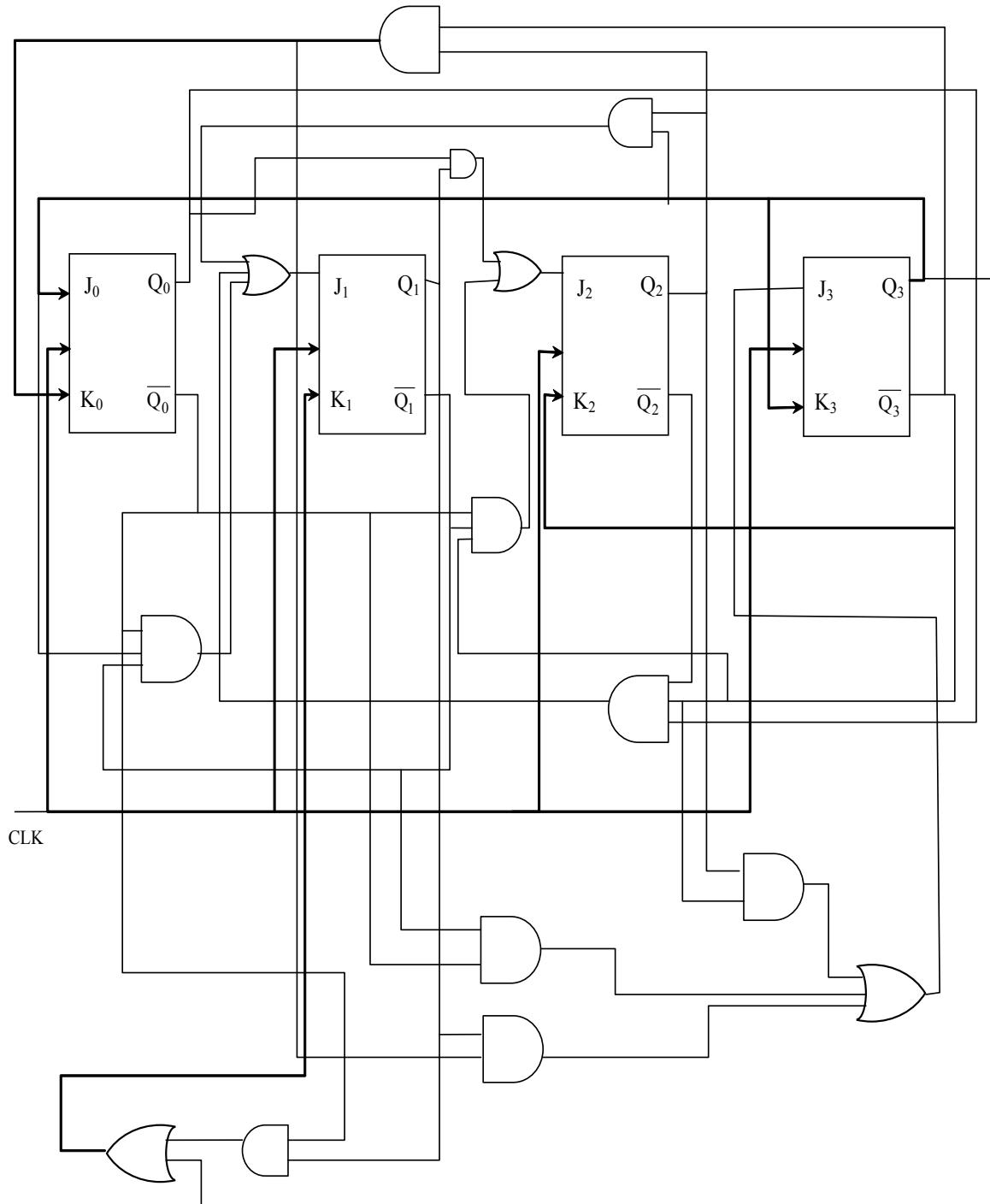
$Q_3 Q_2$	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	1	X	1	X
10	1	1	X	X

$$J_0 = Q_3$$

For K_0

$Q_3 Q_2$	00	01	11	10
00	X	0	0	X
01	X	1	X	X
11	X	X	0	X
10	X	0	X	X

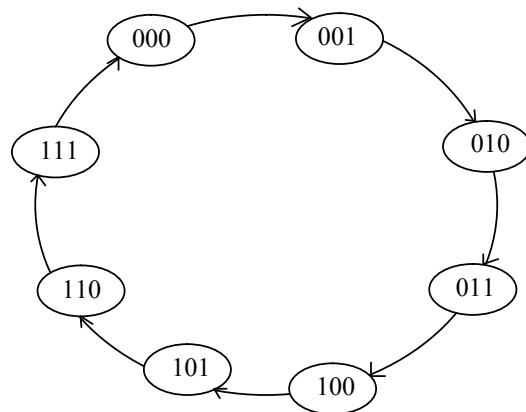
$$K_0 = Q_2 \overline{Q}_3$$

Step 5: Circuit Diagram

Example 3.16: Using D flip flops design a synchronous counter which counts in the sequence, 000, 001, 010, 011, 100, 101, 110, 111, 000.
(May 2013)

Solution:

Step 1: State Diagram



Step 2: State Table

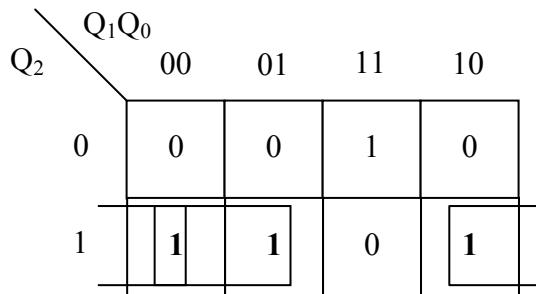
Present state	Next state
000	001
001	010
010	011
011	100
100	101
101	110
110	111
111	000

Step 3: Excitation table for counter

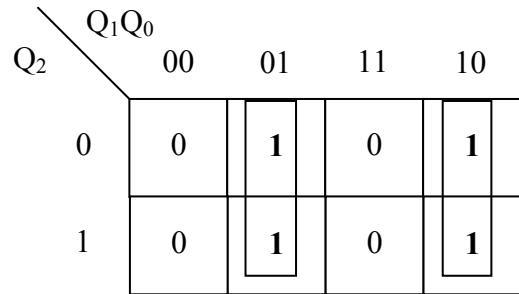
Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

Excitation table for D flip flop

Present table			Next state			Flip Flop Inputs		
q_2	q_1	q_0	Q_2	Q_1	Q_0	D_2	D_1	D_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0	0
1	0	0	1	0	1	1	0	1
1	0	1	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0

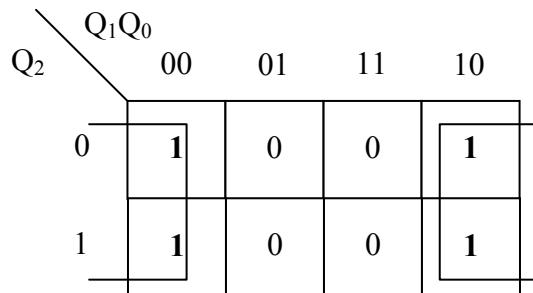
Step 4: K - map simplificationFor D_2 

$$D_2 = Q_2 \bar{Q}_1 + Q_2 \bar{Q}_0$$

For D_1 

$$D_1 = \bar{Q}_1 Q_0 + Q_1 \bar{Q}_0$$

$$= Q_1 \oplus Q_0$$

For D_0 

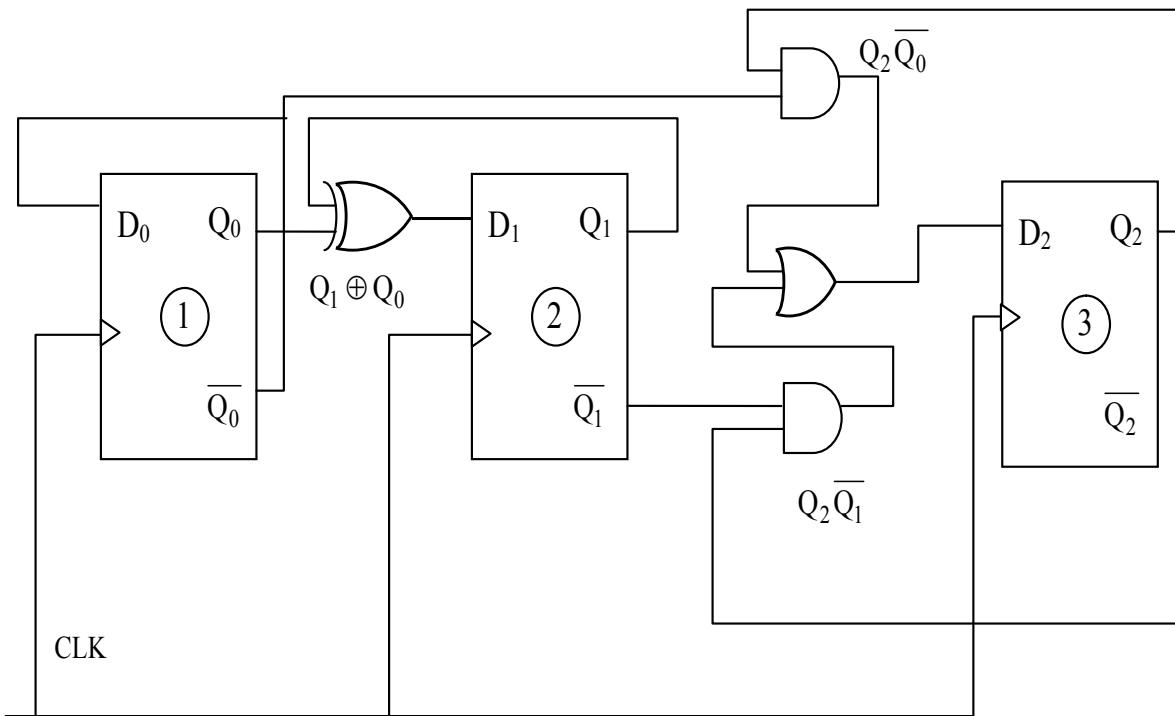
$$D_0 = \bar{Q}_0$$

Step 5: Circuit Diagram

$$D_0 = \overline{Q}_0$$

$$D_1 = Q_1 \oplus Q_0$$

$$D_2 = Q_2 \overline{Q}_1 + Q_2 \overline{Q}_0$$



Example 3.17: Using RS-FFs design a parallel counter which counts in the sequence.

000, 111, 101, 110, 001, 010, 000, ...

(Dec 2010)

Solution:

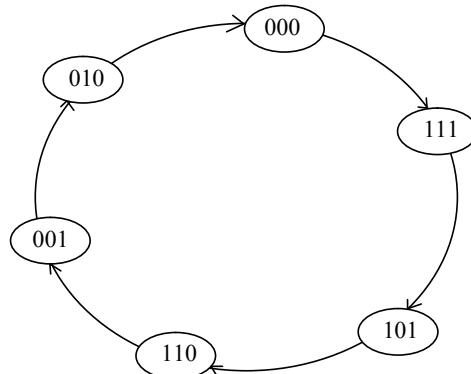
Excitation table

Q	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

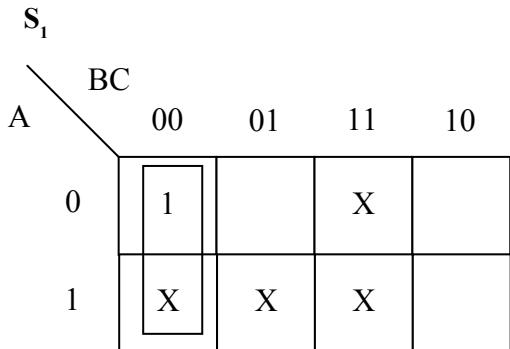
Function Table

S	R	Q	Q'
0	0	N	C
0	1	0	1
1	0	1	0
1	1	Indeterminate	

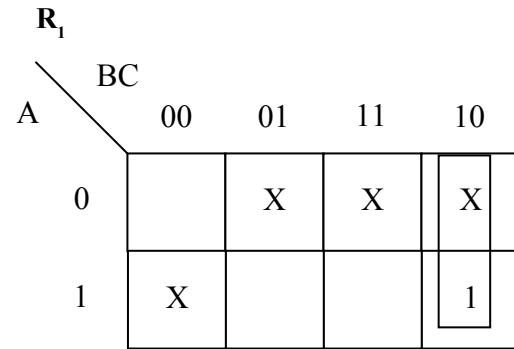
Step 1: State Diagram



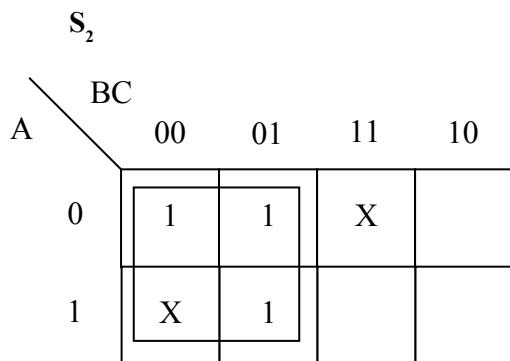
Step 2: State Table

Step 3: K-Map

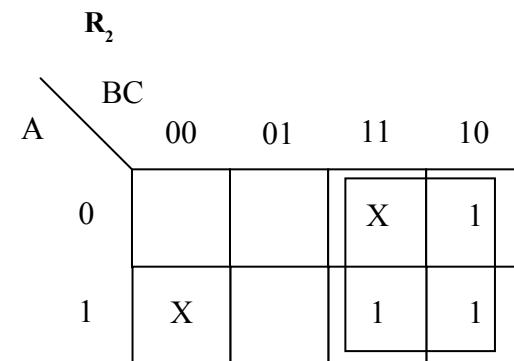
$$S_1 = B'C'$$



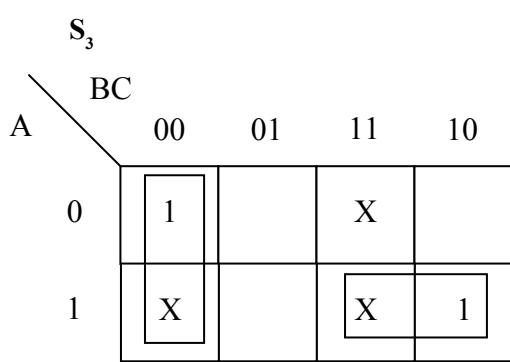
$$R_1 = BC'$$



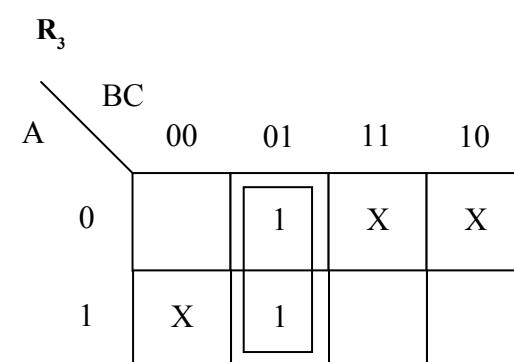
$$S_2 = B'$$



$$R_2 = B$$

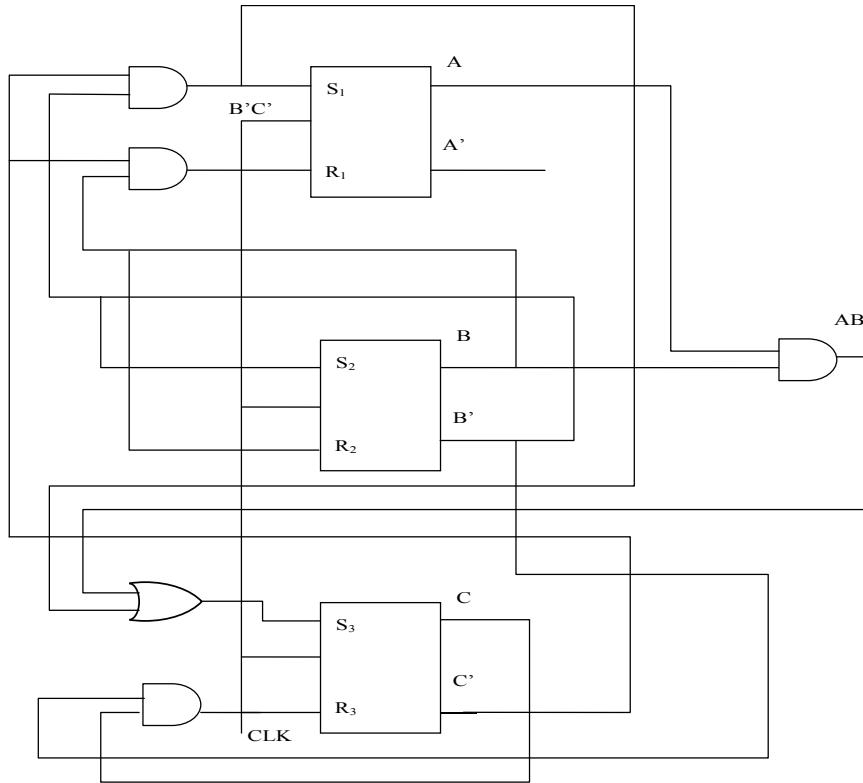


$$S_3 = B'C' + AB$$



$$R_3 = B'C$$

Step 4: Circuit Diagram



3.28 UP-DOWN RIPPLE COUNTER

The UP-DOWN counter is a combination of the up-counter and the down-counter. As the UP-DOWN counter has the capability of counting upwards as well as downwards, it is also called **Multimode counter**. In an UP-counter, each flip-flop is triggered by the **normal** output of the preceding flip-flop; in a DOWN-counter, each flip-flop is triggered by the inverted output of the preceding flip-flop. In both the counters, the first flip-flop is triggered by the input pulses.

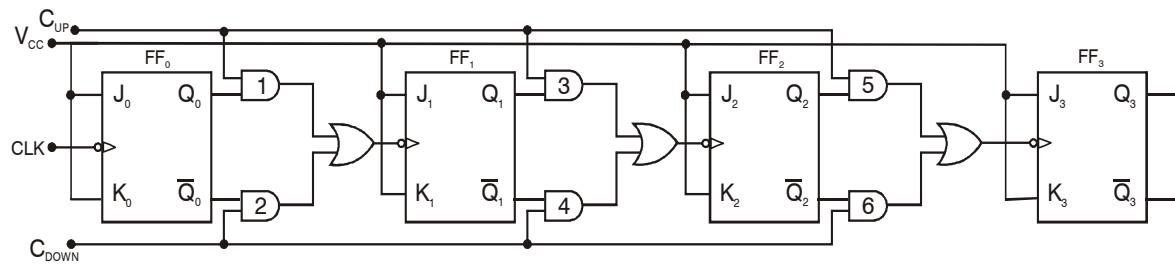


Fig. 3.92 : Asynchronous 4-bit UP-DOWN counter

Three logic gates per stage are required to switch the individual stages from COUNT-UP to COUNT-DOWN mode. The logic gates are used to allow either the non-inverted output or the inverted output of one flip-flop to the clock input of the following flip-flop, depending on the status of the

control inputs. When the C_{UP} line is held at 1 while the C_{DOWN} line is at 0, the lower AND gates (2, 4 and 6) will be disabled and their outputs are zero. So, it will have no effect on the outputs of OR gates. Also, the upper AND gates (1, 3 and 5) will be enabled, i.e., it will allow Q_A to pass through the OR gate and into the clock input of the B flip-flop. Similarly, the Q_B and Q_C output will be gated into the clock input of flip-flops C and D respectively. Thus, as input pulses are applied, the counter will count up and follow a natural binary counting sequence from 0000 to 1111.

With $C_{UP} = 0$, $C_{DOWN} = 1$, the upper AND gates (1, 3 and 5) are disabled and the lower AND gates (2, 4 and 6) are enabled, allowing \bar{Q}_0 , \bar{Q}_1 and \bar{Q}_2 , to pass through to the clock inputs of the following flip-flops. Thus, for this condition, the counter will count down as input pulses are applied.

When the control inputs are both 0 or 1, the counter will not count up or count down because the clock inputs of B , C and D will be held constant at either 0 or 1. The flip-flop (FF_0) will keep toggling because it is always being clocked. These conditions are not normally used.

TABLE 3.52 : Truth Table of 4-bit up-down counter

States	COUNT-UP Mode				States	COUNT-DOWN Mode			
	Q_D	Q_C	Q_B	Q_A		Q_D	Q_C	Q_B	Q_A
0	0	0	0	0	15	1	1	1	1
1	0	0	0	1	14	1	1	1	0
2	0	0	1	0	13	1	1	0	1
3	0	0	1	1	12	1	1	0	0
4	0	1	0	0	11	1	0	1	1
5	0	1	0	1	10	1	0	1	0
6	0	1	1	0	9	1	0	0	1
7	0	1	1	1	8	1	0	0	0
8	1	0	0	0	7	0	1	1	1
9	1	0	0	1	6	0	1	1	0
10	1	0	1	0	5	0	1	0	1
11	1	0	1	1	4	0	1	0	0
12	1	1	0	0	3	0	0	1	1
13	1	1	0	1	2	0	0	1	0
14	1	1	1	0	1	0	0	0	1
15	1	1	1	1	0	0	0	0	0
0	0	0	0	0	15	1	1	1	1

3.29 HDL FOR SEQUENTIAL LOGIC CIRCUITS

3.29.1 S-R Latch

S-R latch using NOR gates is one of the sequential logic circuit. Two NOR gates are cross coupled so that the output of NOR gate 1 is connected to one of the inputs of NOR gate 2 and vice versa. The latch has two inputs *S* (set) and *R* (Reset) and two outputs *Q* and *QN*. **Figure 3.93** shows the *S-R* latch using NOR gates and HDL program for *S-R* latch is given.

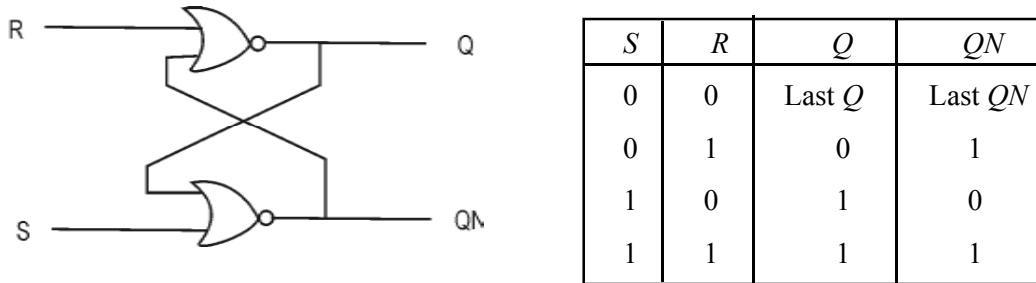


Fig. 3.93 : S-R latch

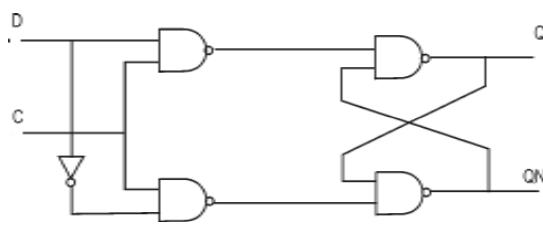
```

library IEEE;
use IEEE . std_logic_1164 . all;
entity srslatch is
    port (S, R : in STD_LOGIC;
          Q, QN : buffer STD_LOGIC);
end srslatch

architecture synth of srslatch is
begin
    QN <= S nor Q ;
    Q <= R nor QN ;
end synth;
```

3.29.2 D-Latch

Figure 3.94 shows a ‘*D*’ latch with truth table. The input conditions (00, 11) of SR latch can be avoided by making them complement of each other. This modified SR latch is known as D-latch. The control input of a ‘*D*’ latch labeled as *C* or CLK or ENABLE (EN). The behavioural program in VHDL for D latch is given below:



<i>C</i>	<i>D</i>	<i>Q</i>	<i>QN</i>
1	0	0	1
1	1	1	0
0	X	last Q	last QN

Fig. 3.94 : D-latch

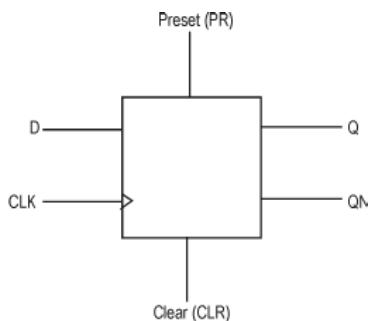
```

library IEEE;
use IEEE . Std_logic_1164 . all;
entity dlatch is
    port (C, D : in STD_LOGIC;
          Q, QN : buffer STD_LOGIC);
end dlatch;

architecture behave of dlatch is
begin
    process (C, D, Q)
        begin
            if (C = '1') then Q <= D ; }> end if;
            QN <= not Q;
        end process;
    end behave;

```

3.29.3 D flip-flop



<i>CLK</i>	<i>D</i>	<i>Q</i>
↑	0	0
↑	1	1
0	X	last Q

Fig. 3.95 : 'D' flipflop

A positive edge triggered D flip-flop combines a pair of D-latches to create a circuit that samples its D-input and changes its Q and QN outputs only at the rising edge of a controlling CLK signal. The D-flipflop shown in **Figure 3.95** has asynchronous inputs that may be used to force the flip-flop to a particular state independent of CLK and D-inputs. These inputs labeled PR(Preset) and CLR(Clear). The VHDL program for a positive edge triggered D-flipflop is,

```

library IEEE;
use IEEE . Std_logic_1164 . all;

entity Dff is
    port (CLK, CLR, D, PR : in STD_Logic;
          Q, QN : out STD_LOGIC);
end Vff;

architecture behave of Dff is
begin
    process (CLK, CLR)
    begin
        if CLR = '1' then Q <= '0' ; QN <= '1';
        elsif CLK event and CLK = '1' then Q <= D ; QN <= not D ;
        end if;
    end process;
end behave;
```

3.29.4 SHIFT REGISTER

The bi-directional shift register allows shifting of data either to the left or to the right side. It can be implemented by using logic gates that enables the transfer of data from one stage to the next stage to the right or to the left, depending on the level of a control line. **Figure 3.96** shows a 4-bit bi-directional shift register. The **RIGHT/LEFT** is the control input signal which allows data shifting either towards right or towards left. If **RIGHT/LEFT** = 1, the shifting of data towards right. If **RIGHT/LEFT** = 0, the shifting of data towards left.

When **RIGHT/LEFT** = 1, gates G_1, G_2, G_3 and G_4 are enabled and the state of the Q output of each flip-flop is passed through the D-input of the **following** flipflop. When a clock pulse arrives, the data are shifted one place to the right.

When **RIGHT/LEFT** = 0, gates G_5, G_6, G_7 and G_8 are enabled and the state of the Q output of each flipflop is passed through the D input of the **preceding** flip-flop. When clock pulse arrives, the data are shifted one place to the left.

The 4 bit type used for the input and output of the shifter is declared in package *shift_types*. This package is used by entity *shifter* to declare ports *DIN* and *DOUT*. Port *CLK*, *LOAD* and *LEFT_RIGHT* are *STD_LOGIC* signals used to control the functions of the shifter.

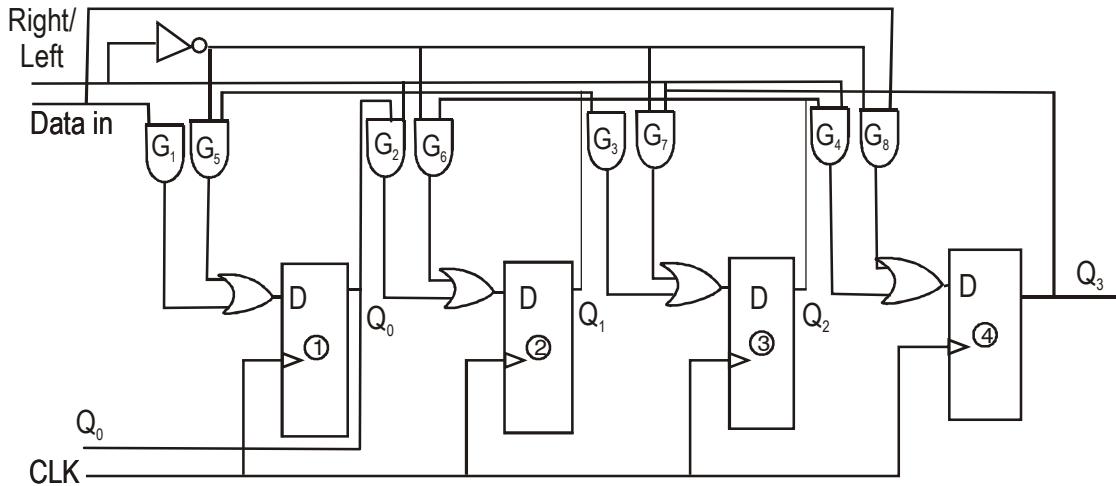


Fig. 3.96 : 4-bit bi-directional Shift Register

The VHDL program for bi-directional shift register is given. In this program two processes are used:

- ◆ *current* process
- ◆ *nxt* process

(i) **Current Process:**

Process *current* is used to keep track of the current value of the shifter. It is a process that has a single *WAIT* statement and a single signal assignment statement. When the *CLK* signal has a rising edge occur, the signal assignment statement is activated and the next calculated value of shifter (*SHIFT_VAL*) is written to the signal that holds the current state of the shift register (*DOUT*).

(ii) **nxt Process:**

Process *nxt* is used to calculate the next value of *SHIFT_VAL* to be written into *DOUT*. *LOAD* is the highest priority input and if equal to '1' causes *SHIFT_VAL* to receive the value of *DIN*. Otherwise, signal *LEFT_RIGHT* is tested to see if the shift register is shifting left or right.

```

library IEEE;
use IEEE . Std_logic_1164 . all;
package SHIFT_TYPES is
    subtype BIT 4 is STD_LOGIC_VECTOR (3 downto 0);
end SHIFT_TYPES;
```

```
use WORK . shift_types.all;
library IEEE;
use IEEE . Std_logic_1164.all;

entity shifter is
    port (DIN : in BIT 4;
          CLK, LOAD, LEFT_RIGHT : in STD_LOGIC;
          DOUT : inout BIT 4);
end shifter;

architecture synth of shifter is
    signal SHIFT_VAL : BIT 4;
begin
    NXT: Process (LOAD, LEFT_RIGHT, DIN, DOUT)
        begin
            if (LOAD = '1') then
                SHIFT_VAL <= }> DIN;
            elseif (LEFT_RIGHT = '0') THEN
                SHIFT_VAL (2 downto 0) <= DOUT (3 downto 1);
                SHIFT_VAL (3) <= '0';
            else
                SHIFT_VAL (3 downto 1) <= DOUT (2 downto 0);
                SHIFT_VAL (0) <= '0';
            end if;
        end process;
    CURRENT : Process
        begin
            wait until CLK' event and CLK = '1';
            DOUT <= SHIFT_VAL;
        end process;
end Synth;
```

3.29.5 COUNTERS

The most popular MSI counter is the 74163, a synchronous 4-bit binary counter. A logic symbol is shown in **Figure 3.97** and the logic diagram is shown in **Figure 3.98**.

This counter can be synchronously preset to any 4-bit binary number by applying the proper levels to the parallel data inputs. When a LOW is applied to the **LOAD** input, the counter will assume the state of the data inputs on the next clock pulse. Thus, the counter sequence can be started with any 4-bit binary number. The state table for this counter is given in **Table 3.53**.

Also, there is an active-LOW clear input (**CLR**), which synchronously resets all flip-flops in the counter. There are two enable inputs, **ENT** and **ENP**. These inputs must be HIGH for the counter to sequence through its binary states. When at least one input is LOW, the counter is disabled. The ripple clock output (**RCO**) goes HIGH, when the counter reaches a terminal count of 15 ($TC = 15$).

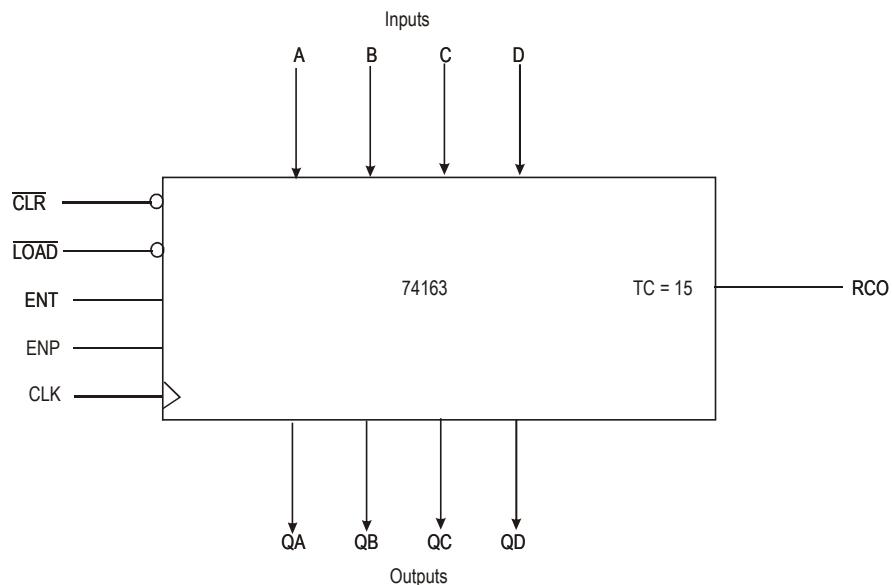


Fig. 3.97 : Logic Symbol for 74163

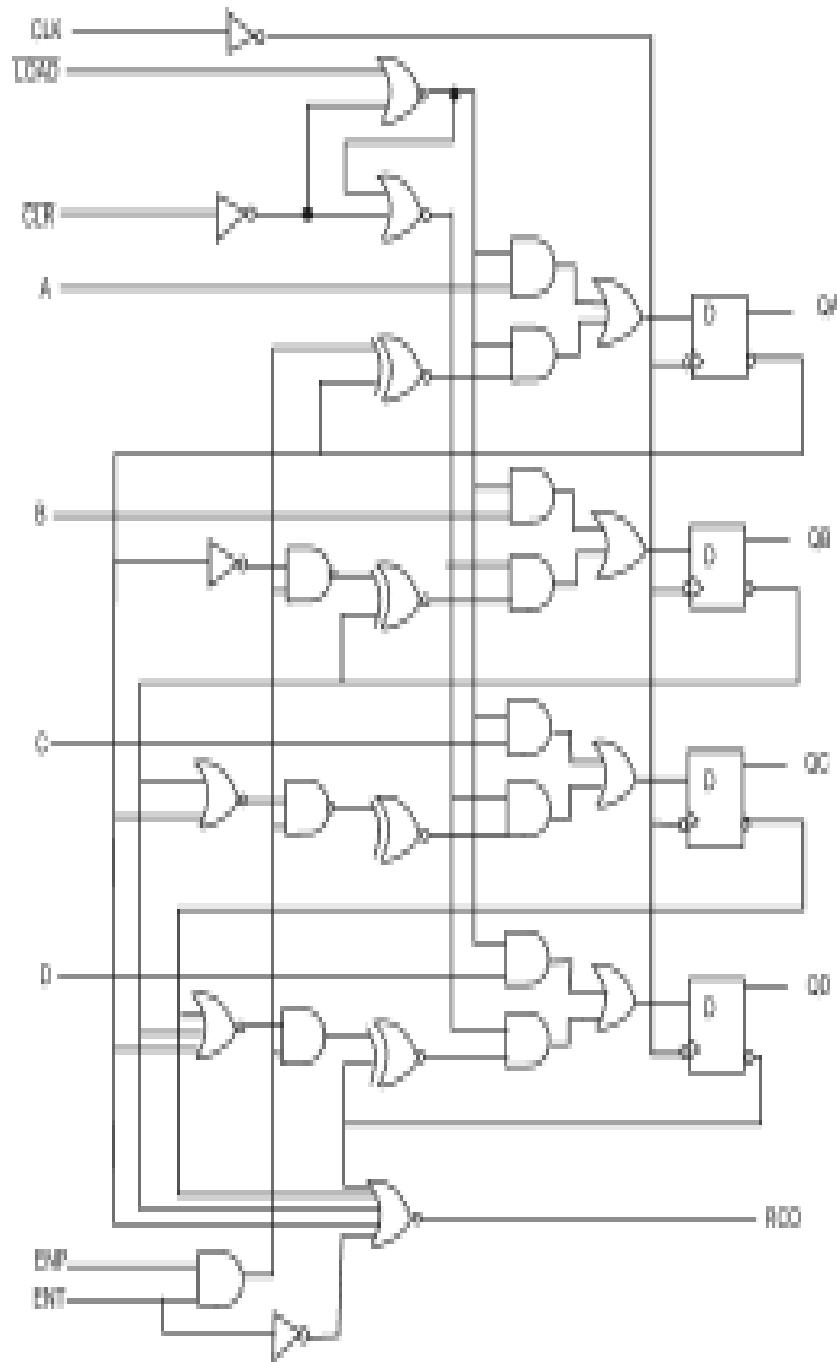


Fig. 3.98 : 4 bit Synchronous binary counter-Logic diagram

TABLE 3.53 : State Table for 4 bit binary counter

Inputs				Current State				Next State			
CLR	LOAD	ENT	ENP	QD	QC	QB	QA	QD _N	QC _N	QB _N	QA _N
0	X	X	X	X	X	X	X	0	0	0	0
1	0	X	X	X	X	X	X	D	C	B	A
1	1	0	X	X	X	X	X	QD	QC	QB	QA
1	1	X	0	X	X	X	X	QD	QC	QB	QA
1	1	1	1	0	0	0	0	0	0	0	1
1	1	1	1	0	0	0	1	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	1
1	1	1	1	0	0	1	1	0	1	0	0
1	1	1	1	0	1	0	0	0	1	0	1
1	1	1	1	0	1	0	1	0	1	1	0
1	1	1	1	0	1	1	0	0	1	1	1
1	1	1	1	0	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0	1	0	0	1
1	1	1	1	1	0	0	1	1	0	1	0
1	1	1	1	1	1	0	0	1	0	0	1
1	1	1	1	1	1	1	0	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1

Each D input is driven by a 2-input multiplexer consisting of an OR gate and two AND gates. The multiplexer output is 0 if the **CLR** input is asserted. Otherwise the top AND gate passes the data input (A, B, C or D) to the output if **LOAD** is asserted. If neither CLR nor **LOAD** is asserted, the bottom AND gate passes the output of an XNOR gate to the multiplexer output.

The XNOR gates perform the counting function. One input of each XNOR is the corresponding count bit (QA, QB, QC or QD); the other input is ‘1’, which complements the count bit, if and only if both enables ENP and ENT are asserted and all of the lower-order count bits are 1. The ripple carry out (RCO) signals indicates a carry from the msb position and is 1 when all of the count bits are 1 and ENT is asserted.

The VHDL program for 4-bit synchronous binary counter is given below. This program uses the **IEEE.Std_logic_arith.all** library, which includes the UNSIGNED type. This library includes definitions of ‘+’ and ‘–’ operators that perform unsigned addition and subtraction on UNSIGNED operands. In this program an internal signal IQ to hold the counter value.

```
library IEEE;
use IEEE . Std_logic_1164 . all;
use IEEE . Std_logic_arith . all;

entity counter is
    port (CLK CLR_L, LOAD_, ENP, ENT : in STD_LOGIC;
          D : in UNSIGNED (3 downto 0);
          Q : out UNSIGNED (3 downto 0);
          RCO : out STD_LOGIC);
end counter;

architecture behave of counter is
signal IQ : UNSIGNED (3 downto 0);
begin
process (CLK, ENT, IQ)
begin
    if(CLK' event and CLK = '1') then
        if CLR_L = '0' then IQ <= (others => '0');
        elsif LOAD_L = '0' then IQ <= D;
        elsif (ENT and ENP) = '1' then IQ <= } > IQ + 1;
        end if;
    end if;
    if (IQ => 15) and (ENT = '1') then RCO <= '1';
    else RCO <= '0';
    end if;
    Q <= IQ ;
end process
end behave;
```

ANNA UNIVERSITY QUESTIONS**PART-A**

1. Define the term triggering the flipflops. **(April 2003)**
2. Distinguish the classification of sequential circuits. **(April 2003)**
3. What is the condition on the JK flipflop to work as a D flipflop? **(April 2003)**
4. Name two sequential switching circuits. **(April 2003)**
5. When is a sequential machine said to be strongly connected? **(April 2003, April 2004)**
6. How will you build a D flipflop from a given JK flipflop? **(April 2003, April 2004)**
7. Derive the characteristic equation of a JK flipflop. **(April 2003)**
8. What are Mealy and Moore machines? **(April 2003)**
9. State the relative merits of series and parallel counters. **(April 2003)**
10. A shift register comprises of JK flipflops. How will you complement the contents of the register? **(April 2003)**
11. Give the excitation table of SR flipflop. **(Nov. 2003)**
12. What is the minimum number of flipflops required to implement a modulo-21 synchronous counter? **(Nov. 2003)**
13. State a limitation of SR flip-flop. **(Nov. 2003)**
14. Convert a D flipflop into a T flip-flop. **(Nov. 2003)**
15. In a serial-in-serial-out shift register has N stages and if the clock frequency is f, what will be the time delay between input and output ? **(Nov. 2003)**
16. What is the minimum number of flipflops needed to design a counter of modulus 60? **(April 2004)**
17. Write the characteristic equation of a JK flipflop. **(April 2004)**
18. Convert an SR flipflop to a D flipflop. **(April 2004)**
19. Derive the characteristic equation of a T flipflop. **(April 2004)**
20. What is the minimum number of flipflops needed to build a counter of modulus Z8? **(April 2004)**
21. What is meant by the term ‘edge triggered’? **(April 2004)**

-
- 22. How many flipflops are required for a counter that will count to 0 to 255? (April 2004)
 - 23. What are shift register counters? What are the types of shift register counter? (Nov. 2004)
 - 24. Explain about state reduction. (Nov. 2004)
 - 25. Derive T FF from JK FF. (Nov. 2004)
 - 26. Draw the timing diagram of 4 bit ring counter. (Nov. 2004)
 - 27. Draw the diagram of a clocked SR flipflop using four NAND gates. (Nov. 2004)
 - 28. When is a counter said to suffer from lockout? (Nov. 2004)
 - 29. Distinguish between combinational logic circuits and sequential logic circuits. (Nov. 2004)
 - 30. A reduced state table has 14 rows. What is the minimum number of flipflops needed to be build the sequential circuit? (Nov. 2004)
 - 31. How will you convert a JK flipflop into a D flipflop. (Nov. 2004)
 - 32. Differentiate between flip-flop and latch. (April 2005)
 - 33. Give the excitation table for JK flip-flop. (April 2005)
 - 34. What is a sequential logic circuit? (April 2005)
 - 35. What are the applications of a shift register? (April 2005)
 - 36. Draw a logic diagram of SR flip-flop. (April 2005)
 - 37. Define a state table. (April 2005)
 - 38. How can a D flip flop be converted into a T flip flop. (Dec. 2005)
 - 39. What are the states of a 4-bit ring counter. (Dec. 2005)
 - 40. What is meant by the term state-reduction ? (Dec. 2005)
 - 41. Draw a 2 bit ripple counter and convert this into a 2 bit ring counter. (Dec. 2005)
 - 42. Define state assignment. (Dec. 2005)
 - 43. How many flip-flops are required to design a mode-7 up-down counter? (Dec. 2005)
 - 44. What is a flip-flop? (May 2006)
 - 45. Differentiate between edge-triggered flip-flop and level triggered flip-flop. (May 2006)
 - 46. What is the drawback of SR flip flop? How is this minimized? (May 2006)
 - 47. What is a sequence generator? (May 2006)
 - 48. What is a synchronous sequential circuit? (May 2006)
 - 49. Draw a scale of 8 ripple counter. (May 2006)
 - 50. Discuss the operation of SR flip flop with the help of the state diagram. (May 2006)
 - 51. What is a binary counter? (Dec. 2006)

-
- 52. What are the models used to represent clocked sequential circuits? **(Dec. 2006)**
 - 53. Write the characteristic equation of JK FF and show how JK FF can be converted into 'T' FF? **(May '07)**
 - 54. Draw the logic diagram of 4 bit universal shift register. **(May 2007)**
 - 55. Convert T-Flipflop into D-Flipflop. **(May 2007)**
 - 56. Draw the logic diagram of Master Slave JK flipflop. **(May 2007)**
 - 57. How can a D flip flop be converted into a T flip flop? **(May 2007)**
 - 58. How many states are there in a 3 bit ring counter? What are they? **(May 2007)**
 - 59. What are the differences between sequential and combinational logic? **(Dec. 2007)**
 - 60. Draw the logic diagram for D-Type Latch. **(Dec. 2007)**
 - 61. Distinguish Mealy and Moore models. **(Dec. 2008)**
 - 62. Write down the characteristic table of JK flip flop. **(Dec. 2008)**
 - 63. With 16 bit shift register, how many timing signals can be generated? **(Dec. 2008)**
 - 64. How many flip flops are required for designing synchronous MOD50 counter? **(May 2009)**
 - 65. What is shift register? List the types. **(May 2009)**
 - 66. Differentiate Moore and Mealy circuit models. **(Dec 2010)**
 - 67. What are the applications of shift registers? **(Dec 2010)**
 - 68. Write the state transition table of J-K Flip - Flop. **(May 2011)**
 - 69. Draw the timing diagram showing the output of a 2 stage synchronous counter with respect to its clock signal. **(May 2011)**
 - 70. Express the next state characteristics of D and SR flip-flops. **(May 2011)**
 - 71. How many flip flops are required for designing syncronous MOD 60 counter?
(Dec 2011)
 - 72. Write down the characteristics equation for JK and T flip flops. **(Dec 2011)**
 - 73. Write the characteristic table and equation of JK flip flop. **(May 2012)**
 - 74. Write any two applications of shift register. **(May 2012)**
 - 75. List any two mechanisms to achieve edge triggering of flip flops. **(Dec 2012)**
 - 76. What is a ring counter? **(Dec 2012)**
 - 77. Derive the characteristic equation of a JK-flipflops. **(May 2013)**
 - 78. What is a Mealy circuit? **(May 2013)**

PART-B

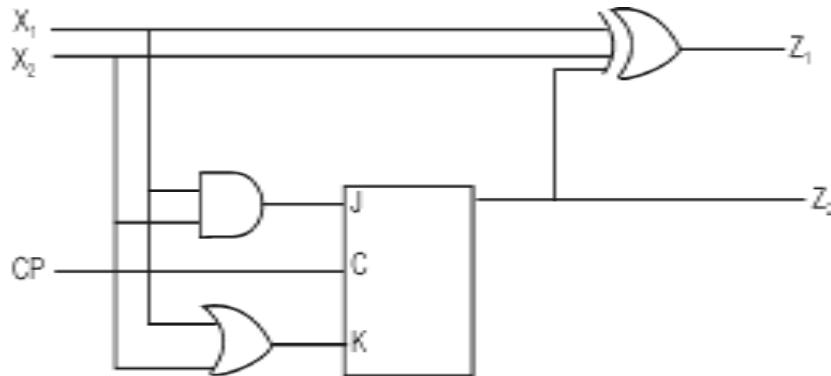
1. A sequential circuit has two JK flipflops A and B , the inputs, X and Y and one output Z . The flipflop input function and the circuit output functions are as follows:

$$JA = BX + \bar{B}\bar{Y} \quad KA = B X \bar{Y}$$

$$JB = \bar{A}X \quad KB = A + X\bar{Y}$$

- (i) Draw the logic diagram, (ii) Tabulate the state table
 (iii) Derive the next state equation for A and B. (16) (April 2003)

2. Analyze the given circuit shown in **Figure**. Obtain the state table and the state diagram and determine the function of the circuit. (16) (April 2003)



3. Describe the working of a BCD ripple counter with next diagram. (8) (April 2003)
 4. (i) Explain the working of Master/Slave JK flipflop. (8) (April 2003)
 (ii) Design and explain the working of an up-down ripple counter. (8) (April 2003)
 5. Design and explain the working of a synchronous mod-3 counter. (16) (April 2003)
 6. (i) Using JK flipflops, design a parallel counter which counts in the sequence 000, 111, 101, 110, 001, 010, 000 (16)

(ii) Minimize the following state table:

(16) (April 2003)

PS	NS, Z	
	x	
	0	1
A	A, 0	D, 1
B	C, 1	D, 0
C	B, 0	A, 1
D	D, 1	A, 1
E	D, 1	A, 1
F	D, 0	A, 0
G	D, 1	A, 1
H	D, 1	C, 1

7. Using D flipflops, design a synchronous counter which counts in the sequence,

000, 001, 010, 011, 100, 101, 110, 111, 000.

(16) (April 2003)

8. Convert the following Mealy machine into a Moore machine:

(8) (April 2003)

PS	NS, Z			
	$x_1 x_2$			
	00	01	11	10
A	A, 0	A, 1	B, 0	A, 1
B	A, 1	B, 0	B, 1	B, 0

9. Using JK flipflops, design a synchronous sequential circuit having one input and one output. The output of the circuit is a 1 whenever three consecutive 1's are observed. Otherwise the output is zero.

(16) (April 2003)

10. (i) Realize a JK flipflop using SR flipflop. (6)

(ii) Realize a SR flipflop using NAND gates and explain its operation. (10) (April 2003)

11. Explain the various steps in the analysis of synchronous sequential circuits with suitable example. (8) (Nov. 2003)

12. Draw the logic diagram of a 4 bit up-down ripple counter using JK flipflop and explain its operation with timing diagram. (16) (Nov. 2003)

13. Minimize the state table shown given below:

(Nov. 2003)

Present State	Next State, Z(output)	
	Input	
	X = 0	X = 1
A	B, 0	C, 0
B	B, 0	D, 0
C	B, 0	C, 0
D	E, 1	C, 0
E	B, 0	D, 0

14. (i) Design and explain the working of mod-7 counter. (8)

(ii) Write notes on state minimisation. (8) (Nov. 2003)

15. Explain the working of master-slave JK flip-flop. (6) (Nov. 2003)

16. Derive the characteristic equation of a SR flipflop. (8) (April 2004)

17. Minimize the following state table. (14) (April 2004)

PS	NS, Z	
	X	
	0	1
A	B, 0	C, 0
B	D, 0	C, 0
C	B, 0	G, 0
D	H, 0	C, 1
G	B, 1	K, 0
H	H, 0	C, 0
K	B, 0	K, 0

18. Using positive edge triggering SR flipflops designs a counter which counts in the following sequence:

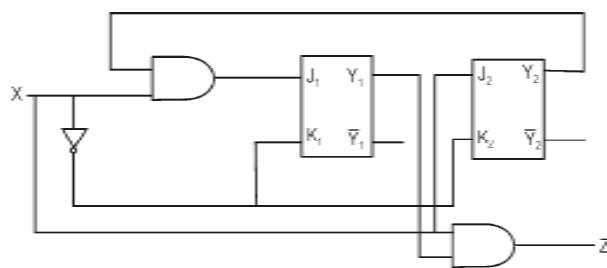
000, 111, 110, 101, 100, 011, 010, 001, 000 (16) (April 2004)

19. (i) Draw a 4-stage switch tail ring counter.

(ii) Show the count sequence

(iii) What is the modification to be used to prevent lock out? (16) (April 2004)

20. Design a synchronous counter with states 0, 1, 2, 3, 0, 1 ... using JK flip-flops. (16) (April 2004)
21. Using SR flip-flops, design a synchronous counter which counts in the sequence 000, 111, 101, 110, 001, 010, 000 ... (16) (April 2004)
- 22.(i) Explain the meaning of Mealy machines and Moore machines. (6)
- (ii) Show that if a sequential machine is strongly connected, then it is reversible but that the converse is not always true. (10) (April 2004)
23. Consider the following synchronous sequential circuit.



Determine its state table. What does the circuit do? (16) (April 2004)

24. Explain the operation of JK and clocked JK flip-flops with suitable diagrams. (16) (Apr. 2004)
25. Design a binary counter using T flip-flops to count pulses in the following sequences:
- (i) 000, 001, 010, 011, 100, 101, 110, 111, 000
(ii) 000, 100, 111, 010, 011, 000 (16) (April 2004)
26. Design a four-state down counter using type T design procedures. (8) (Nov. 2004)
27. Design a 4 bit synchronous 8421 decade counter with ripple carry. (16) (Nov. 2004)
28. Explain the parallel load-serial out data transfer operation in a 4 bit shift register. (16) (Nov. 2004)
29. Explain the working of BCD ripple counter with timing diagrams. (16) (Nov. 2004)
30. Using RS flip-flops, design a parallel counter which counts in the sequence 101, 110, 001, 010, 000, 111, 101, (12) (Nov. 2004)
31. (i) What is an incompletely specified sequential machine? (3)

(ii) Minimize the following state table.

(13) (Nov. 2004)

Present state	Next state	
	Output	
	0	1
A	D, 0	C, 1
B	E, 1	A, 1
C	H, 1	D, 1
D	D, 0	C, 1
E	B, 0	G, 1
F	H, 1	D, 1
G	A, 0	F, 1
H	C, 0	A, 1
I	G, 1	H, 1

32. Distinguish between Mealy machines and Moore machines and with the help of a suitable example, explain how a Mealy machine can be converted to a Moore machine. (12) (Nov. 2004)

33. (i) When is a sequential machine said to be strongly connected? (3)

(ii) With the help of a suitable example, describe the shift register realization of a sequence detector. (13) (Nov. 2004)

34. (i) What is the characteristic equation of a D flip-flop ? (4)

(ii) Using JK flip-flops, design a parallel counter which counts in the sequence.

101, 110, 001, 010, 000, 111, 101, ... (12) (Nov. 2004)

35.(i) Explain the significance of the following statement: "There can be a number of sequential circuits implementing the same sequential machine". (4)

(ii) A synchronous sequential machine produces an output of 1 when exactly two 1's are followed by one 0. Once the output becomes a 1, it remains so until the sequence 10 is received, when the output returns to 0. Derive the minimized state table for the machine. (12) (Nov. 2004)

36. Design a synchronous 3 bit gray code up counter with the help of excitation table. (12) (Apr. 2005)

37. Design a MOD 10 synchronous counter using JK flipflops. Write excitation table and state table. (16) (April 2005)

38. (i) Compare Moore and Mealy circuits. (4)

(ii) Draw and explain the block diagram of Mealy circuit. (12) (April 2005)

39. Explain the different types of triggering used in flipflop. (4) (April 2005)

40. Design a modulus 5 synchronous counter using JK flip-flop and implement it. Construct its timing diagram. (16) (April 2005)

41. Determine a minimal state table equivalent furnished below:

(6)

PS	NS, Z	
	X = 0	X = 1
1	1, 0	1, 0
2	1, 1	6, 1
3	4, 0	5, 0
4	1, 1	7, 0
5	2, 0	3, 0
6	4, 0	5, 0
7	2, 0	3, 0

(April 2005)

42. When do we prefer ASM charts? What are its basic elements?

(5) (April 2005)

43. Explain the working of a master-slave JK flip-flop?

(8) (April 2005)

- 44.(i) Design and explain the working of a mod-11 counter.

(8)

- (ii) Explain the techniques of state minimisation using an example.

(8) (April 2005)

45. Draw a six stage ring counter and explain its operation. Mention about the use of presetting the counter.

(16) (Dec. 2005)

46. Draw a 5 flip flop shift counter, its truth table and waveforms. Explain its operation as a decade counter.

(16) (Dec. 2005)

47. Design a synchronous mod-8 down counter and implement it.

(16) (Dec. 2005)

- 48.(i) Realize SR flip-flop using NOR gates and explain its operation.

(8) (Dec. 2005)

- (ii) Explain the operation of JK master slave flip-flop with suitable diagrams.

(8) (Dec. 2005)

49. Minimize the state table shown given below:

(10)

Present State	Next State, Z (output)	
	X (Input)	
	0	1
A	A, 0	B, 0
B	C, 0	D, 0
C	A, 0	D, 0
D	E, 0	F, 1
E	A, 0	F, 1
F	G, 0	F, 1
G	A, 0	F, 1

50. (i) Explain the working of a ripple counter.

(8)

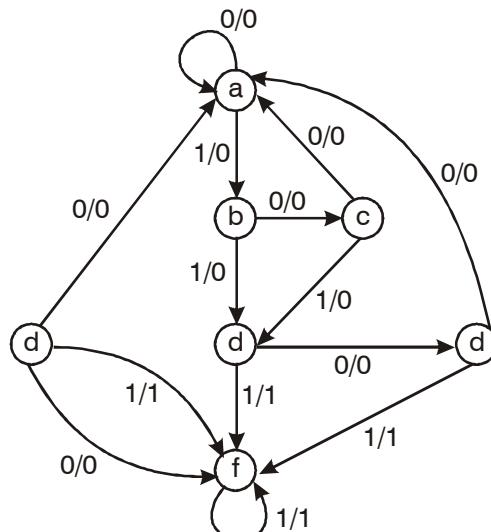
- (ii) Write brief notes on four applications of shift registers.

(8) (Dec. 2005)

51. Design a mod-5 synchronous counter using JK flip-flop.

(16) (Dec. 2005)

52. Design a negative edge triggered T flip-flop. The circuit has two inputs, T(toggle) and C(clock) and outputs Q and \bar{Q} . The output state is complemented if T = 1 and the clock C changes from 1 to 0. Otherwise under any other input condition the output Q remains unchanged. (16) (May 2006)
53. Design and explain the working of a mod-9 counter. (10) (May 2006)
54. Explain the working of master-slave JK flip-flop. State its merit. (8) (May 2006)
55. (i) Draw an asynchronous decade counter and explain its operation drawing heat waveforms. (8)
(ii) Draw a 3 bit reversible counter and explain its operation. (8) (May 2006)
56. (i) Draw a 4 bit serial in serial out shift register and draw as waveforms. (8) (May 2006)
(ii) Draw a 4 bit parallel in serial out shift register and briefly explain. (8) (May 2006)
57. Design a synchronous counter that goes through the sequence 2, 6, 1, 7, 5, 4 and repeat. Use JK flip. (16) (May 2006)
58. (i) Design a mod 5 asynchronous counter. Draw the waveforms. (8) (May 2006)
(ii) Using a code converter at the output of the above mod 5 counter to convert the up counter into down counter. (8) (May 2006)
59. Design a 3 bit binary Up-Down counter. (Dec. 2006)
60. (i) Summarize the design procedures for synchronous sequential circuit. (6) (Dec. 2006)
(ii) Reduce the following state diagram. (Dec. 2006)

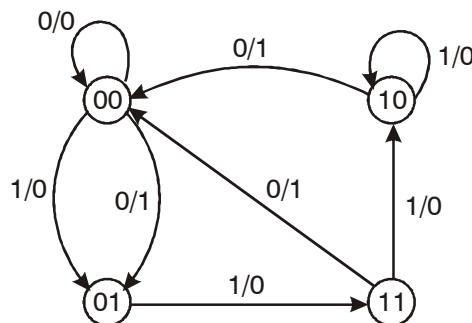


61. Design a synchronous counter which counts in the sequence 0,2,6,1,7,5,0 using D FFs. Draw the logic diagram and state diagram. (16) (May 2007)

62. Reduce the given state table using implication chart (tabular method) (10)

	x=0	x=1	z
A	B	D	1
B	D	F	1
C	D	A	0
D	D	E	0
E	B	C	1
F	C	D	0

- (i) Realise D and T flipflop using JK flipflops. (8)
 (ii) Write the excitation table of SR, JK, D and T flipflops. (8) (May 2007)
63. Design a synchronous counter that counts in the sequence 0,1,3,5,7,4,2,0,6using D flipflops. (16) (May 2007)
64. (i) Explain the operation of D-Type Edge Triggered Flip-Flop. (8) (May 2007)
 (ii) Write HDL code for the following Mealy state diagram. (8) (May 2007)



65. What are the general capabilities of universal shift register? And write the HDL code for the same. (16) (May 2007)
66. Construct a full subtractor circuit and write a HDL program module for the same.
- (i) Compare synchronous with Asynchronous counters. (8) (Dec. 2007)
 (ii) Explain the behavioral Model with suitable example. (8) (Dec. 2007)
67. (i) A positive edge triggered flip-flop has two inputs D_1 and D_2 and a control input that choose between the two. Write an HDL behavioral description of this flip-flop. (8) (Dec. 2007)
 (ii) Construct and explain 4 stage Johnson counter. (8) (Dec. 2007)
68. (a) (i) Draw the logic diagram of a 4-bit shift register with four D flip-flops and our 4x1 multiplexes with mode selection inputs s_1 and s_0 . The register operates as follows (12) (Dec. 2008)

s1	s0	Register Operation
0	0	No change
0	1	Complement
1	0	Clear to 0
1	1	Load parallel data

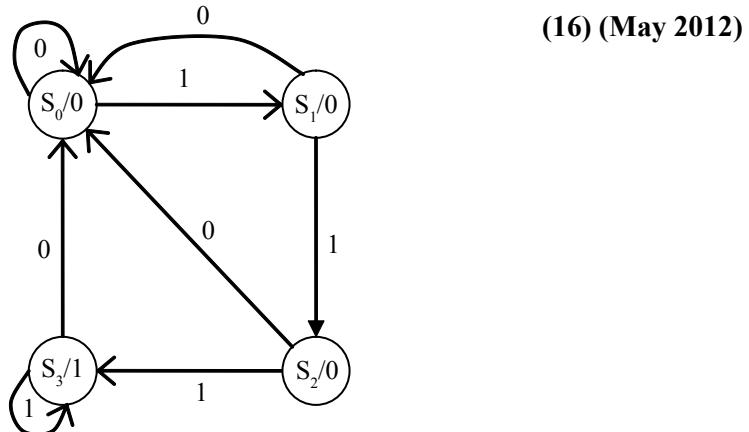
- (ii) Construct a JK flip-flop, a 2x1 multiplexer and an inverter. **(4) (Dec. 2008)**
69. (i) Explain the difference among a truth table, a state table, a characteristic table, and an excitation table. Explain the difference among a Boolean equation, a state equation, a characteristic equation and a flip-flop input equation. **(6) (Dec. 2008)**
- (ii) A sequential circuit has two JK flip-flops A and B, two inputs X and y and one output z. The flip-flop input equations and circuit output equations are
- $$\begin{array}{ll} J_k = Bx + B'y' & K_A = B'xy' \\ J_n = A'x & K_B = A + xy' \\ z = Ax'y + Bx'y' \end{array}$$
- (1) Draw the logic diagram of the circuit.
 (2) Tabulate the state.
 (3) Draw the state diagram. **(10) (Dec. 2008)**
70. Design a synchronous BCD counter using JK flip flop. **(16) (May 2009)**
71. (i) Write a verilog description for JK negative edge triggered flip flop with clock CLK. **(10)**
 (ii) Explain the characteristic table for JK flip flop. **(6) (May 2009)**
72. Design a clocked sequential machine using T flip flops for the following state diagram. Use state reduction if possible. Also use straight binary state assignment. **(16) (Dec 2010)**

73. Using RS-FFs design a parallel counter which counts in the sequence.
 000, 111, 101, 110, 001, 010, 000, ... **(16) (Dec 2010)**
74. Design a T flip flop from logic gates. **(16) (Dec 2010)**
75. A synchronous counter with four JK flip-flops has the following connections :
 $J_A = K_A = 1,$
 $J_B = Q_A Q_D, K_B = Q_A$
 $J_C = K_C = Q_A Q_B$
 $J_D = Q_A Q_B Q_C, \text{ and } K_D = Q_A$

Determine the modulus n of the counter, and draw the output waveforms of the same. **(16) (May 2011)**

76. Design a synchronous counter using JK flip-flops to count the following sequence: "1- 3 - 15 - 5 - 8 - 2 - 0 - 12 - 6 - 9". **(16)(May 2011)**

77. Design a MOD 16 up counter using JK Flip flops. (16) (Dec 2011)
78. With suitable example explain state reduction and state assignment. (16) (Dec 2011)
79. (i) Draw a 4-bit ripple counter with D flip flops. (6) (May 2012)
(ii) Write the HDL for the above circuit. (10) (May 2012)
80. Design the sequential circuit specified by the state diagram using JK flip flop. (16) (May 2012)



81. i) Design a 3-bit binary counter. (10) (Dec 2012)
ii) Write the HDL description of T flip flop and JK flip flop from D flip flop and gates. (6) (Dec 2012)
82. Design a sequential circuit using RS flip flops for the state table given below using minimum number of flip flops. (16) (Dec 2012)

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

83. Using D flip flops design a synchronous counter which counts in the sequence, 000, 001, 010, 011, 100, 101, 110, 111, 000. **(16) (May 2013)**
84. Design a shift register using JK flip flops. **(16) (May 2013)**

UNIT – IV

ASYNCHRONOUS SEQUENTIAL LOGIC

- **Analysis and Design of Asynchronous Sequential Circuits**
- **Reduction of State and Flow Tables**
- **Race-free State Assignment**
- **Hazards**

ASYNCHRONOUS SEQUENTIAL LOGIC

4.1 INTRODUCTION

In synchronous sequential circuits, memory elements are clocked flipflops. In asynchronous sequential circuits, memory elements are either unclocked flipflops (latches) or time delay elements.

Asynchronous means without a synchronizing clock to control the state transitions of a sequential circuit. The inputs and present state change after the internal delay of the circuit elements. Because no synchronizing clock is used, asynchronous circuits are usually faster than synchronous circuits. The comparison between synchronous and asynchronous sequential circuits is given in **Table 4.1**.

TABLE 4.1: Comparison between Synchronous and Asynchronous Sequential Circuits

<i>Sl. No.</i>	<i>Synchronous Sequential Circuits</i>	<i>Asynchronous Sequential Circuits</i>
1.	Memory elements are clocked flipflops.	Memory elements are either unclocked flipflops or time delay elements.
2.	The operating speed of clock depends on time delays involved. Therefore synchronous circuits can operate slower than asynchronous.	Because of absence of clock, it can operate faster than asynchronous sequential circuits.
3.	The change in input signals can affect memory elements upon activation of clock signal.	The change in inputs signals can affect memory elements at any instant of time.
4.	Easier to design.	More difficult to design.

4.2 TYPES OF ASYNCHRONOUS SEQUENTIAL CIRCUITS

Asynchronous Sequential Circuits are typically classified into two main groups:

- ◆ Fundamental Mode Asynchronous Sequential Circuits
- ◆ Pulse Mode Asynchronous Sequential Circuits

4.2.1 Fundamental Mode

In the fundamental mode, only one input is allowed to change at a time and the inputs are considered to be **levels**, (0 or 1). Input level changes can occur no faster than allowed by the slowest propagation path in the circuit.

4.2.2 Pulse Mode

In the pulse mode, only one input is allowed to change at a time and inputs are considered to **pulse** (false-true-false) and the input pulses must be long enough to initiate a state change but not so wide that the input is still true after a new state is reached.

The input signals i.e., level inputs for fundamental mode and pulse inputs for pulse mode are shown in **Figure 4.1**.

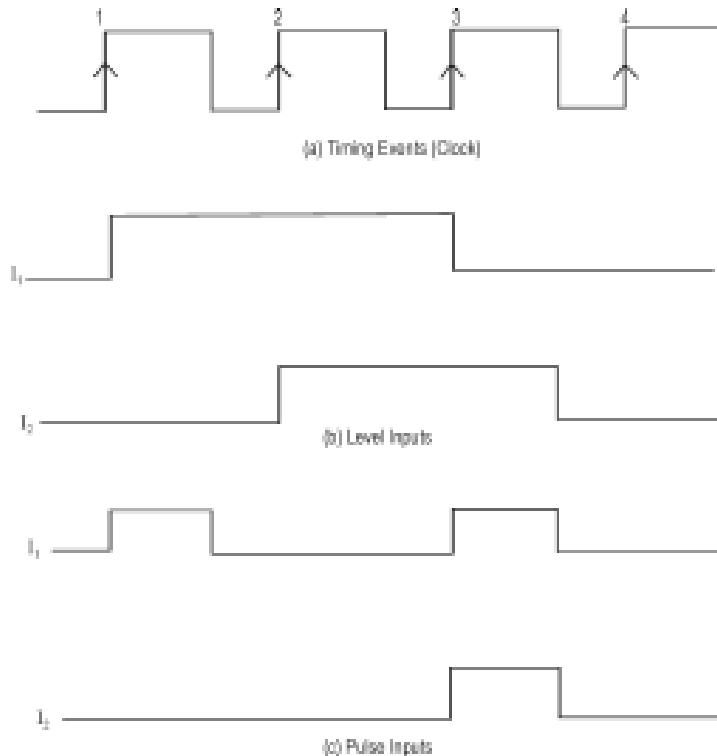


Fig. 4.1: Level and Pulse inputs

4.3 TRANSITION TABLE

A state table with binary assignment is called transition table. Transition table of asynchronous sequential circuits similar to the state table used for synchronous sequential circuits. It is constructed which shows the next states of the flipflops as a function of the present state and inputs.

4.4 FLOW TABLE

During the design of asynchronous sequential circuits, it is more convenient to name the states by letter symbols without making specific reference to their binary values. Such a table is called flow table.

A flow table is similar to a transition table except that the internal states are symbolized with letters rather than binary numbers. The flow table also includes the output values of the circuit for each state table.

4.5 PRIMITIVE FLOW TABLE

A primitive flow table is a special case of flow table. It is defined as a flow table which has exactly one stable state for each row in the table.

4.6 ANALYSIS OF FUNDAMENTAL MODE ASYNCHRONOUS

Sequential Circuits

The analysis of asynchronous sequential circuits consists of obtaining a table or a diagram that describes the sequence of internal states and outputs as a function of changes in the input variables.

Analysis Procedure

The procedure for obtaining a transition table from the given circuit diagram is as follows:

1. Determine all feedback loops in the circuit.
2. Designate the output of each feedback loop with variable Y_1 and its corresponding inputs y_1, y_2, \dots, y_k , where k is the number of feedback loops in the circuit.
3. Derive the Boolean functions of all Y 's as a function of the external inputs and the y 's.
4. Plot each Y function in a map, using y variables for the rows and the external inputs for the columns.
5. Combine all the maps into one table showing the value of $Y = Y_1, Y_2, \dots, Y_k$ inside each square.
6. Circle all stable states where $Y = y$. The resulting map is then the transition table.

Example 4.1: Derive the transition table and primitive flow table for the fundamental mode asynchronous sequential circuit shown in **Figure 4.2**.

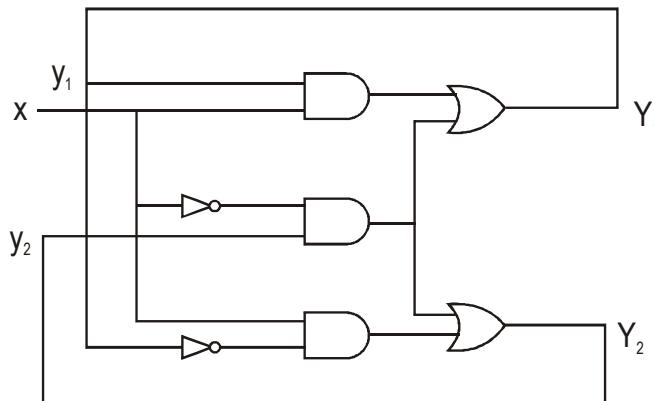


Fig. 4.2: Fundamental mode sequential circuit

Solution: (i) **Boolean Expressions:** Boolean expressions for the logic diagram are:

$$Y_1 = xy_1 + \bar{x}y_2$$

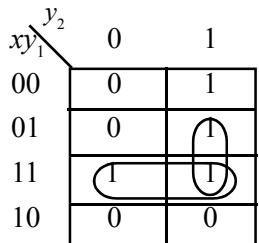
$$Y_2 = x\bar{y}_1 + \bar{x}\bar{y}_2$$

where Y_1, Y_2 = Excitation variables = Next State
 y_1, y_2 = Secondary variables = Present State

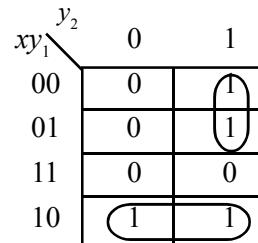
(ii) Maps for Y_1 and Y_2

TABLE 4.2: Next-State Table

x	y_1	y_2	xy_1	$\bar{x}y_2$	$\bar{x}y_1$	$Y_1 = xy_1 + \bar{x}y_2$	$Y_2 = x\bar{y}_1 + \bar{x}y_2$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	1
0	1	0	0	0	0	0	0
0	1	1	0	1	0	1	1
1	0	0	0	0	1	0	1
1	0	1	0	0	1	0	1
1	1	0	1	0	0	1	0
1	1	1	1	0	0	1	0



(a) Map for $Y_1 = xy_1 + x\bar{y}_2$



(b) Map for $Y_2 = x\bar{y}_1 + \bar{x}y_2$

Fig. 4.3: Maps for y_1 and y_2

- (iii) Transition Table:** The transition table shown in **Table 4.3** is obtained from the maps by combining the binary values in corresponding squares. i.e., $Y = Y_1 Y_2$. The first bit of Y is obtained from the value of Y_1 and the second bit of Y is obtained from the value of Y_2 in the same square position. For a state to be stable, the value of Y must be the same as that of $y = y_1 y_2$. Those entries in the transition table where $Y = y$ are circled to indicate a stable condition. An uncircled entry represents an unstable state.

TABLE 4.3 : Transition Table

xy_1	y_2	0	1
00	00	00	01
01	11	01	01
11	11	10	10
10	00	00	10

unstable state → stable state

(iv) Primitive Flow Table

A flow table is similar to a transition table except that the internal states are symbolized with letters like a, b, c, S_1, S_2 , etc. rather than binary numbers. In this example, we assign the following binary values to the states:

$$a = 00$$

$$b = 01$$

$$c = 11$$

$$d = 10$$

A primitive flow table is shown in **Table 4.4**; it has only one stable state in each row.

TABLE 4.4 : Primitive flow table

	y_2	0	1
xy_1			
a		(a)	b
b		c	(b)
c		(c)	d
d		a	(d)

Example 4.2: Derive the transition table and primitive flow table for the given circuit shown in **Figure 4.4**.

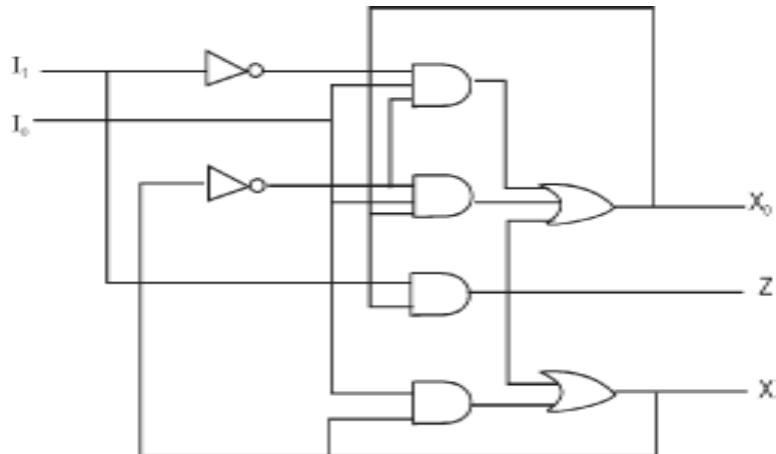


Fig. 4.4 : Sequential circuit

Solution:

Excitation variables (Next State) = X_0, X_1

Secondary variables (Present State) = I_0, I_1 ; Output = Z

Boolean Expressions

$$X_0(t+1) = X_0 I_1 + \bar{X}_1 \bar{I}_1 I_0 + \bar{X}_1 X_0 I_1$$

$$X_1(t+1) = X_0 I_1 + I_0 X_1$$

$$Z = X_0 I_1$$

Table 4.5 illustrates the present state, next state and output variables.

TABLE 4.5 : Next State Table

X_1	X_0	X_1	I_0	$X_1(t+1) = X_0 I_1 + I_0 X_1$	$X_0(t+1) = \bar{X}_1 \bar{I}_1 I_0 + \bar{X}_1 X_0 I_0 + X_0 I_1$	$Z = X_0 I_1$
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	0	1	0	1	0
0	1	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	0	0	0	0
1	0	0	1	1	0	0
1	0	1	0	0	0	0
1	0	1	1	1	0	0
1	1	0	0	0	0	0
1	1	0	1	1	0	0
1	1	1	0	1	1	1
1	1	1	1	1	1	1

Map for $X_1(t+1)$

		$I_1 I_0$	00	01	11	10
		$X_1 X_0$	00	01	11	10
I_1	I_0	00	0	0	0	0
		01	0	0	1	1
I_1	I_0	11	0	1	1	1
		10	0	1	1	0

Map for $X_0(t+1)$

		$I_1 I_0$	00	01	11	10
		$X_1 X_0$	00	01	11	10
I_1	I_0	00	0	1	0	0
		01	0	1	1	1
I_1	I_0	11	0	0	1	1
		10	0	0	0	0

Transition Table:

TABLE 4.6 : Transition Stable

$I_1 I_0$	00	01	11	10	
$X_1 X_0$	00	(00)	01	(00)	(00)
	01	00	(01)	11	11
	11	00	10	(11)	(11)
	10	00	(10)	(10)	00

Primitive Flow Table:

TABLE 4.7 : Primitive Flow Table

$I_1 I_0$	00	01	11	10	
$X_1 X_0$	a	(a)	b	(a)	(a)
	a	a	(b)	c	c
	c	a	d	(c)	(c)
	d	a	(d)	(d)	a

4.7 ANALYSIS OF PULSE MODE ASYNCHRONOUS SEQUENTIAL CIRCUITS

Pulse mode asynchronous sequential circuits rely on input pulses rather than levels. They allow only one input variable to change at a time. They can be implemented by employing a basic flip-flop commonly referred to as an SR Latch. We will first explain the operation of the SR latch with NOR gates and NAND gates. We will then proceed to give examples of analysis of pulse mode asynchronous sequential circuits that employ SR latches.

The SR latch has two inputs S and R and two cross-coupled NOR gates or two cross-coupled NAND gates. The SR latch with NOR gates is shown in **Figure 4.5**. In order to analyze the circuit by the transition-table method, redraw the circuit, as shown in **Figure 4.6**.

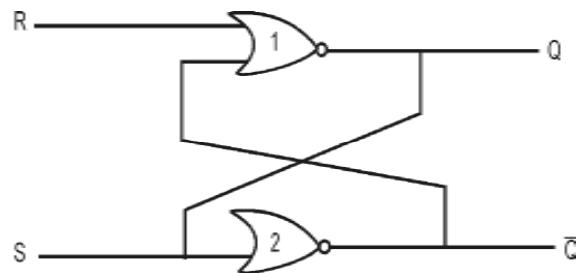


Fig. 4.5 : SR latch

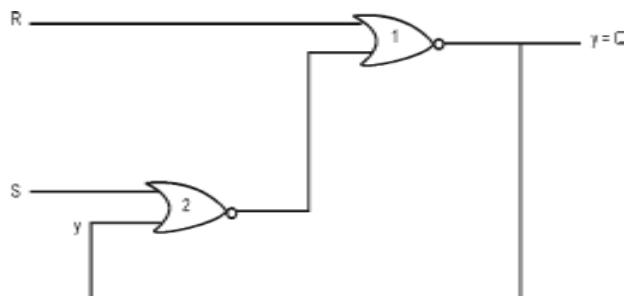


Fig. 4.6 : Circuit Showing feedback

The Boolean function for the output is,

$$\begin{aligned}
 Y &= \left((\overline{S+y}) + R \right) \\
 &= (S+y) \overline{R} \\
 &= S\overline{R} + \overline{R}y
 \end{aligned}$$

TABLE 4.8 : Output Values

<i>S</i>	<i>R</i>	<i>y</i>	<i>S</i> <i>R</i>	<i>R</i> <i>y</i>	<i>Y</i> = <i>S</i> <i>R</i> + <i>R</i> <i>y</i>
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	1	0	0	0

TABLE 4.9 : Transition table

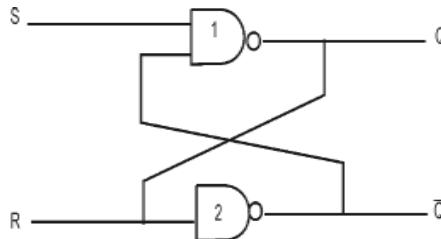
$SR \backslash y$	0	1
00	0	1
01	0	0
11	0	0
10	1	1

$$S\bar{R} + SR = S(\bar{R} + R) = S$$

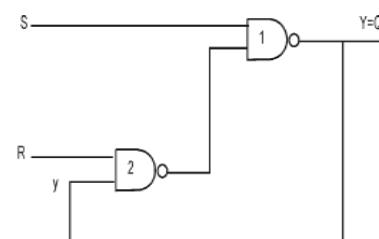
$\therefore S\bar{R} = S$ when $SR = 0$

$$Y = S\bar{R} + \bar{R}y = S + \bar{R}y$$

The SR latch with NAND gates is shown in **Figure 4.7(a)** and **(b)**.



(a) Cross coupled circuit



(b) Circuit showing feedback

Fig. 4.7: SR Latch

$$\text{Output } Y = (\bar{R}y)S = Ry + \bar{S} \text{ with } \bar{S}\bar{R} = 0$$

The values of output Y are shown in **Table 4.10** and the transition table is shown in **Table 4.11**.

TABLE 4.10: Output Values

TABLE 4.11 : Transition Table

S	R	y	Ry	$Y = Ry + \bar{S}$
0	1	0	1	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$SR \backslash y$	0	1
00	1	1
01	1	1
11	0	1
10	0	0

SR latch using NOR gates, $Y = S + \bar{R}y$ with $SR = 0$

SR latch using NAND gates, $Y = \bar{S} + Ry$ with $\bar{S}\bar{R}=0$

Example 4.3: Derive the transition table for the pulse mode asynchronous sequential circuit shown in Figure 4.8.

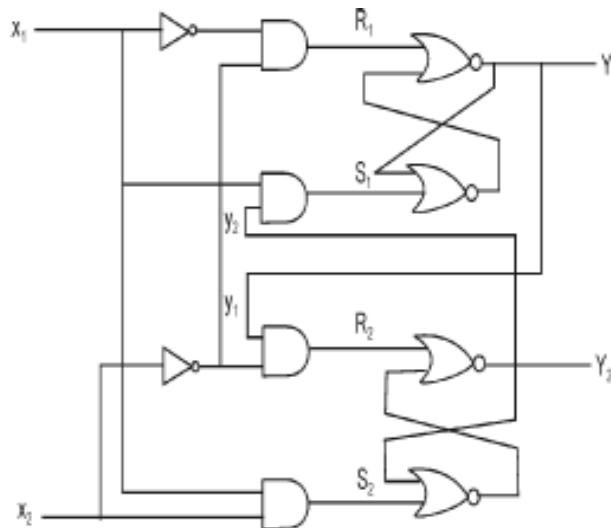


Fig. 4.8 : Pulse Mode sequential circuit

Solution:

Procedure

1. Derive the Boolean functions for S_1, R_1, S_2 and R_2 inputs.
2. Check whether $SR = 0$ for each NOR latch or whether $\bar{S}\bar{R}=0$ for each NAND latch. If this condition is not satisfied, there is a possibility that the circuit may not operate properly.
3. Evaluate $Y = S + \bar{R}y$ for each NOR latch or

$$Y = \bar{S} + Ry \text{ for each NAND latch.}$$

4. Construct maps for Y_1 and Y_2 .
5. Plot the value of $Y = Y_1 Y_2$ in the map. Circle all stable states where $Y = y$. The resulting map is the transition table.

Step 1: Boolean functions for S and R inputs in each table:

$$S_1 = x_1 y_2 \quad R_1 = \bar{x}_1 \bar{y}_2$$

$$S_2 = x_1 x_2 \quad R_2 = \bar{x}_1 y_2$$

Step 2: Check whether the condition $SR = 0$ is satisfied to ensure proper operation of NOR latches.

$$S_1 R_1 = x_1 y_2 \bar{x}_1 \bar{x}_2 = x_1 \bar{x}_1 y_2 x_2 = 0 \quad (\because x_1 x_1 = 0)$$

$$S_2 R_2 = x_1 y_2 \bar{x}_2 y_1 = x_2 \bar{x}_2 x_1 y_1 = 0 \quad (\because x_2 \bar{x}_2 = 0)$$

Step 3: Evaluate Y_1 and Y_2 .

$$\begin{aligned} Y_1 &= S_1 + \bar{R}_1 y_1 \\ &= x_1 y_2 + (\bar{x}_1 \bar{x}_2) y_1 = x_1 y_2 + (x_1 + y_2) y_1 \\ &= x_1 y_2 + x_1 y_1 + x_2 y_1 \end{aligned}$$

$$\begin{aligned} Y_2 &= S_2 + \bar{R}_2 y_2 \\ &= x_1 x_2 + (\bar{x}_2 y_1) y_2 = x_1 x_2 + (x_2 + \bar{y}_1) y_2 \\ &= x_1 x_2 + x_1 y_2 + \bar{y}_1 y_2 \end{aligned}$$

TABLE 4.12

x_1	x_2	y_1	y_2	$x_1 y_2$	$x_1 y_1$	$x_2 y_1$	$x_1 x_2$	$x_2 y_2$	$\bar{y}_1 y_2$	Y_1	Y_2
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	1
0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	1	1	0	1
0	1	1	0	0	0	1	0	0	0	1	0
0	1	1	1	0	0	1	0	1	0	1	1
1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	1	1	1
1	0	1	0	0	1	0	0	0	0	1	0
1	0	1	1	1	1	0	0	0	0	1	0
1	1	0	0	0	0	0	1	0	0	0	1
1	1	0	1	1	0	0	1	1	1	1	1
1	1	1	0	0	1	1	1	0	0	1	1
1	1	1	1	1	1	1	1	1	0	1	1

Step 4: Maps for Y_1 and Y_2

		$Y_1 Y_2$	00	01	11	10
		$x_1 x_2$	00	01	11	10
$x_1 x_2$	00	0	0	0	0	
	01	0	0	1	1	
	11	0	1	1	1	
	10	0	1	1	1	

Map for Y_1

		$Y_1 Y_2$	00	01	11	10
		$x_1 x_2$	00	01	11	10
$x_1 x_2$	00	0	1	0	0	
	01	0	1	1	0	
	11	1	1	1	1	
	10	0	1	0	0	

Map for Y_2

Fig. 4.9: K map simplification

Step 5: Transition Table

TABLE 4.13 : Transition Table

		$Y_1 Y_2$	00	01	11	10
		$x_1 x_2$	00	01	11	10
$x_1 x_2$	00	(00)	(01)	00	00	
	01	(00)	(01)	(11)	(10)	
	11	01	11	(11)	11	
	10	(00)	11	10	(10)	

4.8 RACES

Races exist in asynchronous sequential circuits when two or more binary state variables change during a state transition.

Races are classified as:

- (i) Non-critical races
- (ii) Critical races.

4.8.1 Non-Critical Races

If the final stable state that the circuit reaches does not depend on the order in which the state variables change, the race is called a non-critical race.

If a circuit, whose transition table is shown in **Figure 4.10**, starts in stable state 000 ($y_1 y_2 x$) and then change the input from 0 to 1. The state variable must change from 00 to 11, which defines a race condition. The possible transitions are:

$00 \rightarrow 11$

$00 \rightarrow 01 \rightarrow 11$

$00 \rightarrow 10 \rightarrow 11$

In all cases the final stable state ($y_1 y_2 x = 111$) is the same, which results in a non-critical race condition.

		X	
		0	1
Y ₁ Y ₂		00	11
00		(00)	
01			11
11			(11)
10			11

$00 \rightarrow 11$
 $00 \rightarrow 01 \rightarrow 11$
 $00 \rightarrow 10 \rightarrow 11$

Fig. 4.10 : Non-critical race

4.8.2 Critical Race

A race becomes critical if the correct next state is not reached during a state transition. If it is possible to end up in two or more different stable states, depending on the order in which the state variables change, then it is a critical race. For proper operation, critical races must be avoided.

The transition table for **Figure 4.11** illustrate critical race condition. It starts in stable state $y_1 y_2 x = 000$ and then change the input from 0 to 1. The state variables must change from 00 to 11. If they change simultaneously, the final total stable state is 111. If y_2 changes to 1 before y_1 because of unequal propagation delay, then the circuit goes to the total stable state 011 and remains there. On the other hand, if y_1 changes first, the internal state becomes 10 and the circuit will remain in the stable total state 101. Hence the race is critical because the circuit goes to different stable states depending on the order in which the state variables change.

		x	
		0	1
y ₁ y ₂		00	11
00		(00)	
01			(01)
11			(11)
10			(10)

Possible Transitions	Final State
$00 \rightarrow 11$	111
$00 \rightarrow 01$	011
$00 \rightarrow 10$	101

Fig. 4.11 : Critical race

4.9 CYCLES

Races can be avoided by directing the circuit through intermediate unstable states with a unique state-variable change. When a circuit goes through a unique sequence of unstable states, it is said to have a cycle.

Figure 4.12 illustrates the occurrence of cycles. The state variable starts with $y_1y_2 = 00$ and then change the input from 0 to 1. The transition table **Figure 4.12(a)** gives a unique sequence that terminates in a total stable state 101. The transition table **Figure 4.12(b)** shows that even though the state variables change from 00 to 11, the cycle provides a unique transition from 00 to 01 and then to 11. The cycle should terminate with a stable state. If the cycle does not terminate with a stable state, the circuit will keep going from one unstable state to another, making the entire circuit unstable. This is illustrated in **Figure 4.12(c)**.

	x	0	1
y_1y_2			
00	(00)	01	
01		11	
11		10	
10		(10)	

(a) State transition
 $00 \rightarrow 01 \rightarrow 11 \rightarrow 10$

	x	0	1
y_1y_2			
00	(00)	01	
01		11	
11		(11)	
10		(10)	

(b) State transition
 $00 \rightarrow 01 \rightarrow 11$

	x	0	1
y_1y_2			
00	(00)	01	
01		11	
11		10	
10		01	

(c) Unstable state
 $00 \rightarrow 11 \rightarrow 10$

Fig. 4.12 : Cycles

4.10 RACE FREE STATE ASSIGNMENT

Critical Races may be avoided by making a proper binary assignment to the state variables. The state variables must be assigned binary numbers in such a way that only one stable variable can change at any one time when a state transition occurs in the flow table.

Three techniques are commonly used for making a critical-race free state assignment:

1. Shared row State assignment
2. Multiple row State assignment
3. One Hot State assignment

4.10.1 Shared Row State Assignment

In shared row state assignment, an extra row is introduced in a flow table that is shared between two stable states.

Figure 4.13 shows a transition diagram. It shows that there is a transition from state ‘a’ to state ‘b’ and transition from state ‘a’ to state ‘c’. The state ‘a’ is assigned binary value 00 and $b = 01$, $c = 11$.

This assignment will cause a critical race during the transition from ‘a’ to ‘c’, because there are two changes in the binary state variable.

$00 \rightarrow 01 \rightarrow 11$

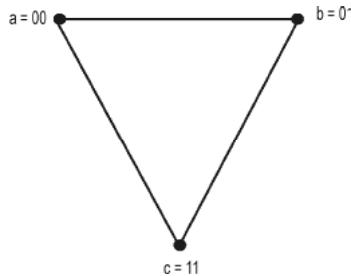


Fig. 4.13 : Transition Diagram

TABLE 4.14: Random State Assignment

State	State Variable	
	F_2	F_1
a	0	0
b	0	1
c	1	1

A race free assignment can be obtained by introducing extra row in a flow Table. This extra state is shared between two stable states. The flow table is shown in **Table 4.15**.

TABLE 4.15: Flow Table

State	F_2	F_1
a	0	0
b	0	1
c	1	1
d	1	0

The transition diagram with addition binary state ($d = 10$) is shown in **Figure 4.14**. State ‘d’ is adjacent to both ‘a’ and ‘c’. Now the transition from ‘a’ to ‘c’ will go through ‘d’. This causes the binary variables to change from $00 \rightarrow 10 \rightarrow 11$, which satisfy the condition that only one binary variable changes during each state transition, thus avoiding the critical race.

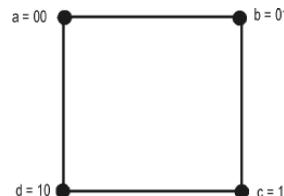


Fig. 4.14 : Modified Transition diagram

4.10.2 Multiple Row State Assignment

The multiple row state assignment technique specifies that each row in the original flow table be replaced with two rows. Each new row or total state is equivalent to the original state before splitting; for instance, state $a = a_1 = a_2$. Row ‘ a ’ existed in a flow table is replaced by rows a_1 and a_2 . The binary assignment for state $a_1(000)$ is the complement of binary assignment for state $a_1(111)$.

Table 4.16 shows the original flow table and **Table 4.17** shows the multiple row state assignment.

State ‘ a ’ is replaced by two equivalent states $a_1 = 000$ and $a_2 = 111$

State ‘ b ’ is replaced by two equivalent states $b_1 = 001$ and $b_2 = 110$

State ‘ c ’ is replaced by two equivalent states $c_1 = 011$ and $c_2 = 100$

State ‘ d ’ is replaced by two equivalent states $d_1 = 010$ and $d_2 = 101$

The output values, not shown here, must be same in a_1 and a_2 , b_1 and b_2 , c_1 and c_2 , d_1 and d_2 .

Table 4.16 Flow Table

	00	01	11	10
a	b	(a)	d	(a)
b	(b)	d	(b)	a
c	(c)	a	b	(c)
d	c	(d)	(d)	c

Table 4.17 : Multiple row state assignment

	00	01	11	10
$a_1 = 000$	b_1	(a_1)	d_1	(a_1)
$a_2 = 111$	b_2	(a_2)	d_2	(a_2)
$b_1 = 001$	(b_1)	d_2	(b_1)	a_1
$b_2 = 110$	(b_2)	d_1	(b_2)	a_2
$c_1 = 011$	(c_1)	a_2	b_1	(c_1)
$c_2 = 100$	(c_2)	a_1	b_2	(c_2)
$d_1 = 010$	c_1	(d_1)	(d_1)	c_1
$d_2 = 101$	c_2	(d_2)	(d_2)	c_2

In the multiple row assignment, the change from one stable state to another will always cause a change of only one binary state variable. Each stable state has two binary assignments with exactly the same output. At any given time, only one of the assignments is in use. For example, if we start with state a_1 and input 01 and then change the input to 11, 01, 00 and back to 01, the sequence of internal states will be a_1, d_1, c_1 and a_2 . Although the circuit starts in state a_1 and terminates in states a_2 , as far as the input-output relationship is concerned, the two states a_1 and a_2 are equivalent to state “ a ” of the original flow table.

4.10.3 One Hot State Assignment

One hot state assignments are made so that only one variable is active or “HOT” for each row in the original flow table. It requires as many state variables as there are rows in a flow table. Additional rows are introduced to provide single variable changes between internal state transitions.

Consider the flow table given in **Table 4.18**. 4 state variables (a, b, c, d) are used to represent the four rows in the table. Each row is represented by a case where only one of the four state variables is a_1 . A transition from state ‘ a ’ to state ‘ b ’ requires two state variable changes. By introducing a new row, one whose state assignment contains is where both states ‘ a ’ and ‘ b ’ have 1s, permits the transition between ‘ a ’ and ‘ b ’ without a race. The complete one hot state assignment flow table is shown in **Table 4.19**.

The transition from state ‘ b ’ to state ‘ a ’ is accomplished by passing through dummy state Q . If the machine is in state ‘ b ’ and a 00 input is received then the machine will change state to Q and then to ‘ a ’, without any further change in the inputs. State ‘ a ’ is replaced by Q in the ‘ b ’ row.

Row c with a state assignment of 0100 makes a change to state ‘ a ’ when the input is 00. State ‘ a ’ has an assignment of 0001. By passing through state R , on the way from c to a , races are eliminated. The state transition path is $C(0100) \rightarrow R(0101) \rightarrow a(0001)$.

TABLE 4.18 : Flow table

State Variables				State	Inputs (XY)			
F_4	F_3	F_2	F_1		00	01	11	10
0	0	0	1	a	(a)	b	c	c
0	0	1	0	b	a	(b)	c	d
0	1	0	0	c	a	b	(c)	(c)
1	0	0	0	d	(d)	b	c	(d)

TABLE 4.19 : One hot state assignment flow table

State Variables				State	Inputs (XY)			
F_4	F_3	F_2	F_1		00	01	11	10
0	0	0	1	a	(a)	Q	R	R
0	0	1	0	b	Q	(b)	S	T
0	1	0	0	c	R	S	(c)	(c)
1	0	0	0	d	(d)	T	U	(d)
0	0	1	1	Q	a	b	—	—
0	1	0	1	R	a	—	c	c
0	1	1	0	S	—	b	c	—
1	0	1	0	T	—	b	—	d
1	1	0	0	U	—	—	c	—

4.11 MINIMIZATION OF PRIMITIVE FLOW TABLE

The minimization of primitive flow table means that reduction of the primitive flow table to a minimum number of rows. This reduction will reduce the number of state variables required to realize the table. This will make it easier to complete the design of the network and will reduce the amount of logic required.

The minimization of primitive flow table is done by eliminating redundant stable states. To do this, we must find equivalent stable states. Two stable states are equivalent iff they have the same inputs and the associated internal states are equivalent. Thus two stable states are equivalent if (i) their inputs are the same.(ii) their outputs are the same.(iii) their next states are equivalent for each possible next input.

Consider the primitive flow table given in **Table 4.20**. From the first column (input 00) states 2, 6 and 8 have the same outputs (01). From the second column (input 01) 5 and 12 have the same outputs (11). From the third column (input 11) 3 and 10 have the same outputs (10) and from the last column (input 10) 4 and 11 have the same outputs (00). Therefore the sets of potentially equivalent states are:

$$(2, 6, 8) \quad (5, 12) \quad (3, 10) \quad (4, 11)$$

- (i) Next we examine the next states of the states in each group. With input 01, the next states of 2 and 6 are 5 and 7. Since $5 \neq 7, 2 \neq 6$. Similarly the next states of 6 and are 7 and 12. Since $7 \neq 12, 6 \neq 8$. However for 2 and 8, with input 01 the next states are 5 and 12 and with input 10 both next states are 4.
- (ii) For 5 and 12, both next states are 6 for input 00 and both next states are 9 for input 11.
- (iii) For 3 and 10, both next states are 7 for input 01.
- (iv) For 4 and 11, the next states are 2 and 8 for input 00 and 3 and 10 for input 11.

Since we have found no additional non-equivalent states, the remaining sets of equivalent states are: (2, 8) (5, 12) (3, 10) (4, 11)

Therefore we can eliminate one equivalent state from the sets. After eliminating redundant states 8, 10, 11 and 12, the reduced primitive flow table is given by **Table 4.21**.

TABLE: 4.20: Primitive flow table to be reduced

States	Inputs (XY)				Output
	00	01	11	10	
1	(1)	7	—	4	11
2	(2)	5	—	4	01
3	—	7	(3)	11	10
4	2	—	3	(4)	00
5	6	(5)	9	—	11
6	(6)	(7)	—	11	01
7	1	7	14	—	10
8	(8)	12	—	4	01
9	—	7	(9)	13	01
10	—	7	(10)	4	10
11	8	—	10	(11)	00
12	6	(12)	9	—	11
13	8	—	14	13	11
14	—	12	14	11	00

TABLE: 4.21: Primitive flow table

States	Inputs (XY)				Output
	00	01	11	10	
1	(1)	7	—	4	11
2	(2)	5	—	4	01
3	—	7	(3)	4	10
4	2	—	3	(4)	00
5	6	(5)	9	—	11
6	(6)	7	—	4	01
7	1	(7)	14	—	10
9	—	7	(9)	13	01
13	2	—	14	(13)	11
14	—	5	(14)	4	00

Example 4.4: Minimize the primitive flow table of **Table 4.22**.

TABLE: 4.22 : Primitive flow table

States	Inputs (XY)				Output
	00	01	11	10	
1	(1)	2	3	4	0
2	8	(2)	7	9	1
3	1	2	(3)	9	0
4	1	6	7	(4)	1
5	(5)	6	7	4	0
6	5	(6)	3	4	1
7	8	2	(7)	9	0
8	(8)	2	3	4	0
9	5	10	3	(9)	1
10	1	(10)	7	9	0
11	5	10	(11)	4	0

Solution: The sets of stable states which have same inputs and output are shown in **Table 4.23**. The states in each set are potentially equivalent.

(1, 5, 8) (2, 6) (3, 7, 11) (4, 9)

TABLE: 4.23: Potentially equivalent states

States	Inputs (XY)				Output
	00	01	11	10	
1	1	2	3	4	
5	5	6	7	4	0
8	8	2	3	4	
2	8	2	7	9	
6	5	6	3	4	1
3	1	2	3	9	
7	8	2	7	9	0
11	5	10	11	4	
4	1	6	7	4	1
9	5	10	3	9	

Now examine the same next states of the states in each group. $1 \equiv 8$, since both states have the same next state for input 01.

$3 \equiv 7$, since both states have same next state for input 10.

Therefore eliminating the redundant states 7 and 8 from the original primitive flow table. The reduced primitive flow table is shown in **Table 4.24**.

TABLE 4.24: Reduced Primitive Flow Table

State	Inputs (XY)				Output
	00	01	11	10	
1	1	2	3	4	0
2	1	2	3	9	1
3	1	2	3	9	0
4	1	6	3	4	1
5	5	6	3	4	0
6	5	6	3	4	1
9	5	10	3	9	1
10	1	10	3	9	0
11	5	10	11	4	0

4.12 DESIGN OF FUNDAMENTAL MODE ASYNCHRONOUS SEQUENTIAL CIRCUITS

Example 4.5: Design a fundamental mode asynchronous sequential circuit with the following behaviour. The circuit has two external inputs A and B and one output Z . The state diagram is shown in **Figure 4.15**.

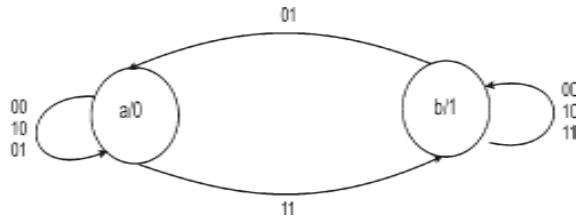


Fig. 4.15: State Diagram

Solution: For this design one does not need to obtain a primitive flow table, merger diagram, reduced primitive flow table, transition diagram since, **Figure 4.15** cannot be reduced to less than two states. For two states, one state variable is required which will be called y . Assigning the state code 0 to state ' a ' and state code 1 to state ' b ' and output $Z = y$. The K-map for the next state and external outputs can be drawn using the state diagram. The logic diagram is shown in **Figure 4.16**.

TABLE 4.25

Present State	AB inputs				Output Z	
	Next State					
	00	01	11	10		
a	a	a	b	a	0	
b	b	a	b	b	1	

$$a \Rightarrow 0$$

$$b \Rightarrow 1$$

TABLE: 4.26: State Table

Present State	Next State				Output Z	
	00	01	11	10		
0	0	0	1	0	0	
1	1	0	1	1	1	

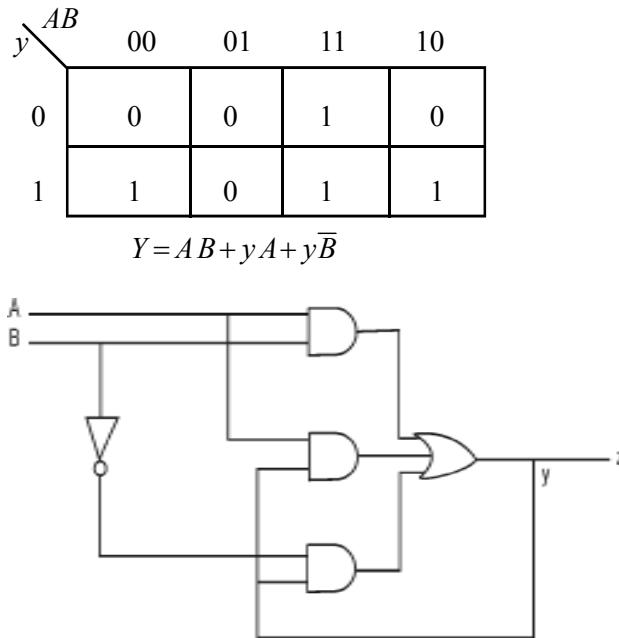


Fig. 4.16: Logic Diagram

Example 4.6: Obtain the primitive flow table for a fundamental mode asynchronous sequential circuit that has two inputs x_1 and x_2 and a single output z .

- The inputs x_1 and x_2 never change simultaneously.
- The output is always 0 when $x_1 = 0$, independent of the value of x_2 .
- The output is to become 1 if x_2 changes while $x_1 = 1$ and is to remain 1 until x_1 becomes 0 again.

Solution:

Table 4.27: Primitive flow table

Present State	Next State				Output (z)			
	Input state (x_1x_2)				Input state (x_1x_2)			
	00	01	10	11	00	01	10	11
A	(A)	B	C	-	0	-	-	-
B		(B)			-	0	-	-
C			(C)		-	-	0	-

Stable State (A)

When the input state $x_1x_2 = 00$, the output must be 0 for stable state (A) since $x=0$ when $x_1 = 0$.

When the input state is changed from $x_1x_2 = 00$ to $x_1x_2 = 01$, the circuit must become unstable, since only one stable state is permitted in each row of a primitive flow table and eventually reach another stable state, say \textcircled{B} . In the next state section of the primitive flow table an uncircled B is placed in the first row, second column and a circled \textcircled{B} is placed in the second row, second column. The output $z = 0$ for stable state \textcircled{B} , since $x_1 = 0$.

When the input state is changed from $x_1x_2 = 00$ to $x_1x_2 = 10$, another row of \textcircled{C} is needed. An uncircled \textcircled{C} is placed in the first row, third column and the circled \textcircled{C} is placed in the third row, third column of the next-state section. The output $z = 0$, since $z = 1$ if x_2 changes while $x_1 = 1$.

When the input state is changed from $x_1x_2 = 00$ to $x_1x_2 = 11$, a dash (–) is placed in the first row, fourth column of the next section, since the inputs x_1, x_2 never change simultaneously. Now the primitive flow table appears as shown in **Table 4.27**.

Table 4.28: Primitive Flow Table

Present state	Next state				Output (z)			
	Input state (x_1x_2)				Input state (x_1x_2)			
	00	01	10	11	00	01	10	11
A	\textcircled{A}	B	C	–	0	–	–	–
B	A	\textcircled{B}	–	D	–	0	–	–
C			\textcircled{C}	–	–	–	0	–
D				\textcircled{D}	–	–	–	0

Stable State \textcircled{B}

When the input state is $x_1x_2 = 01$, the output must be 0 for stable state \textcircled{B} since $x_1 = 0$.

When the input state is changed from $x_1x_2 = 01$ to $x_1x_2 = 00$, the circuit must first become unstable and then reach a stable state. Since stable state \textcircled{A} is already exists in the first column of the next-state section, an uncircled \textcircled{A} is placed in the second row, first column.

When the input state is changed from $x_1x_2 = 01$ to $x_1x_2 = 11$, a stable state must appear in the fourth column of the next-state section of the primitive flow table. Thus, an internal change to a stable state \textcircled{D} is needed. The output $z = 0$ for stable state \textcircled{D} , since x_2 did not change but remained at the value $x_2 = 1$.

When the input state is changed from $x_1x_2 = 10$, a dash (–) is placed in the second row, third column of the next-state section, since both input variable x_1, x_2 cannot change simultaneously. Now the primitive flow table appears as shown in **Table 4.28**.

Table 4.29: Primitive flow table

Present State	Next state				Output (z)			
	Input state (x_1x_2)				Input state (x_1x_2)			
	00	01	10	11	00	01	10	11
A	(A)	B	C	-	0	-	-	-
B	A	(B)	-	D	-	0	-	-
C	A	-	(C)	E	-	-	0	-
D				(D)	-	-	-	0
E				(E)	-	-	-	1

Stable State (C)

When the input state is changed from $x_1x_2 = 10$ to $x_1x_2 = 11$, then a stable state with a 1 output must be reached in the fourth column of the next-state section of the primitive flow table. This cannot be state D, since the output is to become 1 when x_2 changes while $x_1 = 1$. Thus a stable state with a 1 output in the fourth column is needed. This necessitates a fifth row for present state E to be added in the primitive flow table. The uncircled E is placed in the third row, fourth column and circled E is placed in the fifth row, fourth column. The output $z = 1$ for stable state E.



When the input is changed from $x_1x_2 = 10$ to $x_1x_2 = 00$ the circuit must first become unstable and then each a stable state. Since stable state (A) already exists in the first column of the next-state section, an uncircled A is placed in the third row, first column.

When the input is changed from $x_1x_2 = 10$ to $x_1x_2 = 01$, a dash (-) is placed in the third row, second column, since both inputs x_1x_2 never change simultaneously. Now the primitive flow table appears shown in **Table 4.29**.

Table 4.30: Primitive Flow Table

Present State	Next State Input state (x_1x_2)				Output (z) Input state (x_1x_2)			
	00	01	10	11	00	01	10	11
A	(A)	B	C	-	0	-	-	-
B	A	(B)	-	D	-	0	-	-
C	A	-	(C)	E	-	-	0	-
D	-	B	F	(D)	-	-	-	0
E				(E)	-	-	-	1
F			(F)		-	-	1	-

Stable State D

When the input state is changed from $x_1x_2 = 11$ to $x_1x_2 = 10$, then a stable state with a 1 output must be reached in the third column of the next-state section of the primitive flow table. Since the only stable state in the third column thus far has a 0 output, another row is added to the table for a stable state F along with its 1 output.

When the input state is changed from $x_1x_2 = 11$ to $x_1x_2 = 01$, an uncircled B is placed in the fourth row, second column.

When the input is changed from $x_1x_2 = 11$ to $x_1x_2 = 00$, a dash (-) is placed in the fourth row, first column, since both inputs x_1x_2 never change simultaneously. Now the primitive flow table appears as shown in **Table 4.30**.

The remaining entries in the last two rows are determined by similar reasons as mentioned above. The completed primitive flow table is shown in **Table 4.31**.

Table 4.31: Primitive Flow Table

Present State	Next State				Output (z)			
	Input state (x_1x_2)				Input state (x_1x_2)			
	00	01	10	11	00	01	10	11
A	(A)	B	C	-	0	-	-	-
B	A	(B)	-	D	-	0	-	-
C	A	-	(C)	E	-	-	0	-
D	-	B	F	(D)	-	-	-	0
E	-	B	F	(E)	-	-	-	1
F	A	-	(F)	E	-	-	1	-

Example 4.7

Obtain the primitive flow table for an asynchronous sequential circuit that has two inputs x, y and one output z . The inputs x and y never change or are 1 simultaneously. An output $z = 1$ is to occur only during the input state $xy = 01$ and then if and only if the input state $xy = 01$ is preceded by the input sequences $xy = 01, 00, 10, 00, 10, 00$.

(Nov. 2004)

Solution

<i>Input Sequence</i>	<i>Stable State</i>
01	(A)
00	(B)
10	(C)
00	(D)
10	(E)
00	(F)

Stable state \textcircled{A} is placed in the first row, second column of the next-state section of the primitive flow table for the input state $xy = 01$, assumed that the output $z = 0$.

When the input state is changed from $xy = 01$ to $xy = 00$, a second row must be added for stable state \textcircled{B} . An uncircled \textcircled{B} is placed in the first row, first column and a circled \textcircled{B} is placed in the second row, first column.

The input states $xy = 10$ and $xy = 11$ cannot follow the input state $xy = 01$, since the inputs xy never change or are 1 simultaneously.

Next an input state of $xy = 10$ results in the circuit entering stable state \textcircled{C} . This state signifies that the input sequence $xy = 01, 00, 10$ has been applied.

Next an input state of $xy = 00$ results in the circuit entering stable state \textcircled{D} . This state signifies that the input sequence is $xy = 01, 00, 10, 00$.

Similarly, stable state \textcircled{E} and \textcircled{F} are placed in the primitive flow table as shown in **Table 4.32**. Stable state \textcircled{G} is introduced with an associated 1 output.

Table 4.32: Primitive Flow Table

Present State	Next State				Output (z)			
	Input state (xy)				Input state (xy)			
	00	01	10	11	00	01	10	11
A	B	\textcircled{A}	—		—	0	—	
B	\textcircled{B}		C		0	—	—	—
C	D		\textcircled{C}		—	—	0	—
D	\textcircled{D}		E		0	—	—	—
E	F		\textcircled{E}		—	—	0	—
F	\textcircled{F}	G			0	—	—	—
G		\textcircled{G}			—	1	—	—

This asynchronous sequential circuit has the following requirements:

1. The inputs x and y never change simultaneously i.e., only one input variable is allowed to change value at a time.
2. The inputs x and y are never 1 simultaneously.

These requirements imply that only dashes (—) can appear in the fourth column and other some entries in the next-state section of the primitive flow table as shown in the **Table 4.33**.

The unstable states \textcircled{A} are entered in the second and fourth rows of the second column of the next section of the primitive flow table since the sequence to be recognized is broken by the input state $xy = 01$ and hence the recognition process must be restarted.

Table 4.33: Primitive Flow Table

Present State	Next State				Output (z)			
	Input state (xy)				Input state (xy)			
	00	01	10	11	00	01	10	11
A	B	(A)	—	—	—	0	—	—
B	(B)	—	C	—	0	—	—	—
C	D	—	(C)	—	—	—	0	—
D	(D)	—	E	—	0	—	—	—
E	F	—	(E)	—	—	—	0	—
F	(F)	G	—	—	—	1	—	—
G	(G)	—	—	—	—	1	—	—

The unstable states (B) is entered in the seventh row, first column, since the input state for the stable state in that row can serve both as the first $xy = 01$ input of a new sequence as well as the last input of the recognized sequence.

Finally two additional stable states (H) and (J) must be added to handle the situation that the input state $xy = 01$, which defines the beginning and end of the input sequence to be recognized. In the last row an unstable state (A) indicates the first input state of the sequence to be recognized is applied and an unstable state H provides for the network to wait for the beginning of the specified input sequence that is to be recognized. The complete primitive flow table is shown in Table 4.34.

Table 4.34: Primitive flow table

Present State	Next state				Output (z)			
	Input state (xy)				Input state (xy)			
	00	01	10	11	00	01	10	11
A	B	(A)	—	—	—	0	—	—
B	(B)	A	C	—	0	—	—	—
C	D	—	(C)	—	—	—	0	—
D	(D)	A	E	—	0	—	—	—
E	F	—	(E)	—	—	—	0	—
F	(F)	G	H	—	0	—	—	—
G	B	(G)	—	—	—	1	—	—
H	1	—	(H)	—	—	—	0	—
I	(I)	A	H	—	0	—	—	—

4.13 DESIGN OF PULSE MODE ASYNCHRONOUS SEQUENTIAL CIRCUITS

The pulse mode asynchronous sequential circuit input signal restrictions can be stated by the following rules:

Rule 1: Only one input signal pulse is allowed to occur at one time.

Rule 2: Before the next pulse is allowed to occur, the circuit must be given time to reach a new stable state via a single state change.

Rule 3: Each applied input pulse has a minimum pulse width that is determined by the time it takes to change the slowest flipflop in the circuit to a new stable state.

Rule 4: The maximum pulse width of an applied input pulse must be sufficiently narrow so that it is no longer present when the new present state output signals become available.

Example 4.8: Design a pulse mode asynchronous sequential circuit, that has two inputs X_1 and X_2 , one Mealy output Z_1 and one Moore output Z_2 . The Mealy output is coincident with the third consecutive pulse on input X_2 and the Moore output occurs after the third consecutive pulse on input X_2 or after any pulse that occurs on input X_1 .

Solution: The state diagram for the circuit based on the given problem is shown in **Figure 4.17**.

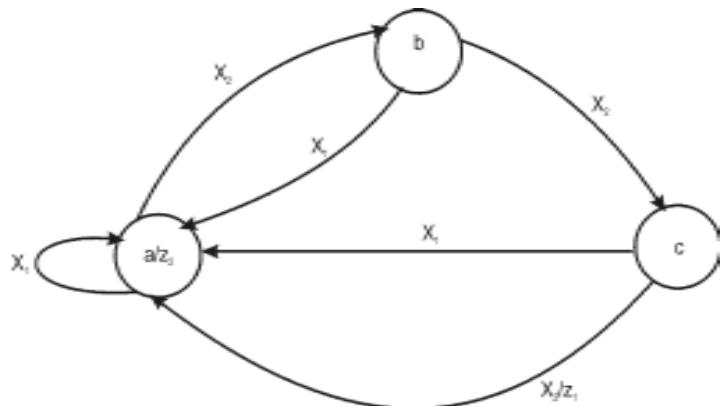


Fig. 4.17: State Diagram

The flow table can be obtained from the state diagram as shown in **Table 4.35**.

The input conditions $X_1 X_2 = 00, 01, 11$ and 10 that are normally shown in flow table for either a synchronous or fundamental mode asynchronous design, do not appear in a pulse mode design. In a pulse mode circuit, the presence of a pulse at an input causes the circuit to change its state; however, the absence of a pulse causes no change in the circuit and hence represents unimportant information. To present the presence of a pulse in a pulse mode circuit, only the name of the pulse is listed; X_1 represents input condition 10 and X_2 represents input condition 01 . Input condition 00 is not represented in the flow table because this is unimportant information, and input condition 11 is not represented in the flow table, because this violates the restriction imposed by Rule 1.

TABLE 4.35: Flow Table

Present State	Next State/Output Z_1		Output Z_2
	X_1	X_2	
a	$a, 0$	$b, 0$	1
b	$a, 0$	$c, 0$	0
c	$a, 0$	a, Z_1	0

The transition table with state assignment is shown in **Table 4.36**. For race-free state assignment, an extra state 'd' is added. The binary code assignment for the states as follows: $a \rightarrow 00, b \rightarrow 01, c \rightarrow 11, d \rightarrow 10$

TABLE 4.36: Transition Table

Present State		Next State/Output Z_1		Output Z_2
y_1	y_2	X_1	X_2	Z_2
0	0	00, 0	01, 0	1
0	1	00, 0	11, 0	0
1	1	00, 0	00, 1	0
1	0	00, 0	00, 0	0

Two state variables are required and y_1 and y_2 are chosen. For these two state variables, two memory storage devices are required. J-K flipflops can be used as T flipflops for the memory storage devices.

The composite excitation inputs T_1 and T_2 are obtained by using excitation table for T flipflop from the transition table of **Table 4.36**.

Remember the excitation table for T flipflops as shown in **Table 4.37**. By using the present state and reset state, flipflop inputs T_1 and T_4 are obtained as shown in **Table 4.38**.

TABLE 4.37: Excitation Table

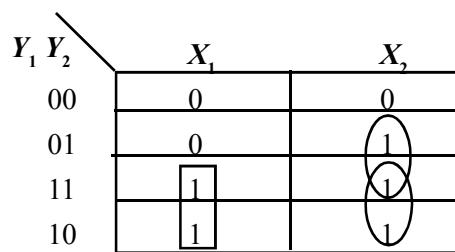
$Q(t)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

TABLE 4.38: Excitation Inputs

Present State		Next State		Flipflop Inputs	
Y_1	Y_2	X_1	X_2	T_1, T_2	T_1, T_2
0	0	00	01	00	01
0	1	00	11	01	10
1	1	00	00	11	11
1	0	00	00	10	10

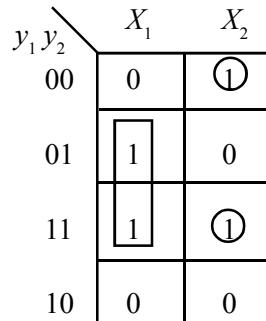
Using the K-map, logic hazard-free equations for T_1 , T_2 , Z_1 and Z_2 can be obtained as shown in Figure 4.18.

(i) Map for T_1



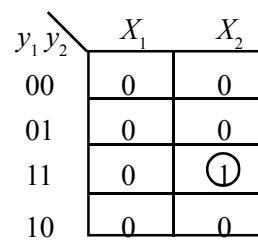
$$T_1 = y_1 x_1 + y_1 x_2 + y_2 x_2$$

(ii) Map for T_2



$$T_2 = y_2 x_1 + \bar{y}_1 \bar{y}_2 x_2 + \bar{y}_1 \bar{y}_2 x_2 + y_1 y_2 x_2 \quad (\text{hazard free equation})$$

(iii) Map for Z_1



$$Z_1 = y_1 y_2 x_2$$

Fig. 4.18: K map simplifications

The logic diagram for pulse mode asynchronous sequential circuit using positive edge triggered T flipflops is drawn by using of the following equations:

$$T_1 = y_1 x_1 + y_1 x_2 + y_2 x_2, \quad T_2 = y_2 x_1 + \bar{y}_1 \bar{y}_2 x_2 + y_1 y_2 x_2$$

$$Z_1 = y_1 y_2 x_2, \quad Z_2 = \bar{y}_1 \bar{y}_2$$

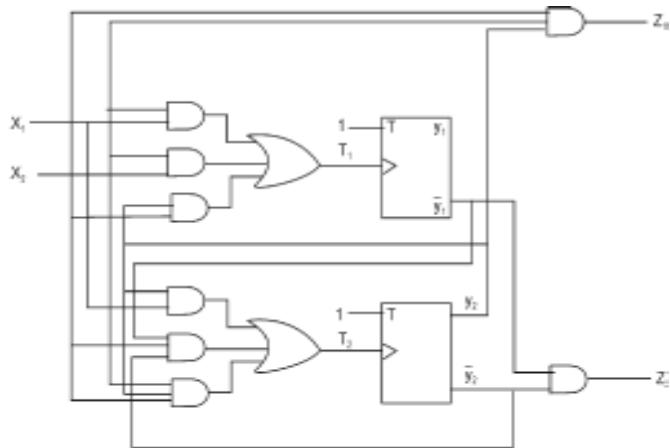


Fig. 4.19: Pulse Mode Sequential Circuit

Example 4.9: Design a pulse mode asynchronous sequential circuit for the automatic toll collecting machine in a road. Suppose the toll is Rupees 9 and the machine accepts Rupees 2 and rupees 5 coins and generate pulses X_2 and X_5 respectively. The sequential circuit should produce a level output whenever the amount received by the toll machine is rupees 9 or more. After a vehicle has passed, a reset pulse X_r is automatically produced, which makes output zero and resets the sequential circuit to its initial state.

Solution: The state table for the given problem is shown in **Table 4.39**.

TABLE 4.39: State Table

Toll in Rupees	Present State	Next State		Output	
		X_2	X_5	X_r	Z
0	A	B	D	A	0
2	B	C	F	A	0
4	C	E	H	A	0
5	D	F	H	A	0
6	E	G	H	A	0
7	F	H	H	A	0
8	G	H	H	A	0
9 or more	H	H	H	A	1

This state table shows there are eight states, hence 3 flipflops ($2^3 = 8$) are needed. **Table 4.42** shows the state assignments and derivation of S-R flipflop inputs with the help of SR flipflop excitation table, shown in **Table 4.40**.

TABLE 4.40: Excitation Table

$Q(t)$	$Q(t + 1)$	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

TABLE 4.41: State assignment and excitation tables

Present States	Next States			Flip-flop inputs								
				X_2			X_5			X_r		
PQR	X_2	X_5	X_r	S_1R_1	S_2R_2	S_3R_3	S_1R_1	S_2R_2	S_3R_3	S_1R_1	S_2R_2	S_3R_3
A \rightarrow 000	001	011	000	0 X	0 X	1 0	0 X	1 0	1 0	0 X	0 X	0 X
B \rightarrow 001	010	101	000	0 X	1 0	0 1	1 0	0 X	X 0	0 X	0 X	0 1
C \rightarrow 010	100	111	000	1 0	0 1	0 X	1 0	X 0	1 0	0 X	0 1	0 X
D \rightarrow 011	101	111	000	1 0	0 1	X 0	1 0	X 0	X 0	0 X	0 1	0 1
E \rightarrow 100	110	111	000	X 0	0 1	0 X	X 0	1 0	1 0	0 1	0 X	0 X
F \rightarrow 101	111	111	000	X 0	1 0	X 0	X 0	1 0	X 0	0 1	0 X	0 1
G \rightarrow 110	111	111	000	X 0	X 0	1 0	X 0	X 0	1 0	0 1	0 1	0 X
H \rightarrow 111	111	111	000	X 0	X 0	X 0	X 0	X 0	X 0	0 1	0 1	0 1

For S_1

X_2				X_5				X_r							
QR	00	01	11	10	QR	00	01	11	10	QR	00	01	11	10	
P	0	0	1	1	P	0	1	1	1	P	0	0	0	0	
0	0	0	X	X	1	X	X	X	X	1	0	0	0	0	
1	X	X	X	X	0	X	X	X	X	0	0	0	0	0	

$$S_1 = X_2 Q + X_5 (R + Q)$$

For R_i :

		X_2				
		QR	00	01	11	10
P		0	X	X	0	0
1		0	0	0	0	0

$$R_i = X_1$$

		X_5				
		QR	00	01	11	10
P		0	X	0	0	0
1		0	0	0	0	0

		X_r				
		QR	00	01	11	10
P		0	X	X	X	X
1		1	1	1	1	1

$$R_i = X_1$$

For S_2 :

		X_2				
		QR	00	01	11	10
P		0	0	(1)	0	0
1		0	(1)	X	X	X

$$S_2 = X_2 \bar{Q}R + X_5 (P + \bar{R})$$

		X_5				
		QR	00	01	11	10
P		0	1	0	X	X
1		1	(1)	1	X	X

		X_r				
		QR	00	01	11	10
P		0	0	0	0	0
1		0	0	0	0	0

$$S_2 = X_2 (\bar{Q}R + \bar{P}Q) + X_r$$

For R_i :

		X_2				
		QR	00	01	11	10
P		0	(X)	1	(1)	D
1		1	(1)	0	0	0

		X_5				
		QR	00	01	11	10
P		0	0	X	0	0
1		0	0	0	0	0

		X_r				
		QR	00	01	11	10
P		0	X	X	1	1
1		X	X	1	1	1

$$R_i = X_2 (\bar{Q}R + \bar{P}Q) + X_r$$

For S_3 :

		X_2				
		QR	00	01	11	10
P		0	(1)	0	X	0
1		0	X	(X)	1	0

		X_5				
		QR	00	01	11	10
P		0	1	X	X	1
1		1	X	X	1	1

		X_r				
		QR	00	01	11	10
P		0	0	0	0	0
1		0	0	0	0	0

$$S_3 = X_2 (\bar{P}\bar{Q}\bar{R} + P\bar{Q}) + X_5$$

For R_3 :

X_2				X_5				X_r									
P		Q	R	P		Q	R	P		Q	R						
0	1	00	01	11	10	0	1	00	01	11	10	0	1	00	01	11	10
0	X	0	(1)	0	X	0	0	0	0	0	0	X	1	1	X	X	
1	X	0	0	0	0	0	0	0	0	0	0	X	1	1	X	X	

$$R_3 = X_2 \bar{P} \bar{Q} R + X_r$$

Fig. 4.20: K map simplification

Figure 4.20 shows the K-map simplification for SR flipflop inputs. The input equations are:

$$S_1 = X_2 Q + X_5 (R + Q)$$

$$R_1 = X_r$$

$$S_2 = X_2 \bar{Q} R + X_5 (P + \bar{R})$$

$$R_2 = X_2 (\bar{Q} \bar{R} + \bar{P} Q) + X_r$$

$$S_3 = X_2 (\bar{P} \bar{Q} \bar{R} + P Q) + X_5$$

$$R_3 = X_2 \bar{P} \bar{Q} R + X_r$$

The output equation is, $Z = PQR$.

The logic diagram for pulse mode asynchronous sequential circuit using S-R flipflops is shown in Figure 4.21.

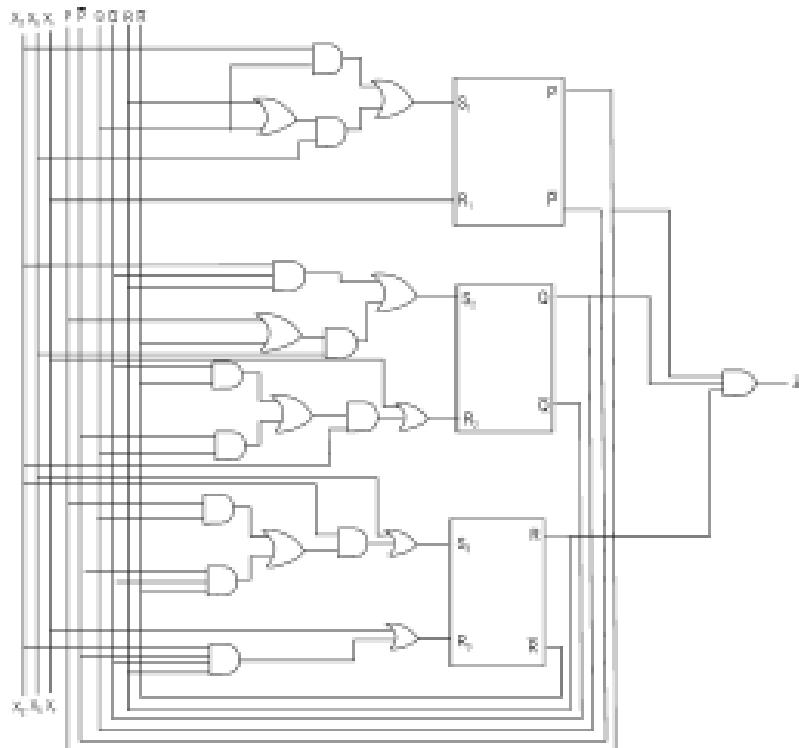


Fig. 4.21: Pulse Mode Sequential Circuit

4.14 HAZARDS

Hazard is an unwanted transient i.e., spike or glitch that occurs due to unequal path or unequal path or unequal propagation delays through a combinational circuit.

Types of hazards:

There are three types of hazards:

- ◆ Static hazard,
- ◆ Dynamic hazard,
- ◆ Essential hazard.

4.14.1 Static Hazards

Static hazard is a condition which results in a single momentary incorrect output due to change in a single input variable when the output is expected to remain in the same state. The two types of static hazard are:

- ◆ Static-0 hazard,
- ◆ Static-1 hazard.

Static-0 hazard: When the output is to remain at the value 0 and a momentary 1 output is possible during the transition between the two input states, then the hazard is called a static-0 hazard.

Static-1 hazard: When the output is to remain at the value 1 and a momentary 0 output is possible during the transition between the two input states, then the hazard is called a static-1 hazard.

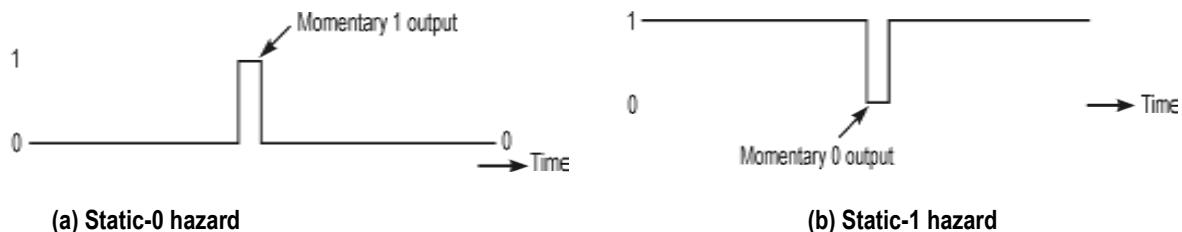


Fig. 4.22: Types of hazards

4.14.2 Dynamic Hazard

Dynamic hazards occur when the output of a network is to change between its two logic states, but a momentary false output signal occurs during the transient behaviour.

A dynamic hazard is defined as a transient change occurring 3 or more times at an output terminal of a logic network when the output is supposed to change only once during a transition between two input states differing in the value of one variable.

4.14.3 Essential Hazard

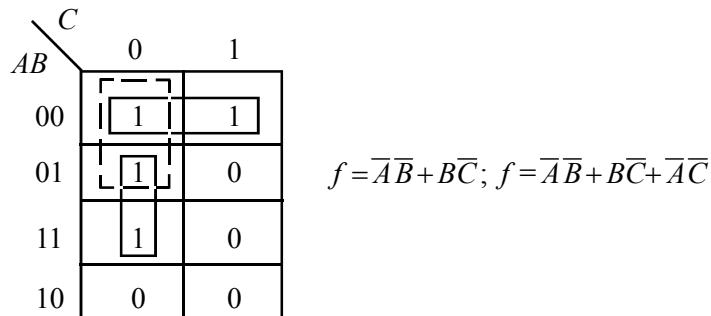
Essential hazard is a type of hazard that exists only in asynchronous sequential circuits with two or more feedbacks. An essential hazard is caused by unequal delays along two or more paths that originate from the same input. An excessive delay through an inverter circuit in comparison to the delay associated with the feedback path may cause essential hazard.

4.15 HAZARDS ELIMINATION

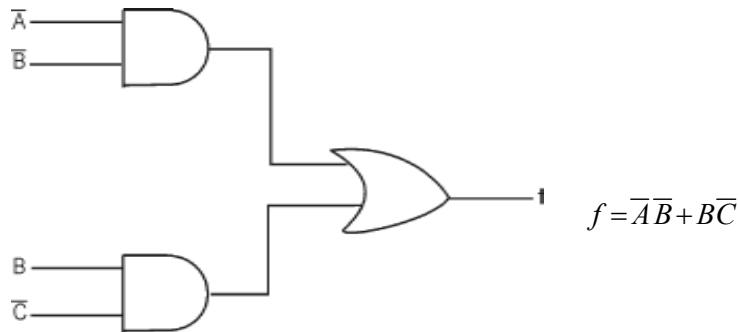
To design a hazard-free switching circuit, all adjacent input combinations having same output occur within some sub cube of the corresponding function. In other words, every pair of adjacent 1 cells or 0 cells in the K-map of a switching function should be covered by at least one sub cube.

4.15.1 Static Hazard

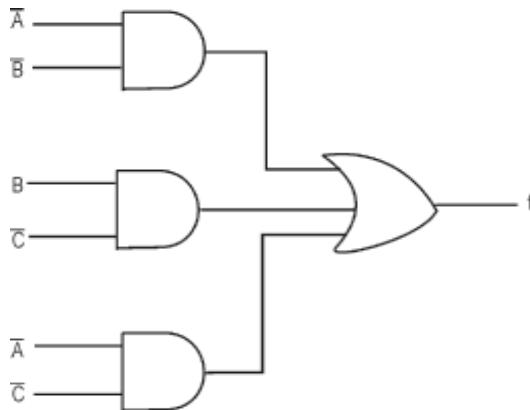
A static hazard can be removed by covering the adjacent cells with a redundant grouping that overlaps both groupings. In the K-map shown in **Figure 4.23(a)**, static hazard can be eliminated by covering adjacent cells corresponding to 000 and 010 as shown by dotted subcube. This leads to redundant grouping that overlaps both \overline{AB} and \overline{BC} groupings. The redundant term is \overline{AC} and the modified circuit is shown in **Figure 4.23(c)**. Now, when the input (ABC) changes from 000 to 010, the output will remain at '1' state (for both 010 and 000 inputs) because of high output at the newly added lower AND gate.



(a) K-map for the function $f = \sum (0, 1, 2, 6)$



(b) Logic circuit with static hazard



(c) Logic circuit without static hazard

Fig. 4.23

4.15.2 Dynamic Hazards Elimination

Dynamic hazard can also be eliminated in a similar manner as that of static hazard elimination by covering every pair of 1 cells and every pair of '0' cells in the K-map by at least one sub cube.

4.15.3 Essential Hazards Elimination

Essential hazard can be eliminated by adding redundant gates as in static hazards. They can be eliminated by adjusting the amount of delay in the affected path. For this, each feedback loop must be designed with extra care to ensure that the delay in the feedback path is long enough compared to the delay of other signals that originate from the input terminals.

Example 4.10: Design a logic hazard-free circuit to implement the following function:

$$F(A, B, C, D) = \sum m (0, 2, 4, 5, 6, 7, 8, 10, 11, 15)$$

Solution: Fig. 4.24(a) shows the K-map with minimum covering of 1's of the function. The adjacent 1s at minterm locations (1, 4), (2, 6), (7, 15) and (10, 11) are not covered in the same sub-cube. Adding product terms to cover these adjacent 1s results in a logic hazard-free function as shown in Fig. 4.24(b).

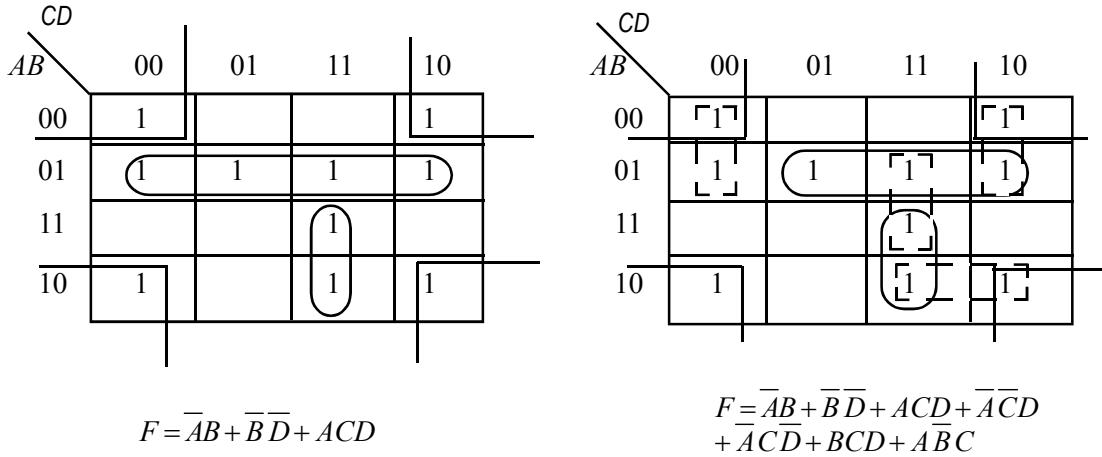


Fig. 4.24: K map simplifications

For hazard-free circuit,

$$F = \overline{AB} + \overline{B}\overline{D} + ACD + \overline{A}\overline{C}\overline{D} + \overline{A}CD + BCD + A\overline{B}C$$

Example 4.11: Implement the switching function $f(x_1, x_2, x_3) = x_1x_2 + \overline{x}_2x_3$ by a static hazard free 2 level AND-OR gate network. (Nov. 2004)

Solution: The K-map and AND-OR circuit are shown in Fig. 4.25 for the switching function $F = x_1x_2 + \overline{x}_2x_3$ with hazards.

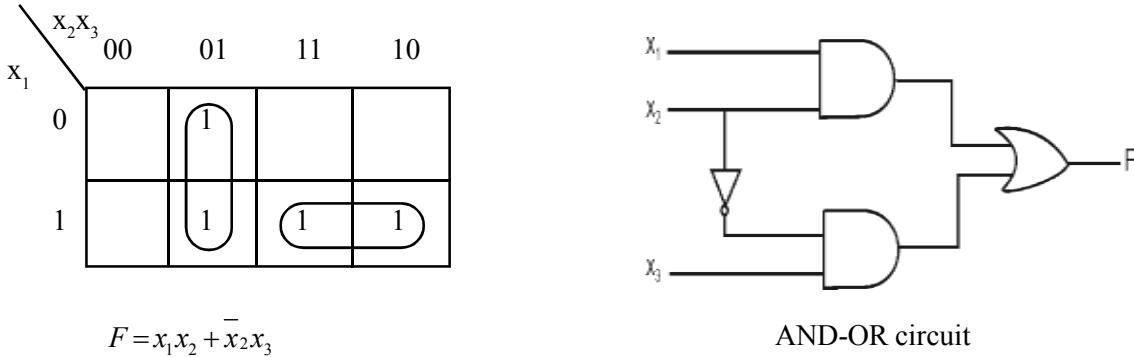


Fig. 4.25: Circuit with hazard

Hazards in combinational circuits can be removed by covering anytwo minterms that may produce a hazard with product term common to both. The removal of hazards requires the addition of redundant gates to the circuit. The hazard free circuit obtained by adding extra gate in the circuit generates the product terms x_1x_3 as shown in **Fig. 4.26**.

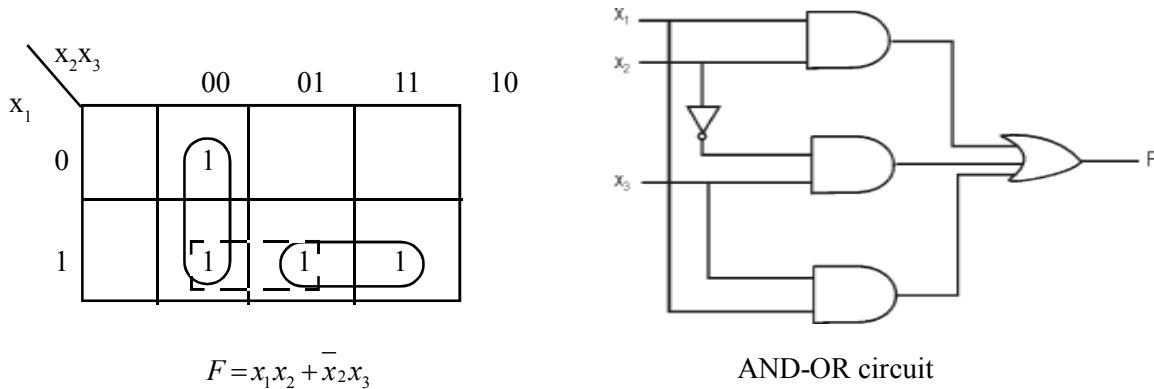


Fig. 4.26: Circuit with hazard

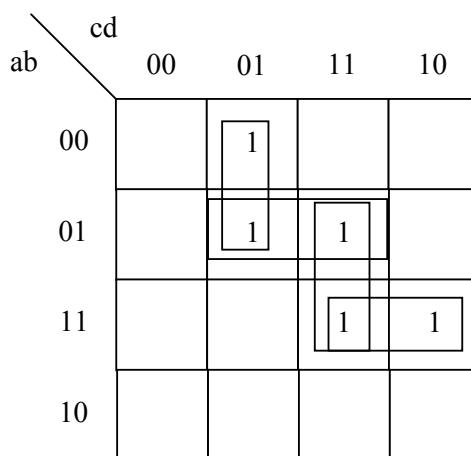
Example 4.12: Find a static and dynamic hazard free realization for the following function using

- (i) NAND gates
- (ii) NOR gates.

$$f(a, b, c, d) = \sum m(1, 5, 7, 14, 15)$$

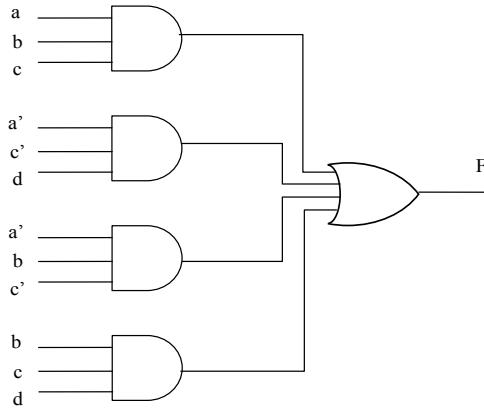
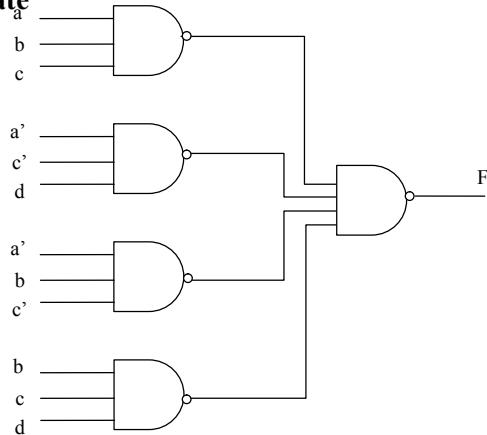
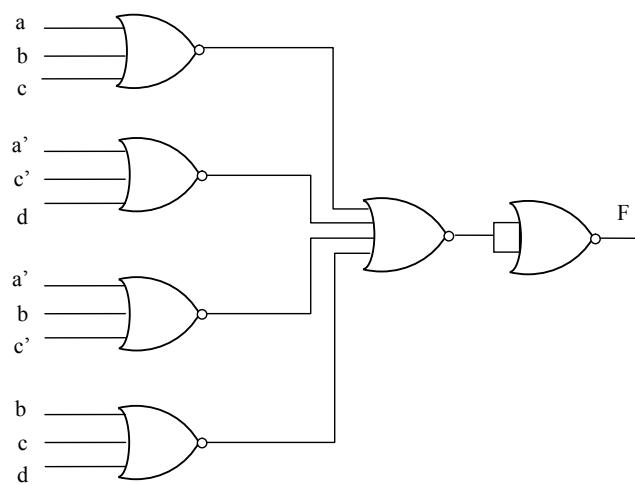
(Dec 2010)

Solution:



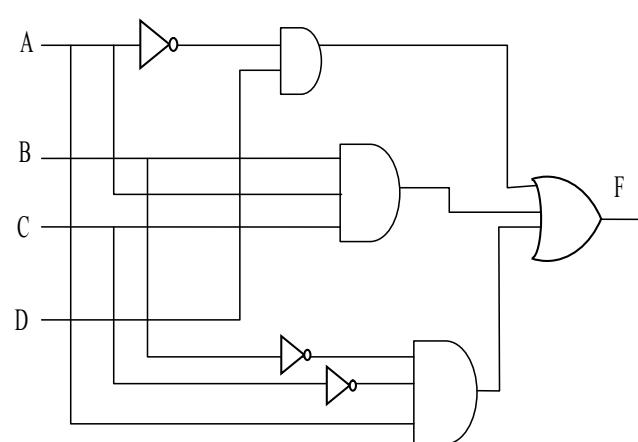
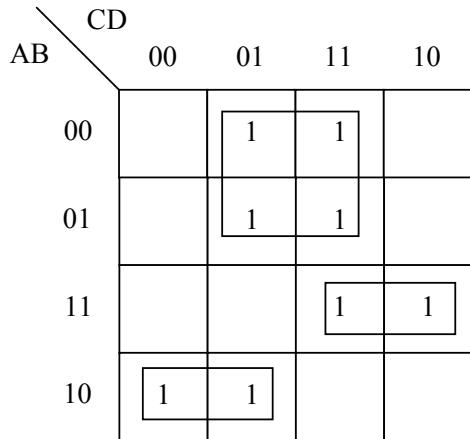
Hazard free expression

$$F = a'c'd + a'bd + bcd + abc$$

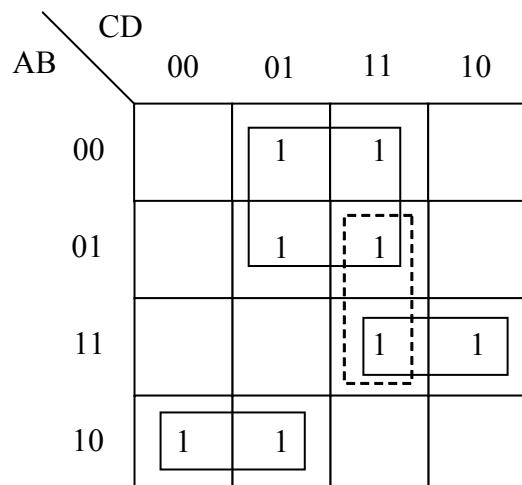
**Using NAND gate****Using NOR Gate**

Example 4.13: Implement the switching function $F = m(1, 3, 5, 7, 8, 9, 14, 15)$ by a static hazard free 2 level AND-OR gate network. (May 2012)

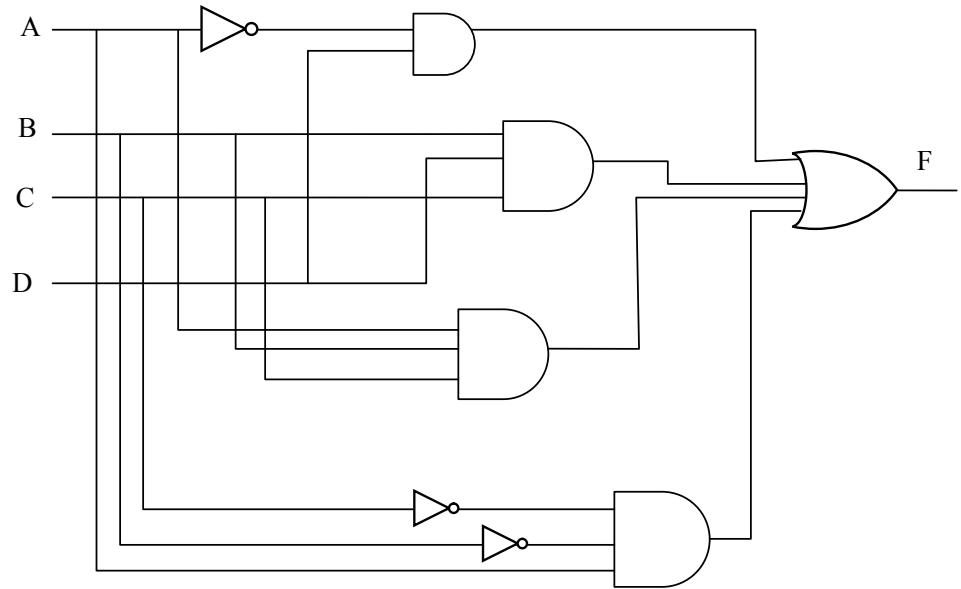
Solution:



$$F = \overline{AD} + ABC + A\overline{BC}$$



$$F = \overline{AD} + ABC + A\overline{BC} + BCD$$

**Fig: 4.27 Hazard free circuit**

ANNA UNIVERSITY QUESTIONS

PART-A

1. What is race around condition? (April 2003)
2. What is the cause for essential hazard? (April 2003)
3. What is static 0 hazard? (April 2003)
4. Why is the pulse mode operation of asynchronous sequential circuits not very popular? (Apr. 2003)
5. What is a dynamic hazard? (April 2003)
6. What is a fundamental mode asynchronous sequential circuit? (Nov. 2003)
7. What is a hazard? (Nov. 2003), (April, 2004)
8. With reference to a JK flipflop, what is racing? (April 2004)
9. Define glitch. (April 2004)
10. Define static hazard. (April 2004)
11. What is dynamic hazard ? (April 2004)
12. What is meant by race condition in a digital circuit. (April 2004)
13. What is an asynchronous sequential circuit? (Nov. 2004)
14. When do hazards occur? (Nov. 2004)
15. What is static 1 hazard ? (Nov. 2004)
16. Why a serial counter is referred to as asynchronous? (April 2005)
17. What is meant by critical race? (April 2005)
18. How to avoid dynamic hazard? Does N occur in sequential logic circuits? (April 2005)
19. Define cycles. (April 2005), (Dec. 2005)
20. Mention any one advantage and disadvantage of asynchronous sequential circuits. (Dec. 2005)
21. What is a critical race? Why should it be avoided ? (Dec. 2005)
22. How does the operation of an asynchronous input differ from that of a synchronous input?
23. Define a primitive flow table. (Dec. 2005)
24. Define the following terms: race, critical race, non-critical race. (Dec. 2005)
25. What is race problem in flipflops? (May 2006)
26. What are the classes of asynchronous sequential circuits ? (May 2006)
27. What are the two types of asynchronous circuits? How do they differ? (May 2006)
28. What is meant by a non-critical race? What is its cause? (May 2006)
29. What are the difference between PLA and PAL? (Dec. 2006)
30. What is a hazard in asynchronous sequential circuit? (May 2007)
31. What are the different modes of operation in asynchronous sequential circuits? (May 2007)
32. Define static 0 and static 1 hazards. (May 2007)

33. Distinguish between pulse mode and fundamental mode asynchronous circuits. (May 2007)
34. What are the assumptions made for pulse mode circuit? (Dec. 2006) (May 2007) (Dec. 2007)
35. What is a hazard in combinational circuits? (May 2007)
36. What is meant by critical race? (Dec. 2010)
37. What are the types of hazards? (Dec. 2010)
38. Define static and dynamic hazards. (May 2011)
39. What is meant by Race condition in Asynchronous sequential circuit. (Dec. 2011)
40. Define Hazard. List the hazards in combinational circuit. (Dec. 2011)
41. Define Race Condition. (May 2012)
42. What is meant by essential hazards? (May 2012)
43. Distinguish between a conventional flow chart and an ASM chart. (Dec. 2012)
44. Draw the block diagram of an asynchronous sequential circuits? (Dec. 2012)
45. What is primitive flow table? (May 2013)
46. What are static '1' ans static '0' hazards? (May 2013)

PART-B

- Design an asynchronous sequential circuit with two inputs X and Y and with one output Z . Whenever Y is 1, input X is transferred to Z . When $Y=0$, the output does not change for any change in X . Use SR latch for implementation of the circuit. (16) (April 2003)
- (i) What is a fundamental mode sequential circuit. (2)
 (ii) Define cycle. (2)
 (iii) Write notes on Races and Hazards. (6 + 6) (April 2003)
- A pulse mode asynchronous machine has two inputs. It produces an output whenever two consecutive pulses occur on one input line only. The output remain at 1 until a pulse has occurred on the other input line. Write down the state table for the machine. (16) (April 2003)
- Implement the switching function $F = \sum (0, 1, 3, 4, 8-12)$ by a state hazard free two level OR-AND gate network. (16) (April 2003)
- Show that no static 0(Static 1) hazard can happen in a two level AND-OR (OR-AND) realisation of a switching function F . (16) (April 2003)
- What is meant by hazard. Differentiate between static, dynamic and essential hazard. (8) (Nov. 2003)
- Write notes on the following giving one example for each:
 (i) Stable state, (ii) Unstable state, (iii) Cycles, (iv) Race. (16) (Nov. 2003)
- Implement the switching function $x_1\bar{x}_2y_1 + \bar{x}_1x_2y_2$ by a static hazard free two level AND-OR gate network. (16) (April 2004)

9. Design an asynchronous binary toggle circuit that changes state with each rising edge of the clock input. Assume the initial output as 0. **(16) (April 2004)**
10. (i) How will you minimise the number of rows in the primitive state table of an incompletely specified sequential machine? **(12)**
(ii) State the restrictions on the pulse width in a pulse mode asynchronous sequential machine. **(4) (April 2004)**
11. Find a static and dynamic hazard free realization for the following function using
(i) NAND gates, (ii) NOR gates $F(a,b,c,d)=\sum m(1,5,7,14,15)$. **(16) (April 2004)**
12. What is Race-around condition in latches? How is it overcome? Explain. **(16) (Nov. 2004)**
13. Obtain the primitive flow table for an asynchronous circuit that has two inputs x, y and one output z . An output $z = 1$ is to occur only during the input state $xy = 01$ and then if and only if the input state $xy = 01$ is preceded by the input sequence $xy = 01, \underline{00}, 10, 00, 10, 00$. **(16) (Nov. 2004)**
14. Implement the switching function $f(x_1x_2x_3) = x_1x_2 + x_2x_3$ by a static hazard free 2 level AND-OR gate network. **(8)**
15. Show that dynamic hazards do not occur in 2 level AND-OR gate networks. **(8) (Nov. 2004)**
16. Write short notes on asynchronous sequential circuits. **(8) (Nov. 2004)**
17. (i) What is race around condition? How is it avoided? **(6)**
(ii) Draw the schematic diagram of master-slave JK FF and input and output waveforms. Discuss how does it prevent race around condition. **(10) (April 2005)**
18. (i) Explain races and hazards with suitable examples. **(8)**
(ii) Discuss methods of designing race free and hazard free circuits with examples. **(8) (April 2005)**
19. (i) What are the 2 types of asynchronous circuits? Differentiate between them. **(6)**
(ii) An asynchronous sequential circuit is described by the following excitation and output function
- $$Y = X_1 \cdot X_2 + (X_1 + X_2) \cdot Y ; Z = Y$$
- Draw the logic diagram of the circuit. Derive the transition table and output map. Describe the behaviour of the circuit. **(10) (Dec. 2005)**
20. Implement a scalar with a period of 24 by technique of asynchronous coupling and explain the procedure. **(16) (Dec. 2005)**
21. Describe the hazards that could occur in asynchronous sequential circuits. What are the ways in which they get eliminated? **(16) (Dec. 2005)**
22. Explain the principle of pulse mode asynchronous sequential logic circuit. What are the restrictions to be laid on the input signal of a pulse mode asynchronous sequential circuit? **(Dec. 2005)**
23. Design an asynchronous sequential circuit that has two internal states and one output. The excitation and output function describing the circuit are as follows:

$$X = x_1x_2 + x_1y_2 + x_2y_1, \quad Y = x_2 + x_1y_1y_2 + x_1y, \quad Z = x_2 + y_1 \quad \text{(8) (Dec. 2005)}$$

24. (i) Give the hazard free realization for the following functions:

$$f = (a, b, c, d) = \sum m(0, 2, 6, 7, 8, 10, 12) \quad (8)$$

(ii) Explain how can essential hazards determined from the flow table and how can be eliminated from the network. **(8) (Dec. 2005)**

25. Design a circuit with inputs A and B to give an output $Z = 1$ when $AB = 11$ but only if A becomes 1 before B , by drawing total state diagram, primitive flow table and output map in which transient state is included. **(16) (May 2006)**

26. Design a circuit with primary inputs A and B to give an output Z equal to 1 when A becomes 1 if B is already 1. Once $Z = 1$ it will remain so until A goes to 0. Draw waveform diagram, total state diagram, primitive flow table for designing this circuit. **(16) (May 2006)**

27. Describe the steps in the design of asynchronous sequential circuits. Apply these steps to design a T flip flop. **(16) (May 2006)**

- 28.(i) Draw the structure of fundamental mode synchronous sequential logic circuit and define the terms input states, secondary (internal states), excitation variables and stable state. **(10)**

- (ii) Give examples for critical race and cycle and explain. **(6) (May 2006)**

29. Develop state diagram and primitive flow table for a logic system that has 2 inputs, x and y and an output z . And reduce primitive flow table. The behaviour of the circuit is stated as follows. Initially $x = y = 0$. Whenever $x = 1$ and $y = 0$ then $z = 1$, whenever $x = 0$ and $y = 1$ then $z = 0$. When $x = y = 0$ or $x = y = 1$ no change it z it remains in the previous state. The logic system has edge triggered inputs without having a clock. The logic system changes state on the rising edges of the 2 inputs. Static input values are not to have any effect in changing the z output. **(Dec. 2006)**

30. (i) What is the objective of state assignment in asynchronous circuit? Give hazard-free realization for the following Boolean functions.

$$f(A, B, C, D) = \sum M(0, 2, 6, 7, 8, 10, 12) \quad (8) \text{ (Dec. 2006)}$$

- (ii) Summarize the design procedure for asynchronous sequential circuit. **(8) (Dec. 2006)**

31. (i) Write short notes on races and cycles that occur in fundamental mode circuits. **(10)**

- (ii) What is essential hazard? Explain with example. **(6) (May 2007)**

32. (i) Explain how hazard free realization can be obtained for a Boolean function. **(8)**

- (ii) Discuss a method used for race free assignments with examples. **(8) (May 2007)**

33. (i) Give hazard-free realization for the following Boolean functions $f(A, B, C, D) = \sum m(1, 3, 6, 7, 13, 15)$. **(8) (May 2007)**

- (ii) Summarize the design procedure for asynchronous sequential circuit. **(8) (May 2007)**

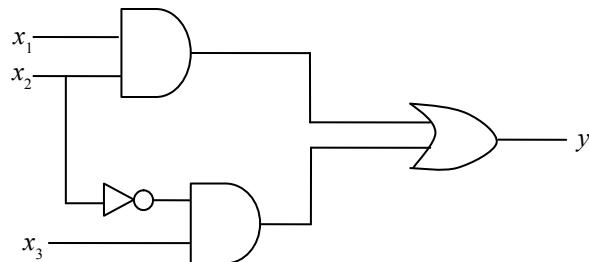
34. An asynchronous sequential circuit is described by the following excitation and output function.

$$Y = X_1 X_2 + (X_1 + X_2) Y$$

- (i) Draw the logic diagram of the circuit.
(ii) Derive the transition table and output map.
(iii) Describe the behaviour of the circuit. **(16)(May 2007)**
- 35.** (i) Explain the need for key debounce circuit. **(8) (Dec. 2007)**
(ii) What is the objective of stage assignment in asynchronous circuit? Give hazard-free realization for the following Boolean function
 $F(A,B,C,D) = \sum M(0,1,5,6,7,9,11)$ **(8) (Dec. 2007)**
- 36.** An synchronous sequential circuit is described by the following excitation and output function.
 $B = (A_1' B_2)B + (A_1 + B_2)$
 $C = B$
(i) Draw the logic diagram of the circuit. **(5) (Dec. 2007)**
(ii) Derive the transition table and output map. **(6) (Dec. 2007)**
(iii) Described the Behaviour of the circuit **(5) (Dec. 2007)**
- 37.** (i) with examples, explain the different types of Asynchronous sequential circuits. **(6) (Dec. 2008)**
(ii) An asynchronous sequential circuit is described by the excitation and output functions
 $Y=x_1x_2' + (x_1+x_2')y$ and $z=y$ where Y and z are excitation and output functions respectively. **(10) (Dec. 2008)**
- 38.** Design an asynchronous sequential circuit with two inputs x_1 and x_2 and one output z . Initially both the inputs are equal to 0. When x_1 or x_2 becomes 1, z becomes 1. When second input also becomes 1, $z = 0$; The output stays at 0 until circuit goes back to initial state. **(16)(May 2009)**
- 39.** Discuss in detail the static hazards. **(16) (May 2009)**
- 40.** Find a static and dynamic hazard free realization for the following function using
(i) NAND gates
(ii) NOR gates.
 $f(a, b, c, d) = \sum m(1, 5, 7, 14, 15)$ **(16) (Dec 2010)**
- 41.** (i) Explain the working principle of switch debounce logic. **(6)**
(ii) Determine whether the circuit is stable or not whose excitation function is given by
 $y = (x_1 y)' x_2$. **(10) (May 2011)**
- 42.** (i) Derive a circuit specified by the following flow table. **(10)**

	xy			
	00	01	10	10
A	A,0	A,0	A,0	B,0
B	A,0	A,0	B,1	B,1

- (ii) Determine whether the following circuit has a static hazard or not. If yes, design a hazard - free logic for the same input and output relation. **(6) (May 2011)**



43. Write a detailed note on Race free state assignment. **(16) (Dec 2011)**
44. With suitable design example. Explain ASM Chart. **(16) (Dec 2011)**
45. Design an asynchronous sequential circuit that has 2 inputs X2 and X1 and one output Z. When X1 = 0, the output Z is 0. The first change in X2 that occurs while X1 is 1 will cause output Z to be 1. The output Z will remain 1 until X1 returns to 0. **(16) (May 2012)**
46. Find a circuit that has no static hazards and implements the Boolean function $F(A,B,C,D) = \sum m(1,3,5,7,8,9,14,15)$ **(16) (May 2012)**
47. Explain race-free state assignment with an example. **(16) (Dec 2012)**
48. Write detailed notes on hazards in combinational circuits and sequential circuits. **(16) (Dec 2012)**
49. (i) Explain the types of hazards in digital circuits.
(ii) Implement the switching function $F = \sum m(1, 3, 5, 7, 8, 9, 14, 15)$ by a static hazard free 2 level AND-OR gate network.
51. Explain the steps for the design of asynchronous sequential circuits. **(16) (May 2013)**

UNIT – V

MEMORY AND PROGRAMMABLE LOGIC

- **RAM and ROM**
- **Memory Decoding**
- **Programmable Logic Array**
- **Programmable Array Logic**
- **Error Detection and Correction**
- **Sequential Programmable Devices**
- **Application Specific Integrated Circuits**

UNIT V

MEMORY AND PROGRAMMABLE LOGIC

5.1 INTRODUCTION

A memory unit is a collection of storage cells with associated circuits needed to transfer information in and out of the device. A memory unit stores binary information in group of bits called words. A word in memory is an entity of bits that move in and out of storage as a unit. A word is a group of 1's and 0's and may represent a number, an instruction, one or more alphanumeric characters or any other binary-coded information. A group of 8 bits is called a **byte**. The byte can be split into two 4-bit units that are called **nibbles**. The **capacity** of a memory unit is the total number of bytes that can be stored. Suppose the memory capacity is 1024×8 , it means that the number of words is 1024 and the number of bits per word is 8.

Each word stored in a memory location is represented by an **address**.

5.2 MEMORY UNIT

The communication between a memory and its environment is achieved through data input/output lines, address selection lines and control lines that specify the direction of transfer. A block diagram of the memory unit is shown in **Figure 5.1**. The '*n*' data input lines provide the information to be stored in memory and the '*n*' data output lines supply the information coming out of memory. The '*K*' address lines specify the particular word chosen among the many variable. The two control inputs (read and write) specify the direction of transfer desired. The write input causes binary data to be transferred into the memory and the read input causes binary data to be transferred out of memory.

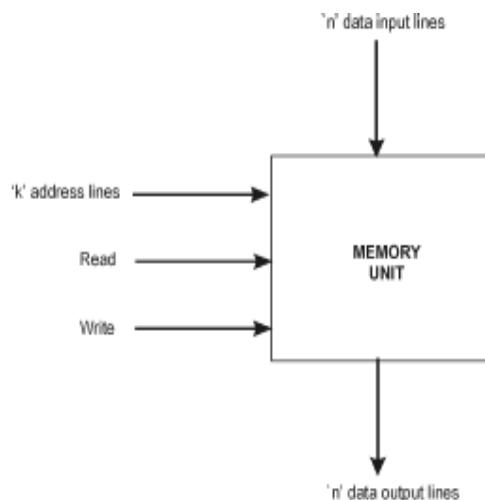


Fig. 5.1: Block diagram of a memory unit

Data units go into the memory and come out of the memory on a set of lines called data bus. The data bus is bi-directional, which means that data can go in either direction. For a write or a read operation, an address is selected by placing a binary code representing the desired address on a set of lines called the address bus. The address code is decoded internally and the appropriate address is selected. The number of lines in the address bus depends on the capacity of the memory. For example, a 16 bit address code can select 65536 locations (2^{16}) in the memory. The block diagram of memory operation is shown in **Figure 5.2**.

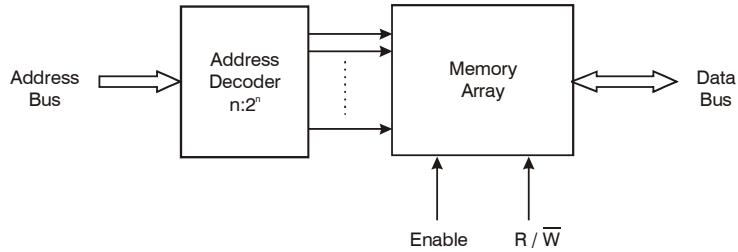


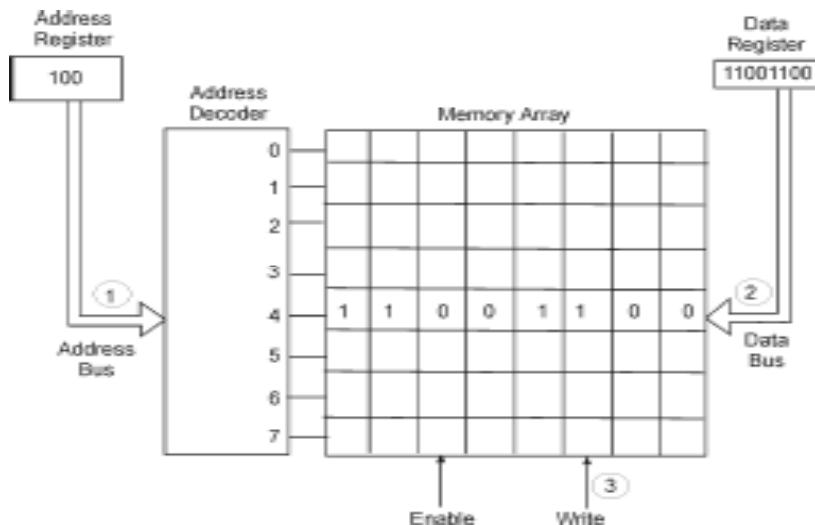
Fig. 5.2: Block diagram of memory operation

5.3 WRITE OPERATION

During write operation, the binary data are stored in a specified memory location. When R / W control input is low, write operation is performed. The following steps are followed to write a content in the memory:

1. The address decoder decodes the address for specified memory location based on the given input of the decoder.
2. Data is placed on the data bus.
3. The data is written in above specified location when the memory chip receives the write signal.

The write operation is explained in the **Figure 5.3**.



1. Address code 100 is placed on the address bus and address 4 is selected.
2. Data byte is placed on the data bus.
3. Write command causes the data byte to be stored in address 4, replacing previous data.

Fig. 5.3: Write Operation

5.4 READ OPERATION

During read operation, the required contents are read from the specific memory location. When the R/\bar{W} control input is high, the read operation is performed. The following steps are followed during the read operation:

1. The address decoder decodes the specified memory location based on the input of the decoder.
2. The read command is sent to memory.
3. The content of specified memory location is placed on the data bus.

The read operation is illustrated in **Figure 5.4**.

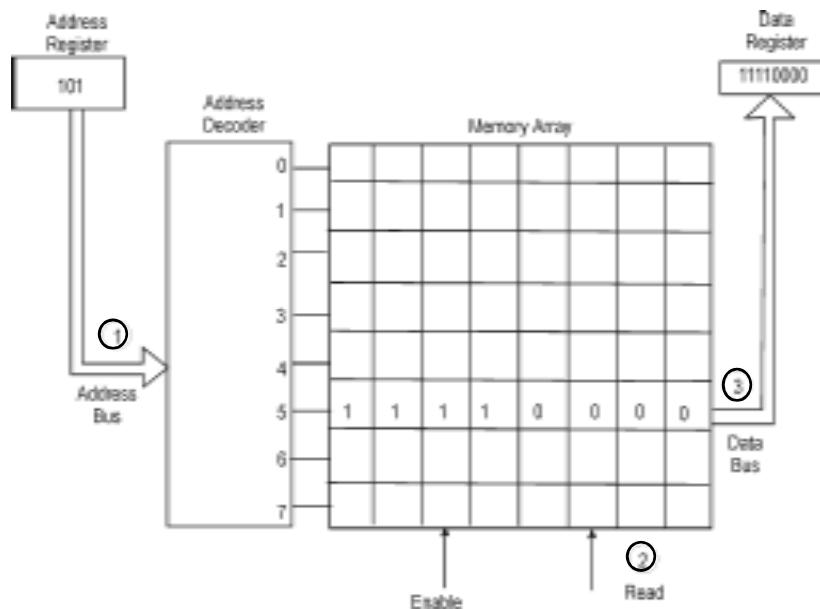


Fig. 5.4 : Read Operation

1. Address code 101 is placed on the address bus and address 5 is selected.
2. Read command is applied.
3. The contents of address 5 is placed on the data bus and shifted into data register.

5.5 CLASSIFICATION OF MEMORIES

Memories are usually classified as either bipolar or metal oxide semiconductor (MOS) or complementary MOS (CMOS) according to the type of transistor used to construct the memory cells. The operation of bipolar memories is fast but greater packing density. MOS and CMOS memories are cheap and small in size and require low power.

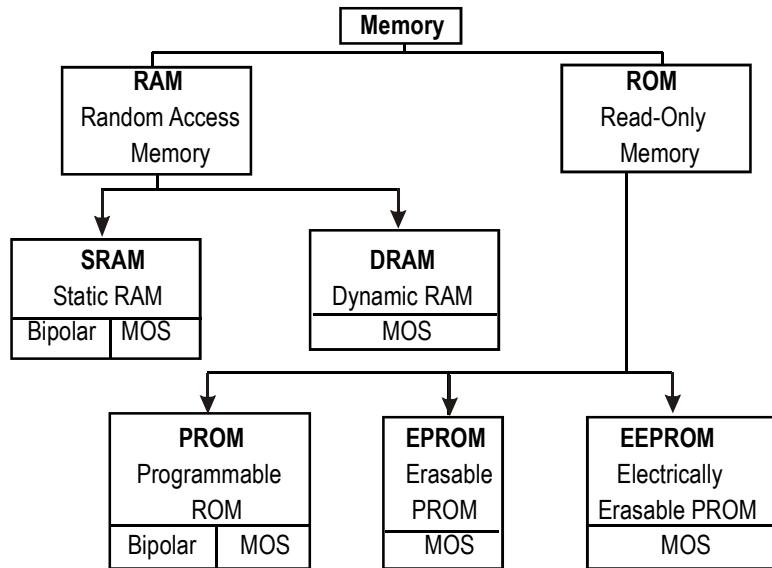


Fig. 5.5: Classification of Memories

The two general categories of memory, RAM and ROM can be further divided as illustrated in **Figure 5.5**.

5.5.1 RAM

Random Access Memory is a volatile chip memory in which both read and write operations can be performed. RAMS are volatile because the stored data will be lost once the d.c. power applied to the flip-flops is removed.

Random access means a bit (0 or 1) can be written (stored) in any cell or read (detected) from any cell. A control signal (Enable) is used to enable or disable the chip. In the read mode, data from the selected memory cells is made available at the output. In the write mode, information at the data input is written into the selected memory cells. The address lines determine the cells written into or read from. A typical RAM chip is shown in **Figure 5.6**.

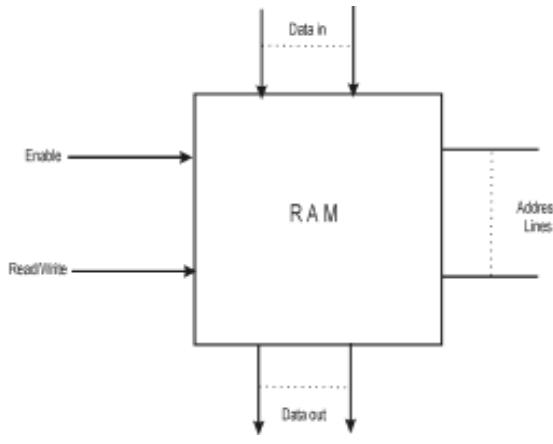


Fig. 5.6 : RAM Chip

5.5.2 STATIC RAM

Static RAMs use flipflops as storage elements and can therefore store data indefinitely as long as d.c. power is applied. SRAMs can be of either Bipolar or MOS technology.

5.5.3 Dynamic RAM

Dynamic RAMs use capacitors as storage elements and cannot retain data very long without the capacitors being recharged by a process called refreshing. DRAMs can store much more data than SRAMs. DRAMs are available in only MOS technology.

5.5.4 ROM

Read Only Memory (ROM) is a semi conductor memory device used to store the information permanently. It performs only read operation and does not have a write operation. A ROM stores data that are used repeatedly in system applications, such as tables, conversions or programmed instructions for system initialization and the user cannot alter its function.

5.5.5 PROM

The PROM, Programmable ROM is the type in which the data are electrically stored by the user with the aid of specialized equipment but cannot be reprogrammed. PROMs are available in both bipolar and MOS technologies.

5.5.6 EPROM

A PROM device that can be erased and reprogrammed is called Erasable PROM (EPROM). It is strictly a MOS device.

To program a new data, all cells in the EPROM must be erased. This is done by illuminating cells by a strong ultraviolet (UV) light. EPROMS are provided with a transparent quartz window on the top of the chip to allow UV rays for erasing the data. Changes in the selected memory locations cannot be made in the reprogramming. The entire memory should be erased before reprogramming.

5.5.7 EEPROM

Electrically Erasable PROM (EEPROM) can be erased and programmed by the application of controlled electric pulses to the IC in the circuit. The changes can be made in the selected memory locations without disturbing the correct data in other memory locations.

5.6 RAM ORGANIZATION

Figure 5.7 shows the organization of a typical $16K \times 1$ SRAM. The memory cell array is arranged in 128 rows and 128 columns. The chip select, CS must be low for the memory to operate. Seven of the 14 address lines are decoded by the row decoder to select one of the 128 rows. Seven of the 14 address lines are decoded by the column decoder to select one of the 128 columns.

If \overline{CS} and R/W are low, then it is a write operation. In write operation, data on the D_{in} input is written into the addressed cell while the output will remain in the high impedance state. For read operation \overline{CS} signal must be low and R/W signal must be high. In read operation, data from addressed cell appears at the output while the input buffer is disabled.

When \overline{CS} is high both input and output buffers are disabled and the chip is electrically disconnected. This makes the IC to operate in power down mode.

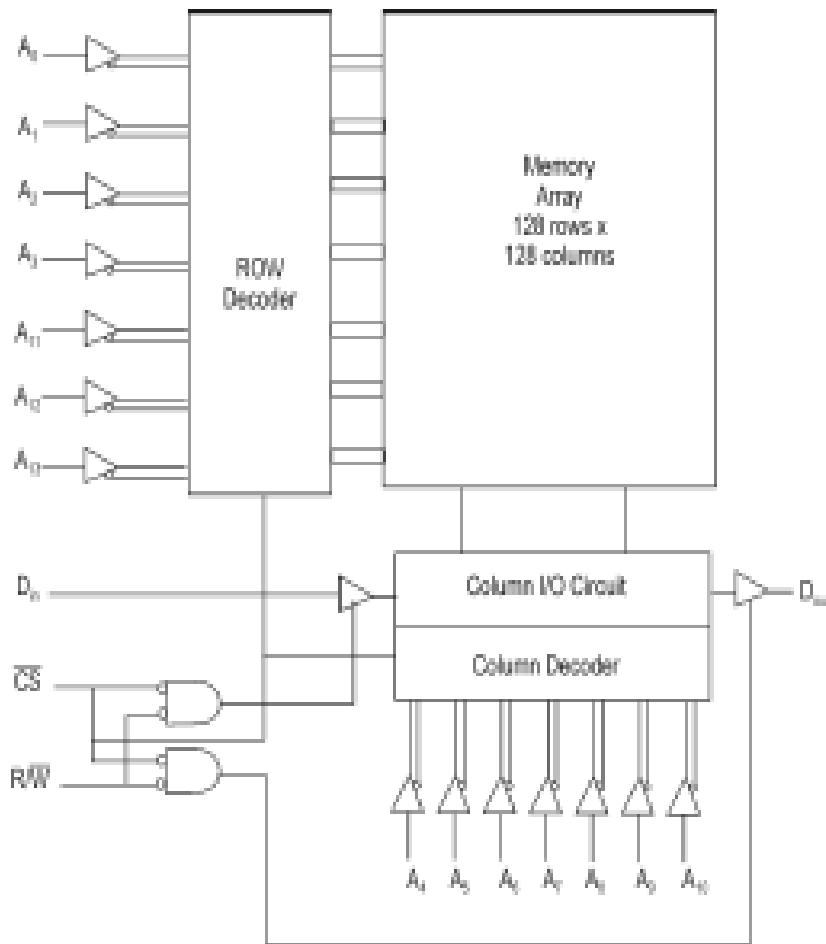


Fig. 5.7 : RAM organization (16K × 1)

5.7 STATIC RAM CELL

The cell is a single storage element in the memory. The cell is selected by high values on the row and column lines. The input data bit (0 or 1) is written into the cell by setting the flip-flop for a 1 and resetting the flip-flop for a 0 when R/W line is low. When R/W line is high, the flip-flop is unaffected.

It means that the stored bit (data) is gated to the Data out line. The logic diagram of a static RAM cell is shown in **Figure 5.8**. The flip-flop is static RAM cell can be constructed using BJT (Bipolar technology) and MOSFET (MOS technology).

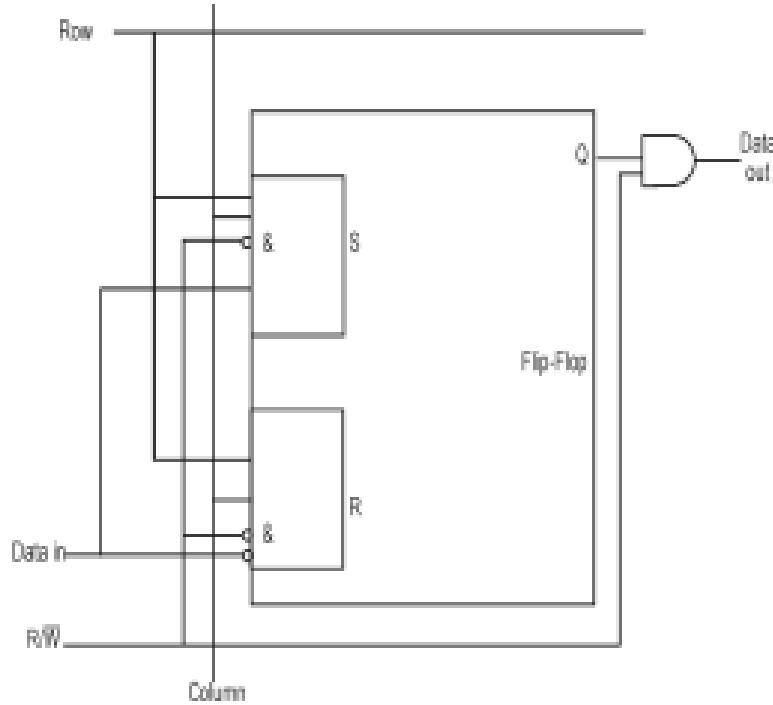


Fig. 5.8 : Logic diagram of a static RAM cell

5.8 BIPOLAR RAM CELL

The Bipolar memory cell is implemented using TTL (Transistor-Transistor-Logic) multiple emitter technology. It stores 1 bit information. It is nothing but a flip-flop. It can store either 0 or 1 as long as power is applied and it can set or reset to store either 1 or 0, respectively. In a bipolar static RAM cell shown in **Figure 5.9**, two BJTs Q_1 and Q_2 are cross coupled to form a flip-flop. Each transistor has three emitters, namely Row select input, Column select input and Write input. To select the cell, both the row and column select lines must be held high. When selected a data bit can be stored in the cell (Write operation) or the content of the cell can be read (Read operation). If either row or column select line is low, then the memory cell is disabled.

The Row select line and Column select line select a cell from matrix. The Q_1 and Q_2 are cross coupled, hence one is always OFF while the other is ON. A '1' is stored in the cell if Q_1 is conducting and Q_2 is OFF. A '0' is stored in the cell if Q_2 is conducting and Q_1 is OFF. The state of the cell is changed to '0' by pulsing a high on the Q_1 (write input) emitter. This turns OFF Q_1 . When Q_1 is turned OFF, Q_2 is turned ON. As long as Q_2 is ON, its collector is low and Q_1 is held OFF.

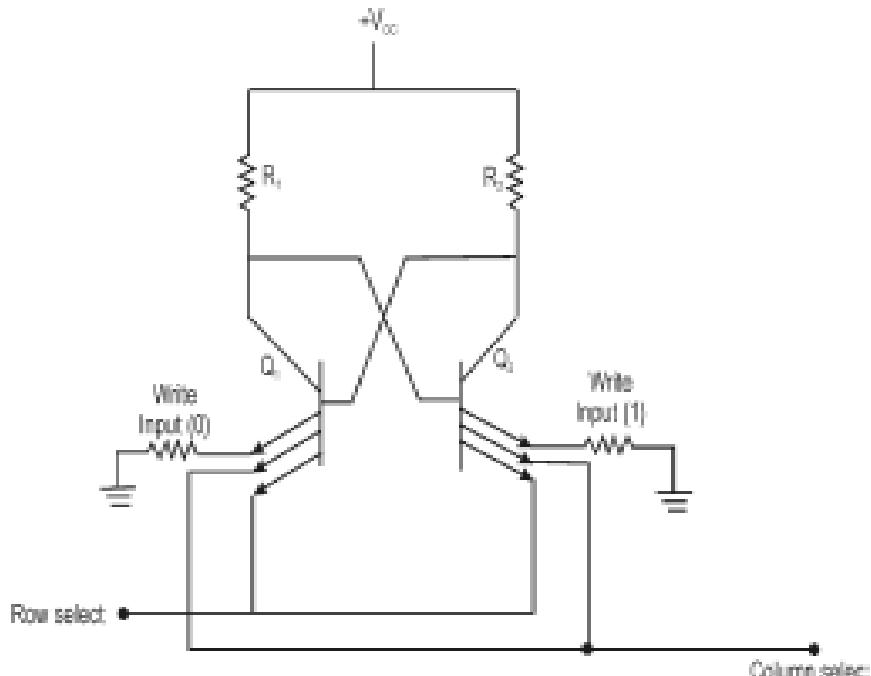


Fig. 5.9 : Bipolar SRAM cell

A 1 can be rewritten by pulsing the Q_2 (write input) emitter high. Large number of these memory cells are organized on a row and column basis to form a memory chip.

5.9 MOSFET RAM CELL

Each cell of static RAM is constructed by using flipflops. The number of flipflops is equal to the number of cells in a memory. Since the number of cells is usually very large, so it is required to use minimum devices per cell to occupy minimum chip area per cell in order to have more number of cells per chip. This reduces cost, shortens access time etc. Therefore these flip-flops are constructed using MOSFETs.

If the MOSFET T_1 is turned ON then the output Q is low, which means that MOSFET T_2 is cut OFF. Since T_2 is cut off, the output is high, ensuring that T_1 is turned ON. Thus we have a static situation as long as the bias voltage $+V_{CC}$ is applied to the circuit. The MOSFETs T_A and T_B are connect the memory cell to the normal and complementary data lines (Data and $\overline{\text{Data}}$). The MOSFETs T_3 and T_4 are act as load resistors.

When the bit select line is LOW, both T_A and T_B are cut off and the memory cell is isolated. The data stored in the cell remain stored as long as power is applied to the cell. When the bit select line is HIGH, the memory cell is connected to the data lines so that the data in the cell can be read or new data can be written into the cell. The MOS RAM cell is shown in **Figure 5.10**.

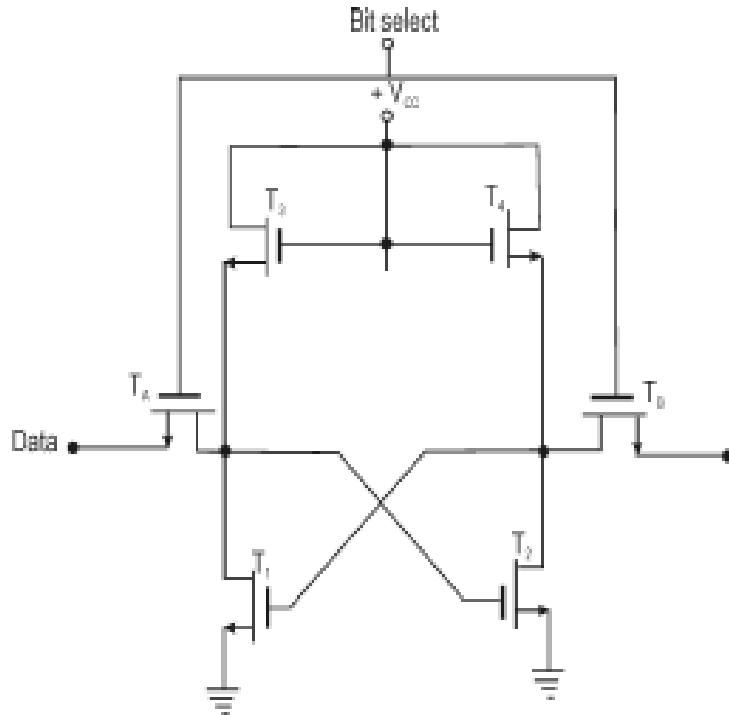


Fig. 5.10 : MOS RAM cell

5.10 DYNAMIC RAM CELL

The dynamic RAM (DRAM) stores its binary information in the form of electric charges on capacitors. The basic storage device in DRAM is not a flip-flop but a simple MOSFET and a capacitor. The advantage of DRAM is that it is very simple, thus allowing very large memory arrays to be constructed on a chip at a lower cost per bit. The disadvantage of DRAM is that the storage capacitor cannot hold its charge over an extended period of time and will lose the stored data bit unless its charge is refreshed periodically.

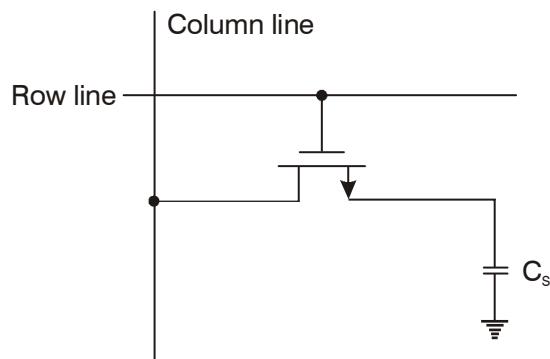


Fig. 5.11 : DRAM cell

In DRAM memory cell, a bit of data is stored as charge on storage capacitor, where the presence or absence of charge determines the value of the stored bit 1 or 0. The DRAM cell includes a MOSFET and a storage capacitor (C_s) as shown in **Figure 5.11**. When column line and row line go high, the MOSFET conducts and charges the capacitor. When column and row lines go low, the MOSFET opens and the capacitor retains its charge. In this way it stores 1 bit.

5.10.1 Operation

The DRAM cell consists of 3 tri-state buffers: Input buffer, Output buffer and Refresh buffer. Input and output buffers are enabled and disabled by controlling R/\bar{W} line. When $R/\bar{W} = 0$, input buffer is enabled and output buffer is disabled. When $R/\bar{W} = 1$, input buffer is disabled and output buffer is enabled.

(i) Write Operation

The write operation is illustrated in **Figure 5.12**. To enable write operation R/\bar{W} line is made low, which enables input buffer and disables output buffer. To write a 1 into the cell, the D_{IN} line is high and MOSFET 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. When the Row line is made Low, the transistor turns OFF and disconnects the capacitor from the data line, thus storing the charge (1 or 0) on the capacitor.

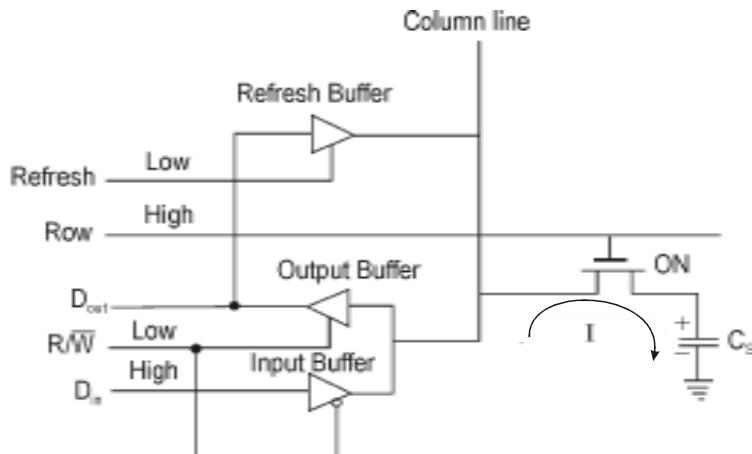


Fig. 5.12 : Writing a 1 into the memory cell

(ii) **Read Operation:** To read data from the cell, the R/\bar{W} line is made high, which enables output buffer and disables input buffer. Then Row line is made high. It turns MOSFET ON and connects the capacitor to the D_{OUT} line through output buffer. Read operation is shown in **Figure 5.13**.

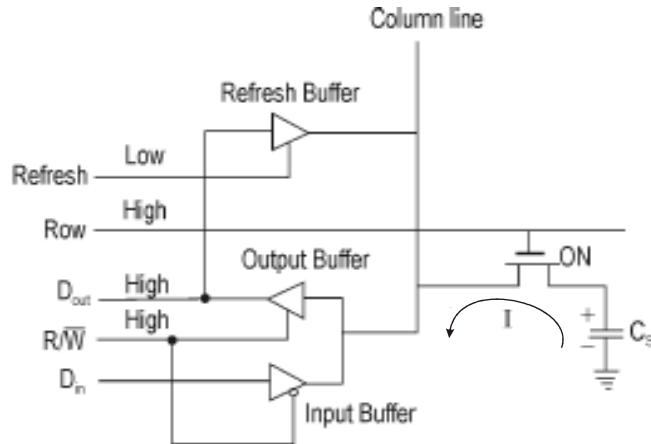


Fig. 5.13 : Reading a 1 from the memory cell

(iii) **Refresh Operation:** To enable refresh operation, R/W Line, Row line and Refresh line are made high. This turns ON MOSFET and connects capacitor to column line. As R/W is high, output buffer is enabled and the stored data bit is applied to the input of refresh buffer. The enabled refresh buffer then produces a voltage on column line corresponding to the stored bit and thus replenishing the capacitor. This refresh operation is illustrated in **Figure 5.14**.

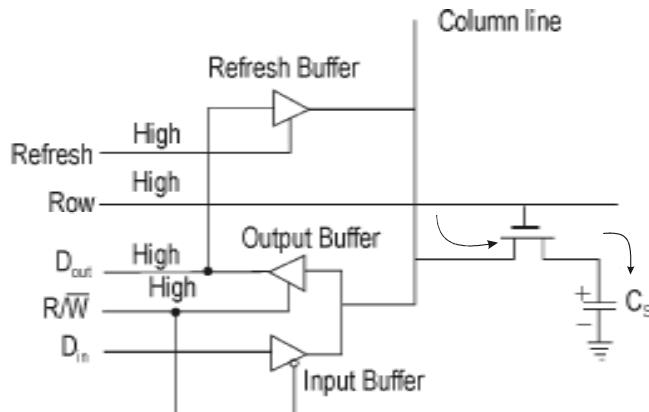


Fig. 5.14 : Refreshing a stored 1

5.10.2 DRAM Organization

In DRAM address multiplexing is used to reduce the number of address lines. The block diagram of 16K DRAM is shown in **Figure 5.15**. $16K = 16 \times 1024 = 16384$ bits = 2^{14} , therefore 14 bit address is applied to address inputs. First, the seven-bit Row address has to be applied and RAS (Row Address Strobe) line latches the seven bits into the Row address latch.

Next, the seven bit column address is applied to the address inputs and \overline{CAS} (Column Address Strobe) latches the remaining 7 bits into the column address latch. Then the 7 bit Row address and the

5.12

Digital Principles and System Design

7 bit column address are decoded to select the appropriate memory cell in the 128×128 dynamic memory array for a Read or Write operation.

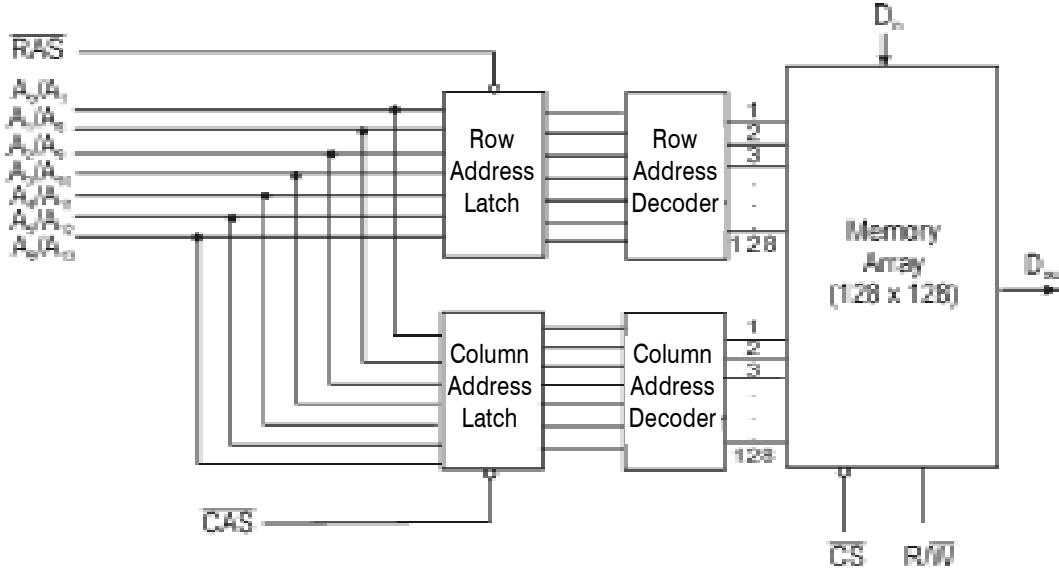


Fig. 5.15 : Block diagram of 16K DRAM

5.11 ROM

It is a Read Only Memory. ROM is a memory device in which permanent binary information is stored. The binary information must be specified by the designer and is then embedded in the unit to form the required interconnection pattern. Once the pattern is established, it stays within the unit even when power is turned off and on again, i.e., it is non-volatile memory. The ROMs are classified as follows:

- (i) Masked ROM or ROM
- (ii) Programmed ROM (PROM)
- (iii) Erasable PROM (EPROM)
- (iv) Electrically Erasable PROM (EEPROM)

5.12 ROM CELL

5.12.1 Diode ROM

The presence of a connection from a row line to the anode of the diode represents a 1 at that location because when the row line is taken HIGH, all diodes are turn ON and connect the HIGH (1) to the associated column lines. The ROM cell using diode is shown in **Figure 5.16**.

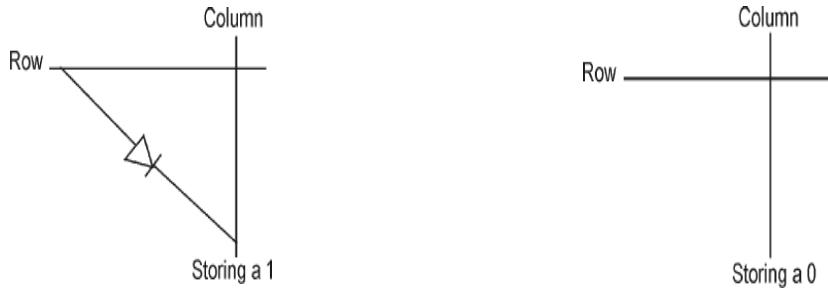


Fig. 5.16 : ROM cell using diode

5.12.2 Bipolar ROM

The presence of a connection from a row line to the base of a transistor represents a 1 at that location because, when the row line is taken HIGH, all transistors with base connection to that row line turn ON and connect the high (1) to the associated column lines. When there are no base connection at row or column, the column lines remain low (0) when the row is addressed.

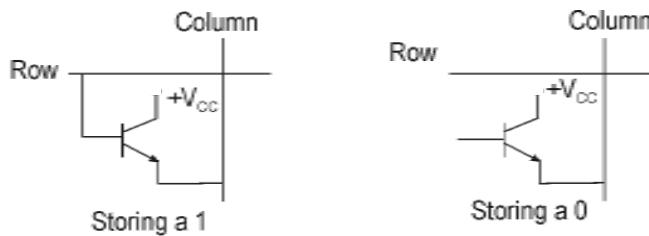


Fig. 5.17 : Bipolar ROM cell

5.12.3 MOS ROM

The presence of a connection from a row line to the gate of a MOSFET represents a 1 at that location because when the row line is taken high, all MOSFETs with a gate connection to that row line turn ON and connect the high (1) to the associated column lines. At row/column junctions where there are no gate connections, the column lines remain low (0) when the row is addressed. The MOS cells are shown in **Figure 5.18**.

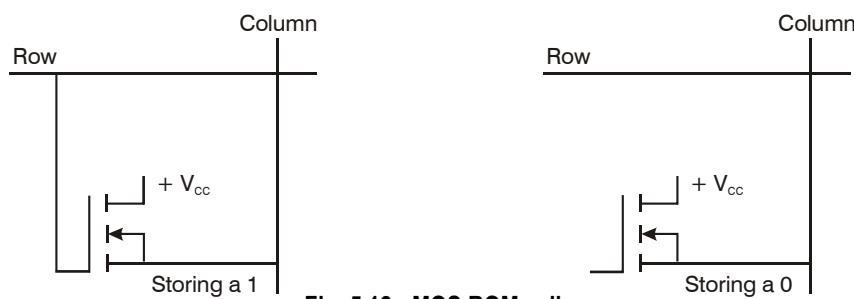


Fig. 5.18 : MOS ROM cell

5.13 ROM ORGANIZATION

A simple ROM organization is shown in **Figure 5.19**. The dark squares represent ROM cells stored 1s and the light squares represent ROM cells stored 0s. When a 4-bit binary address is applied to the address inputs, the corresponding row line become high. This high is connected to the column lines through the transistors at each junction (cell) where a 1 is stored. At each cell where a 0 is stored, the column line stays low because of the terminating resistor. The column lines form the data output. Thus the 8 data bits stored in the selected row appear on the output lines.

A 16×8 ROM organization is shown in **Figure 5.19**. It is organized into 16 addresses, each of which stores 8 data bits. Its total capacity is 128 bits (16 bytes).

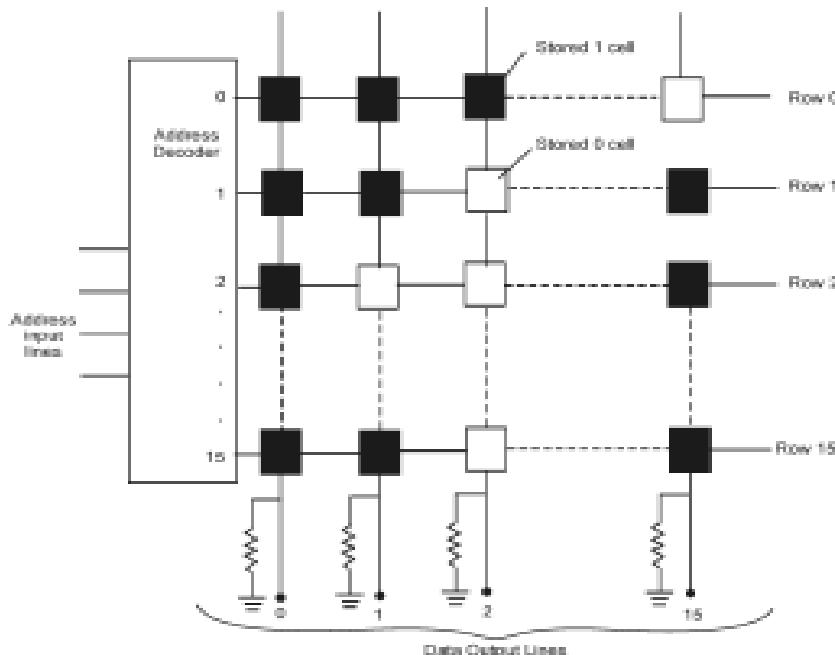


Fig. 5.19 : 16×8 ROM

A typical TTL ROM IC 74187 is shown in **Figure 5.20**. It is a 16 pin, 1024 bits, 256 words of 4 bits each ROM. The memory cells are organized in a 32×32 matrix. Five of the eight address lines ($A_0 - A_4$) are decoded by the row decoder to select one of the 32 rows. Three of the eight address lines ($A_5 - A_7$) are decoded by the column decoder to select four of the 32 columns. The result of this structure is that when an 8 bit address code ($A_0 - A_7$) is applied, a 4 bit data word appears on the data outputs when the chip enable lines (\overline{E}_0 and \overline{E}_1) are LOW to enable the output buffers.

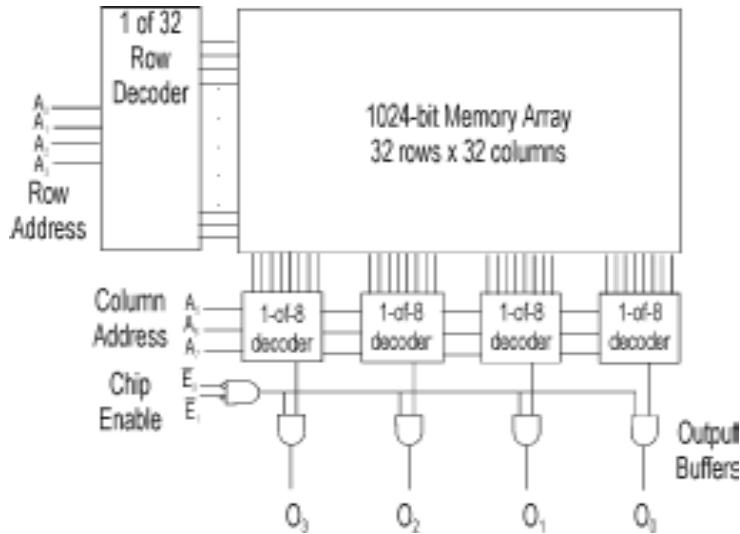


Fig. 5.20 : 1024 bit ROM

5.14 PROM

Programmable ROM (PROM) can be programmed electrically by the user but cannot be reprogrammed. A PROM uses some type of fusing process to store bits, in which a memory link is burned open on left intact to represent a 0 or a 1. The using process is irreversible; once a PROM is programmed, it cannot be changed.

The fusible links are manufactured into the PROM between the cathode of each diode and its column line as shown in **Figure 5.21**. In the programming process, a sufficient current is injected through the fusible link to burn it open to create a stored 0. The link is left intact for a stored 1.

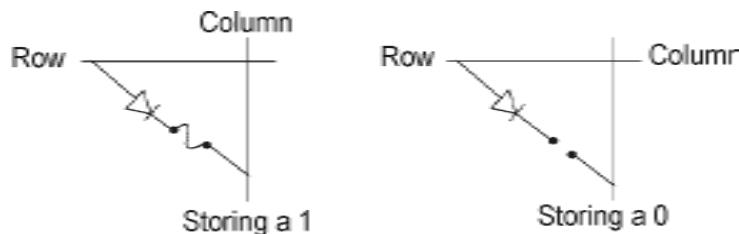


Fig. 5.21 : PROM cell using diodes

In Bipolar PROM, the fusible links are connected between the emitter of each cell's BJT and its column line as shown in **Figure 5.22**.



Fig. 5.22 : Bipolar PROM cell

In MOS PROM, the fusible links are connected between the source of each cell's MOSFET and its column line as shown in **Figure 5.23**.

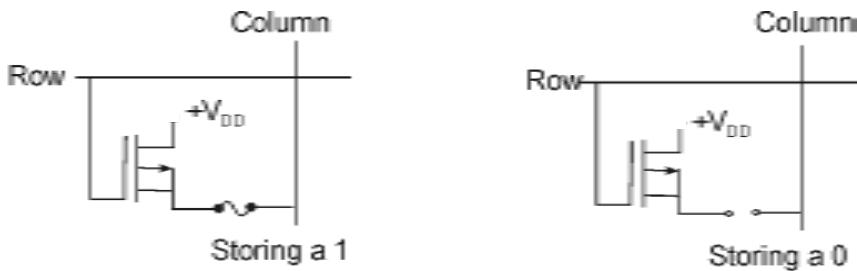


Fig. 5.23 : PROM using MOSFET

A PROM is normally programmed by inserting it into a special instrument called a **PROM programmer**. An address is selected by the electronic switches on the address lines and then a pulse is applied to those output lines corresponding to bit locations where 0s are to be stored (the PROM starts out with all 1s). These pulses blow the fusible links, thus creating the desired bit pattern. The next address is then selected and the process is repeated. This sequence is automatically done by a software-driven PROM programmer.

5.14.1 Fuse Technologies

The fuse technologies used in PROMs are metal links, silicon links and P-N junctions. A brief description of each of these follows:

- (i) Metal links are made by Nichrome. Each bit in the memory array is represented by a separate link. During programming, the link is either blown open or left intact. This is done by forcing a sufficient amount of current through the link to cause it to open.
- (ii) Silicon links are formed by narrow, notched strips of polycrystalline silicon. Programming of these fuses requires melting of the links by passing a sufficient amount of current through them. This current causes a high temperature at the fuse location that oxidizes the silicon and forms an insulation around the new-open link.

- (iii) Shorted junction or avalanche-induced migration technology consists basically of two $p-n$ junctions arranged back-to-back. During programming, one of the diode junctions is avalanched and the resulting voltage and heat short the junction. The remaining junction is then used as a forward biased diode to represent a data bit.

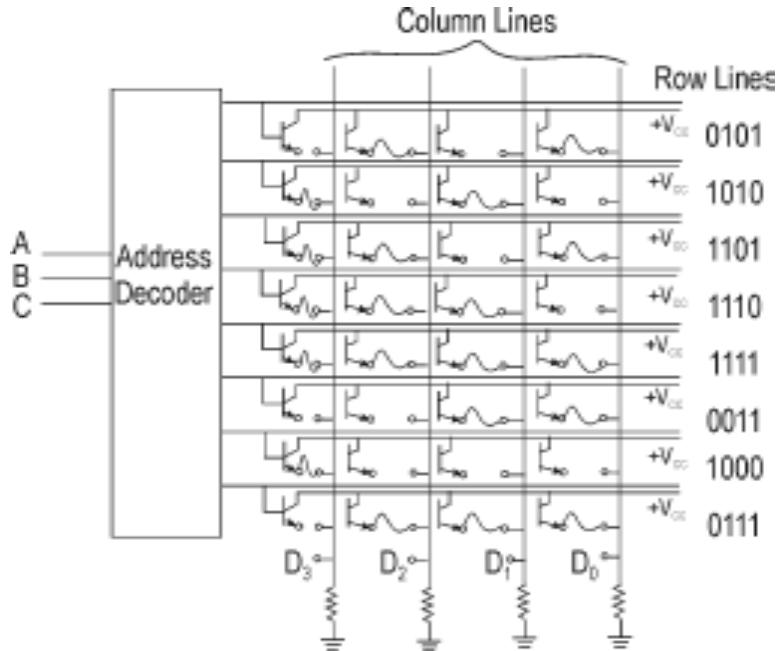


Fig. 5.24 : PROM with fusible links

5.15 EPROM

An EPROM is an erasable PROM. EPROMs can be reprogrammed by the user with a special EPROM programmer. We can erase the stored data in EPROM.

An EPROM uses an array of N-channel enhancement type MOSFET with an isolated gate structure. The isolated transistor gate (floating gate) has no electrical connections and can store an electrical charge for indefinite periods of time. The data bits in this type of array are represented by the presence or absence of a stored gate charge. Erasure of a data bit is a process that removes the gate charge. **Figure 5.25** shows the basic structure and symbol of a typical EPROM cell.

Consider the programming of an EPROM with 1s as initial values in all the cells. To program or store a 0 in a such a cell, the floating gate must be charged. For this, a high voltage of about 16 to 20 volts is applied between the source and drain, and a voltage of about 25 to 50 volts is applied to the control gate for a specified amount of time (50 ms per address location). Due to high electric field established by the positive control gate voltage, the high energy electrons penetrate the thin insulating SiO_2 and reach the floating gate. Thus the charge is stored on the floating gate. Since more negative charge accumulates on the floating gate, the electric field strength is reduced and thereby further accumulation is inhibited. Programming actually involves selecting the desired cell gates and repeatedly

injecting charge onto the floating gate until a sufficient amount of charge is trapped. Since the gate is surrounded by SiO_2 , there is no discharge path available. Therefore, the charge remains trapped on the floating gate for an indefinite period of time. Now the cell is programmed for a logic 0.

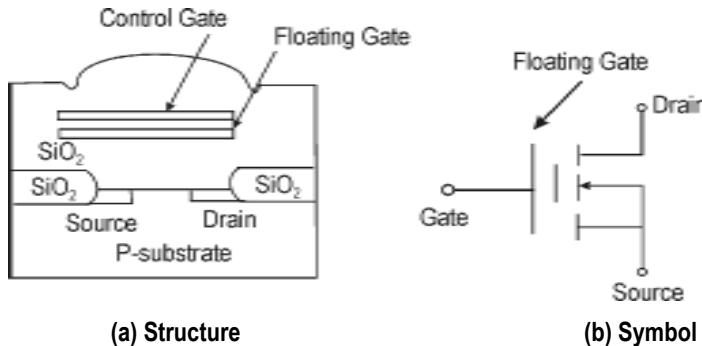


Fig. 5.25 : EPROM cell

To program a different data, all cells in the EPROM must be erased. This is done by electrically or ultraviolet light.

Thus EPROMs are classified as:

- ◆ Ultraviolet Erasable PROM (UV PROM)
- ◆ Electrically Erasable PROM (EE PROM)

5.16 UV EPROM

The UV PROM has a transparent quartz lid on the package. The stored data is erased by exposure of the memory array chip to high-intensity ultraviolet radiation through the quartz window on top of the package for 15 to 20 minutes. The positive charge stored on the gate is neutralized after several minutes of exposure time.

It is not possible to erase selective information, when erased the entire information is lost. The chip can be reprogrammed many times.

5.17 EEPROM

EEPROM (Electrically Erasable PROM) can be both erased and programmed by the application of controlled electric pulses to the IC in the circuit and thereby changes can be made in the selected memory locations without disturbing the correct data in other memory locations. EEPROM is non-volatile memory.

Data is stored as charge or no charge on an insulated layer or an insulated floating gate in the device. The insulating layer is made very thin. Therefore, a voltage as low as 20 to 25 volts can be used to move charges across the thin barrier in either direction for programming or erasing.

EEPROMs are small in size and flexible. With EEPROM, the programs can be altered remotely.

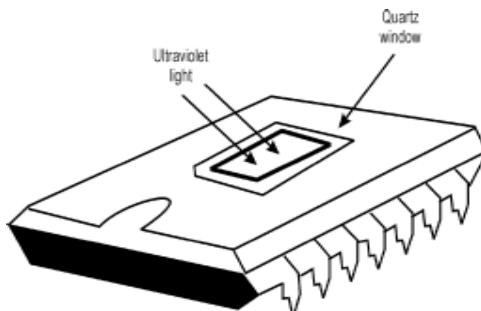


Fig. 5.26 : UV EPROM Chip

5.18 MEMORY CYCLES AND TIMING WAVE FORMS

5.18.1 Read Cycle

The timing wave form for SRAM read cycle is shown in **Figure 5.27**. The timing wave form and timing parameters for SRAM read cycle are identical to ROM read cycle. The timing parameters for read operation in a static RAM are given below:

t_{AA} – **Access time from address.** Assuming that \overline{OE} and \overline{CS} are already asserted, or will be soon enough not to make a difference, this is how long it takes to get stable output data after a change in address.

t_{ACS} – **Access time from chip select.** Assuming that the address and \overline{OE} are already stable, or will be soon enough not to make a difference, this is how long it takes to get stable output data after \overline{CS} is asserted. Often this parameter is identical to t_{AA} , but sometimes it's longer in SRAMs with a “power-down” mode and shorter in SRAMs without one.

t_{OE} – **Output-enable time.** This is how long it takes for the three-stage output buffers to leave the high-impedance state when \overline{OE} and \overline{CS} are both asserted. This parameter is normally less than t_{ACS} , so it is possible for the RAM to start accessing data internally before \overline{OE} is asserted.

t_{OZ} – **Output-disable time.** This is how long it takes for the three-state output buffers to enter the high-impedance state after \overline{OE} or \overline{CS} is negated.

t_{OH} – **Output-hold time.** This parameter specifies how long the output data remains valid after a change in the address inputs.

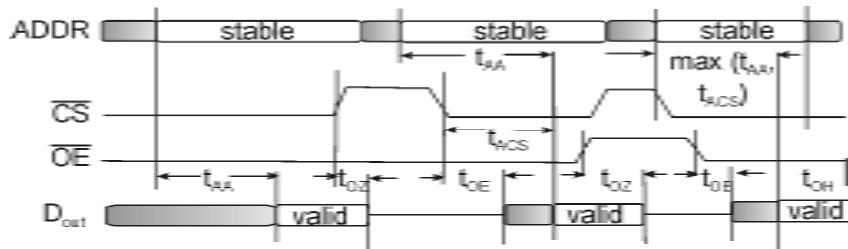


Fig. 5.27: Read cycle ($\overline{WE} = \text{High}$)

5.18.2 Write Cycle

The timing parameters for write cycle are shown in **Figure 5.28** and are described below:

t_{AS} – **Address setup time before write.** All of the address inputs must be stable at this time before both \overline{CS} and \overline{WE} are asserted. Otherwise, the data stored at unpredictable locations may be corrupted.

t_{AH} **Address hold time after write.** Analogous to t_{AS} , all address inputs must be held stable until this time after \overline{CS} or \overline{WE} is negated.

t_{CSW} **Chip-select setup before end of write.** \overline{CS} must be asserted at least this long before the end of the write cycle in order to select a cell.

t_{WP} **Write-pulse width.** \overline{WE} must be asserted at least this long to reliably latch data into the selected cell.

t_{DS} **Data setup time before end of write.** All of the data inputs must be stable at this time before the write cycle ends. Otherwise, the data may not be latched.

t_{DH} **Data hold time after end of write.** Analogous to t_{DS} , all data inputs must be held stable until this time after the write cycle ends.

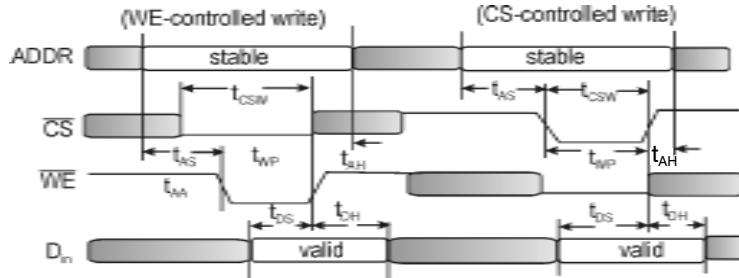


Fig. 5.28 : Write cycle (\overline{WE} = low)

5.19 MEMORY DECODING

In addition to the storage components in a memory unit, there is a need for decoding circuits to select the memory word specified by the input address. In addition to internal decoders, a memory unit may also need external decoders. This happens when RAM ICs are connected in a multichip memory configuration. The use of an external decoder to provide a large capacity memory.

Microprocessor system includes memory devices and I/O devices. Microprocessor can communicate (read/write) with only one device at a time, since the data, address and control buses are common for all the devices. In order to communicate with memory or I/O devices, it is necessary to decode the address from the microprocessor. Due to this, memory or I/O device can be accessed independently. The decoding techniques are

- ◆ Absolute Decoding
- ◆ Linear Decoding

Absolute decoding is normally used in large memory systems. In this technique, all the higher address lines are decoded to select the memory chip and the memory chip is selected only for the specified logic levels on these high-order address lines; no other logic levels can select the chip.

For the design of memory decoder, each memory IC used in the system should be assigned with a range of addresses according to its capacity. The address range assigned to a particular memory IC should not be assigned to some other memory IC in the same digital system.

For example, to select an EPROM and a RAM of 1 K byte capacity each, kept the address assignment in non-overlapped manner. Let the address assigned to EPROM and RAM are as follows:

EPROM (1 K byte) : 0000 H – 03FF H

RAM (1 K byte) : 2000 H – 23FF H

It means that the starting address of the EPROM is 0000 H and the end address is 03FF H while the starting address for RAM is 2000 H and the end address is 23FF H. The range of addresses can be written in binary form as given in **Table 5.1**.

TABLE 5.1 : Address Assignment

Memory	Address (Hex)	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
EPROM Starting address	0000 H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EPROM End address	03FF H	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
RAM Starting address	2000 H	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
RAM End address	23FF H	0	0	1	0	0	0	1	1	1	1	1	1	1	1	1	1

The memory decoder for the given address assignment can be designed for the given address assignment can be designed using a 3 to 8 decoder IC 74LS138 as shown in **Figure 5.29**.

Linear decoding or partial decoding technique is used in small memory systems. In this decoding, address line A_{15} is directly connected to \overline{CS} of EPROM and after inversion using a NOT gate, it is connected to \overline{CS} of RAM. Therefore, when the status of A_{15} line is 0, EPROM gets selected, otherwise RAM gets selected. The status of other address lines is not considered. Let the address assigned to EPROM and RAM are

EPROM (1 K byte) : 0000 H – 03FF H

RAM (1 K byte) : 8000 H – 83FF H

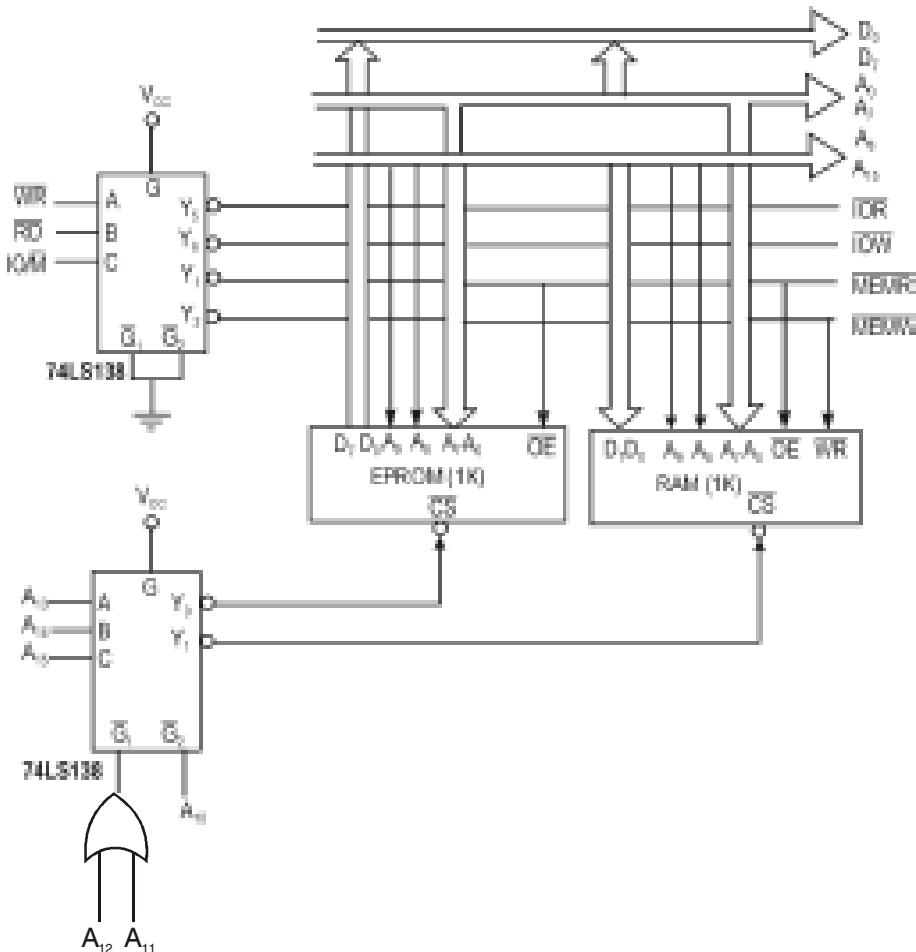


Fig. 5.29 : Absolute decoding technique

5.20 MEMORY EXPANSION

Available memory can be expanded to increase the word length (number of bits in each address) or the word capacity (number of different addresses) or both. Memory expansion is accomplished by adding an appropriate number of memory chips to the address, data and control buses.

5.20.1 Word Length Expansion

To increase the word length of a memory, the number of bits in the data bus must be increased. An 8-bit word length can be achieved by using two memories, each with 4 bit words as shown in **Figure 5.30**. The 16 bit address bus of 64 K ROM is commonly connected to both ROMs so that the combination memory still has the same number of addresses ($2^{16} = 65536$) as each individual memory. The 4 bit data buses from the two memories are combined to form an 8 bit data bus. Now when an address is selected, 8 bits are produced on the data bus - four from each ROM.

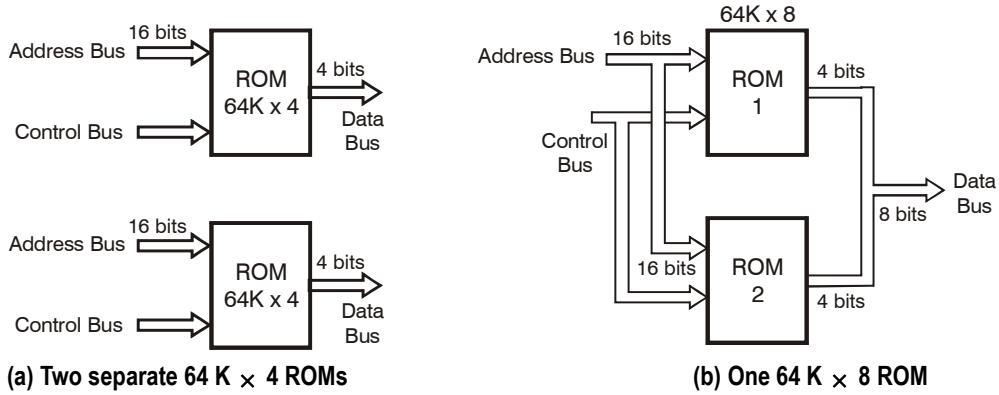
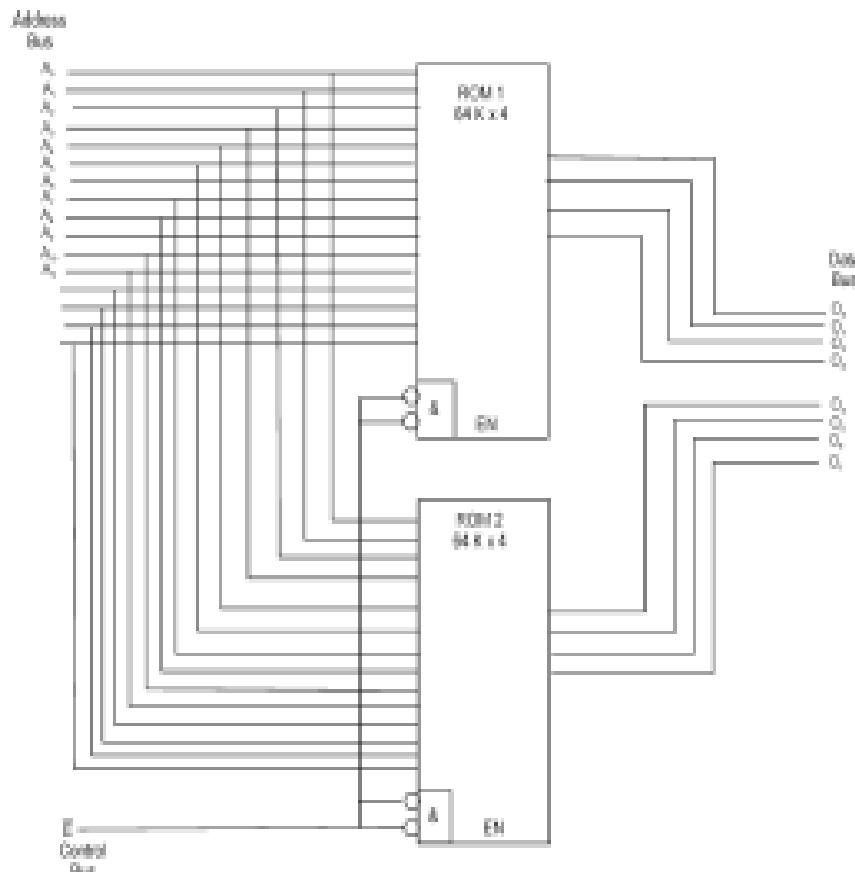
Fig. 5.30 : $64 \text{ K} \times 4$ ROMs into $64 \text{ K} \times 8$ ROM

Figure 5.30(b) is explained with address bus ($A_0 - A_{15}$), control bus (\overline{E}) and data bus ($O_0 - O_7$) is illustrated in **Figure 5.31**.

Fig. 5.31: $64 \text{ K} \times 4$ ROMs to $64 \text{ K} \times 8$ ROM

Example 5.1: Form $64 \text{ K} \times 16$ ROM using $64 \text{ K} \times 4$ ROM.

Solution

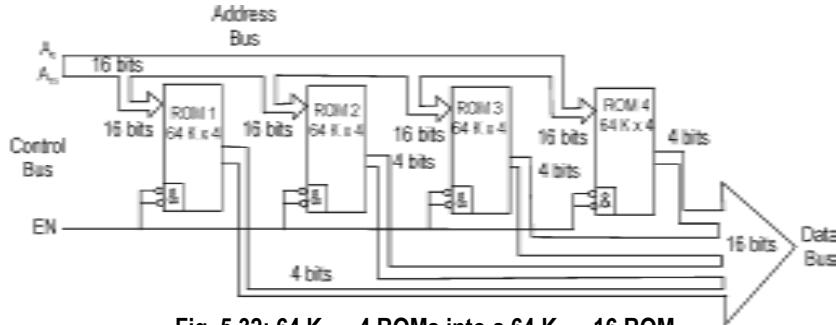


Fig. 5.32: $64 \text{ K} \times 4$ ROMs into a $64 \text{ K} \times 16$ ROM

5.20.2 Word-Capacity Expansion

When memories are expanded to increase the word capacity, the number of addresses is increased. To achieve this increase, the number of address bits must be increased. Two $1\text{M} \times 8$ RAMs are expanded to form a $2\text{M} \times 8$ RAM is shown in **Figure 5.33**. The twenty first address bit is used to enable the appropriate memory chip. The data bus for the expanded memory remains eight bits wide.

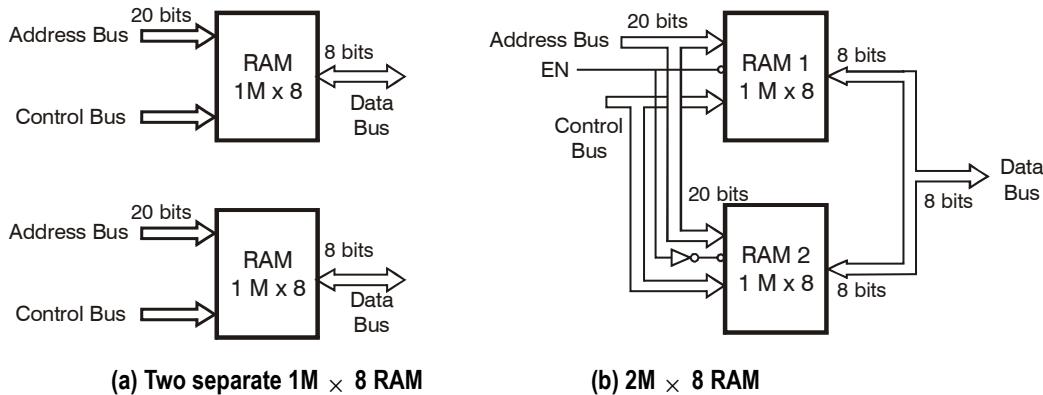


Fig. 5.33: Word Capacity Expansion

Example 5.2: Use $512\text{K} \times 4$ RAMs to implement a $1\text{M} \times 4$ RAM.

Solution: The expanded addressing is achieved by connecting the chip enable (\bar{E}_0) input to the 20th bit (A_{19}).

When $A_{19}(\bar{E}_0)=0$, RAM 1 is selected; RAM 2 is disabled; (A_0-A_{18}) access each of the address in RAM 1.

When $A_{19}(\bar{E}_0)=1$, RAM 2 is enabled using a NOT gate; RAM 1 is disabled; (A_0-A_{18}) access each of the RAM 2 address.

Input \bar{E}_1 is used as an enable input common to both memories.

The $1M \times 4$ RAM implementation is shown in **Figure 5.34**.

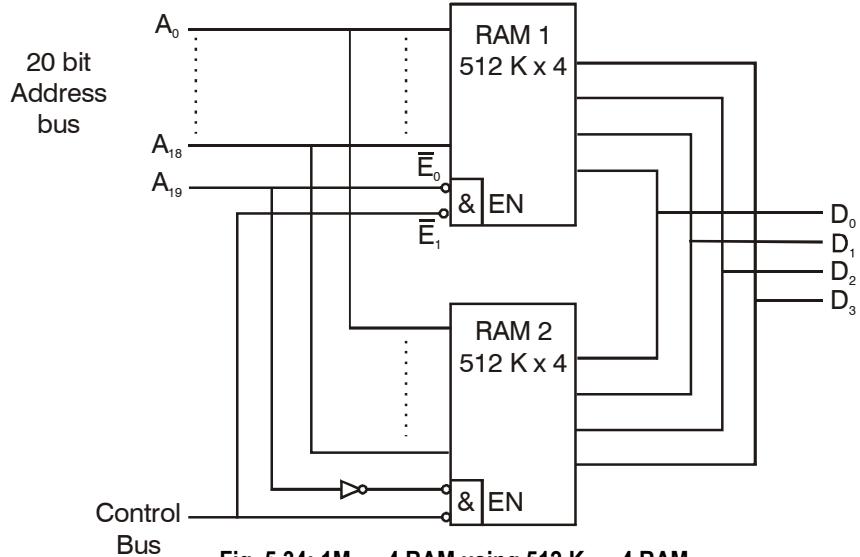


Fig. 5.34: $1M \times 4$ RAM using $512 K \times 4$ RAM

5.21 ADVANTAGES OF RAM

RAM has the following advantages: **1.** Fast operating speed. (< 150 nS), **2.** Low power dissipation (< 1 mW), **3.** Economy, **4.** Compatibility, **5.** Non-destructive read-out.

5.22 ADVANTAGES OF ROM

ROM has the following advantages:

- ◆ Ease and speed of design.
- ◆ Faster than MSI devices PLD and FPGA.
- ◆ The program that generates the ROM contents can easily be structured to handle unusual or undefined cases.
- ◆ A ROM's function is easily modified just by changing the stored pattern, usually without changing any external connections.
- ◆ More economical.

5.23 DISADVANTAGES OF ROM

- ❖ For functions more than 20 inputs, a ROM based circuit is impractical because of the limit on ROM sizes that are available.
- ❖ For simple to moderately complex functions, ROM based circuit may costly; consume more power; run slower.

5.24 COMPARISON BETWEEN RAM AND ROM

RAM	ROM
<p>RAMs have both read and write capability.</p> <p>RAMs are volatile memories. They lose stored data when the power is turned off</p> <p>RAMs are available in both bipolar and MOS technologies</p> <p>Types: SRAM, DRAM, EEPROM</p>	<p>ROMs have only read operation</p> <p>ROMs are non-volatile memories. They retain stored data even if power is turned off</p> <p>ROMs are available in both bipolar and MOS technologies</p> <p>Types: PROM, EPROM</p>

5.25 COMPARISON BETWEEN SRAM AND DRAM

Static RAM	Dynamic RAM
<p>SRAM consists of flipflops. Each flipflop stores one bit.</p> <p>SRAM contains less memory cells per unit area.</p> <p>Its access time is less, hence faster memories.</p> <p>Cost is more.</p> <p>Refreshing circuitry is not required.</p>	<p>DRAM stores the data as charge on the capacitor. It consists of MOSFET and the capacitor for each cell.</p> <p>DRAM contains more memory cells per unit area.</p> <p>Its access time is greater than SRAM.</p> <p>Cost is less.</p> <p>Refreshing circuitry is required.</p>

5.26 COMPARISON OF TYPES OF MEMORIES:

Memory type	Non-Volatile	High Density	One-Transistor cell	In-system writability
SRAM	No	No	No	Yes
DRAM	No	Yes	Yes	Yes
ROM	Yes	Yes	Yes	No
EPROM	Yes	Yes	Yes	No
EEPROM	Yes	No	No	Yes

5.27 IMPLEMENTATION OF COMBINATIONAL LOGIC CIRCUITS USING ROM

Example 5.3: Draw 4×2 ROM with AND-OR gates.

Solution

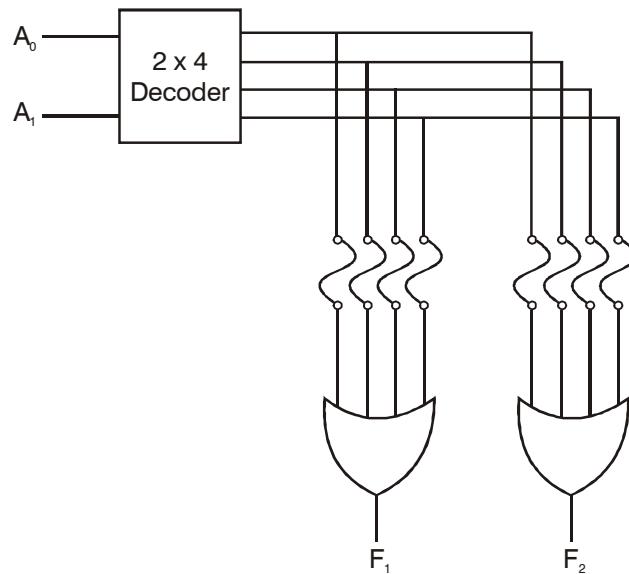


Fig. 5.35 : 4×2 ROM

Example 5.4: Draw the logic construction of 32×4 ROM.

Solution

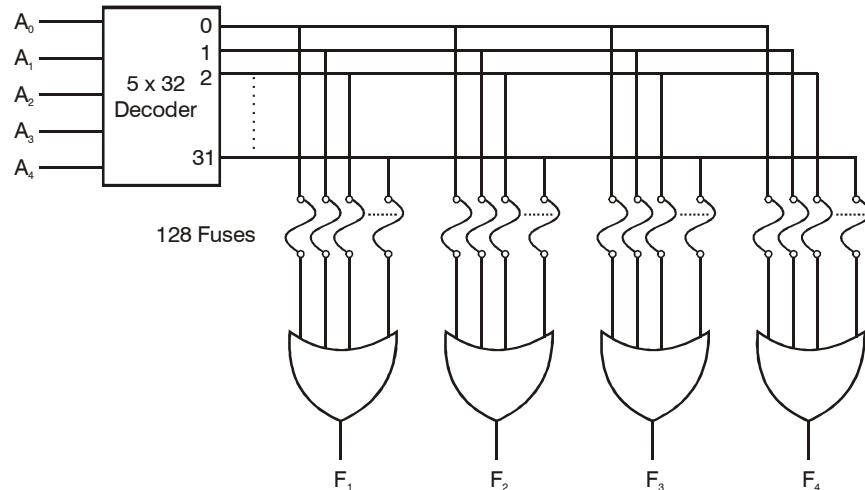


Fig. 5.36 : 32×4 ROM

Example 5.5: Implement the following Boolean functions using ROM

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

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

Solution:

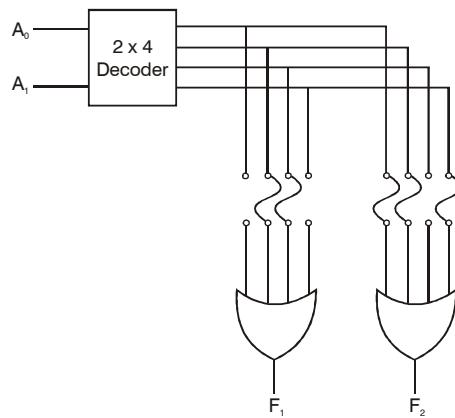


Fig. 5.37 : ROM circuit

Example 5.6: Design a combinational circuit for 3 bit binary to excess-3 code converter using ROM.

Solution: Truth Table for Binary to Excess 3 code converter.

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

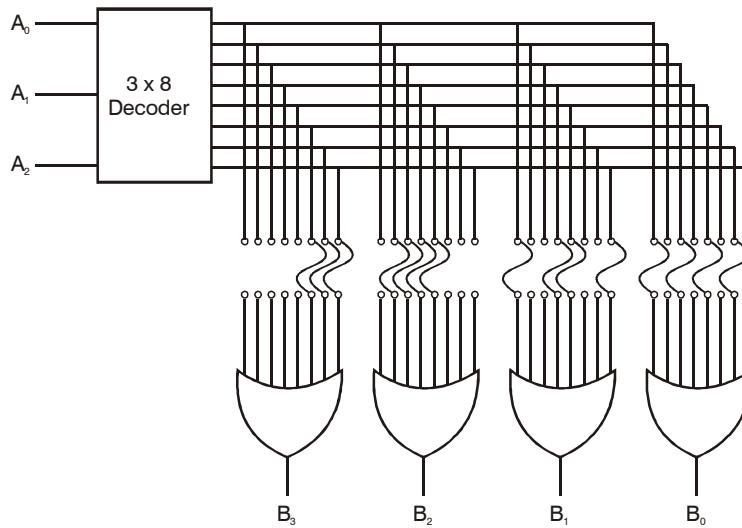


Fig. 5.38 : Binary to Excess 3 code converter

Example 5.7: Implement the following Boolean function using ROM (PROM)

$$F_1 = \sum m(1, 2, 3); F_2 = \sum m(0, 1, 3).$$

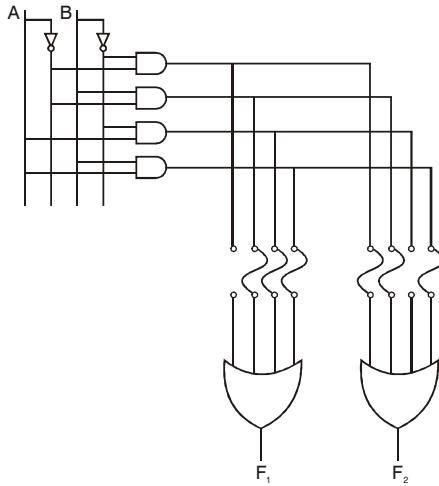


Fig. 5.39 : PROM circuit

5.28 PROGRAMMABLE LOGIC DEVICES

A programmable logic device (PLD) is an integrated circuit with internal logic gates that are connected through electronic fuses. Programming the device involves the blowing of internal fuses to achieve a desired logic function. The initial state of PLD has all the fuses intact as shown in **Figure 5.56(a)**.

All PLD consist of programmable arrays. A programmable array is essentially a grid of conductors that form rows and columns with a fusible link at each cross point. Arrays can be either fixed or programmable. The gates in a PLD are divided into an AND array and an OR array that are connected together to provide an AND-OR implementation.

Figure 5.56(b) shows an OR array consists of OR gates connected to a programmable matrix with fusible links at each cross point of a row and column. The array can be programmed by blowing fuses to eliminate selected variables from the output functions as shown in **Figure 5.56(b)**.

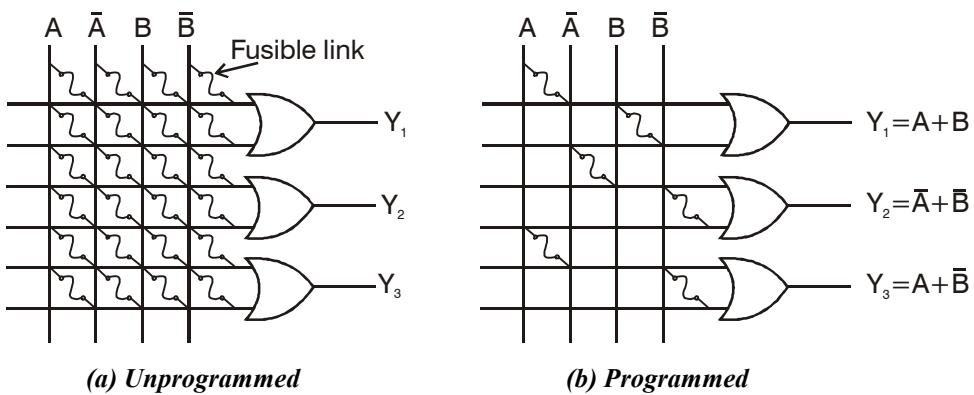


Fig. 5.40 : Programmable OR array

Figure 5.57(a) shows an AND array consists of AND gates connected to a programmable matrix with fusible links at each cross point of a row and column. The array can be programmed by blowing fuses to eliminate variables from the output function as shown in **Figure 5.41(b)**.

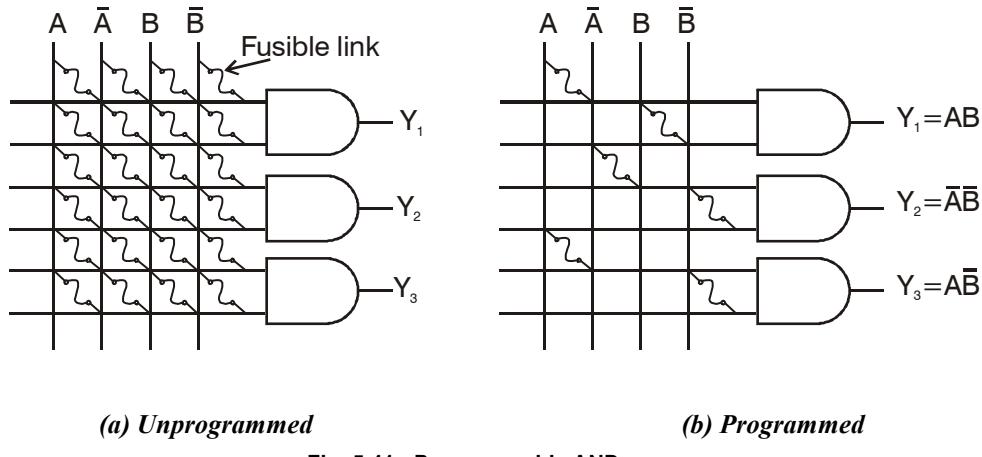


Fig. 5.41 : Programmable AND array

5.29 CLASSIFICATION OF PLDs

The classification of PLDs are as follows:

1. Simple Programmable Logic Devices (SPLD)
 - (i) Programmable ROM (PROM)
 - (ii) Programmable Logic Array (PLA)
 - (iii) Programmable Array Logic (PAL)
 - (iv) Generic Array Logic (GAL)
2. Complex Programmable Logic Devices (CPLD)
3. Field Programmable Gate Array (FPGA).

5.29.1 PROM

The PROM has a fixed (non-programmable) AND array and programmable fuses for the output OR gates. The PROM implements Boolean function in sum of minterms.

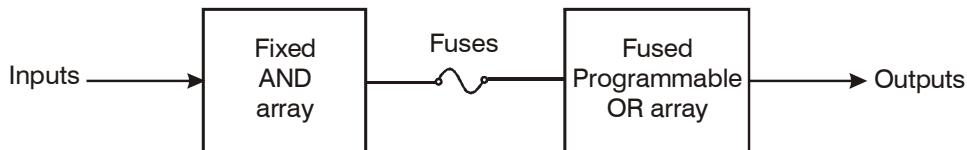


Fig. 5.42 : PROM

5.29.2 Programmable Array Logic (PAL)

The PAL has a fused programmable AND array and a fixed OR array as shown in **Figure 5.43**. The AND gates are programmed to provide the product terms for the Boolean functions that are logically summed in each OR gate.

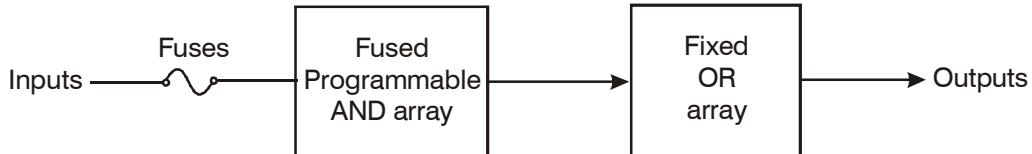


Fig. 5.43 : PAL

5.29.3 Programmable Logic Array (PLA)

The PLA has a programmable AND array and a programmable OR array as shown in **Figure 5.44**. The product terms in the AND array may be shared by any OR gate to provide the required sum of products implementation. The PLA may be mask programmable or field programmable.

(i) Mask Programmable Logic Array

With a mask PLA, the customer must submit a PLA program table to the manufacturer. This table is used by the vendor to produce a custom-made PLA that has the required internal paths between inputs and outputs.

(ii) Field Programmable Logic Array (FPLA)

The FPLA can be programmed by the user by means of certain recommended procedures. Commercial hardware programmer units are available for use in conjunction with certain FPLAs.

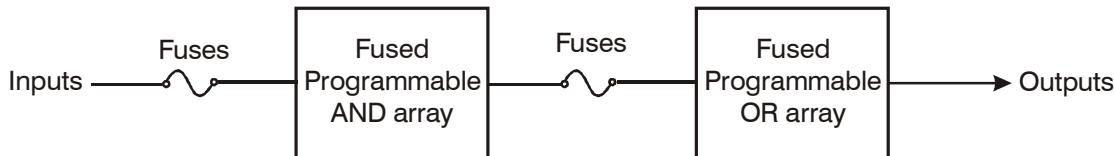


Fig. 5.44: PLA

5.29.4 Field Programmable Gate Array (FPGA)

The FPGA consists of an array of logic blocks with programmable row and column interconnecting channels surrounded by programmable I/O blocks.

Types of PLDs

<i>Device</i>	<i>AND-array</i>	<i>OR-array</i>
PROM	Fixed	Programmable
PLA	Programmable	Programmable
PAL	Programmable	Fixed

PLDs have hundreds of gates interconnected through hundreds of electronic fuses. It is sometimes convenient to draw the internal logic of such devices in a compact form referred to as **Array Logic**.

Figure 5.45 shows the convenient and array logic symbols for multiple-input AND gate and NOT gate.

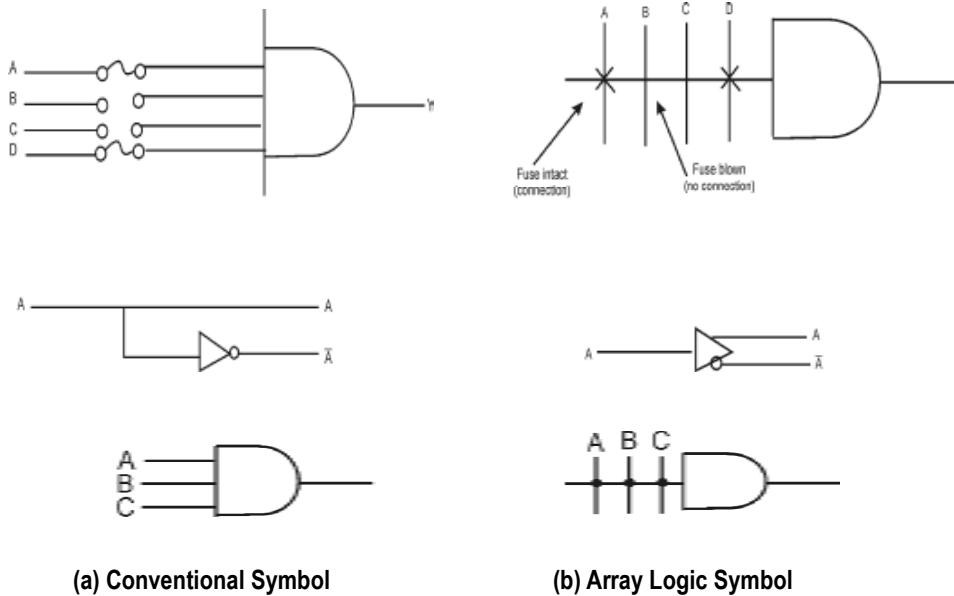


Fig. 5.45 : Array Logic Symbols

5.30 PROGRAMMABLE ROM (PROM)

PROMs are used for code conversions, generating bit patterns for characters and as look-up tables for arithmetic functions.

As a PLD, PROM consists of a fixed AND-array and a programmable OR array. The AND array is really an n -to- 2^n decoder and the OR array is simply a collection of programmable OR gates. The OR array is also called the memory array. The decoder serves as a minterm generator. The n -variable minterms appear on the 2^n lines at the decoder output. The 2^n outputs are connected to each of the ' m ' gates in the OR array via programmable fusible links. **Figure 5.46** illustrates a $2^n \times m$ PROM.

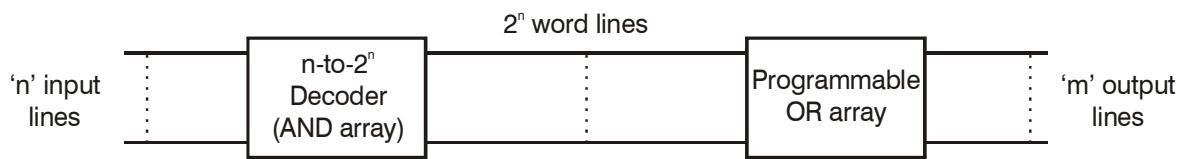


Fig. 5.46 : $2^n \times m$ PROM

Example 5.8: Design a logic circuit for the following Boolean expressions using (PROM).

$$f_1(x, y, z) = \sum m(0, 1, 2, 5, 7)$$

$$f_2(x, y, z) = \sum m(1, 2, 4, 6)$$

Solution:

Truth Table

x	y	z	f_1	f_2
0	0	0	1	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	0	1
1	1	1	1	0

PROM Logic Design

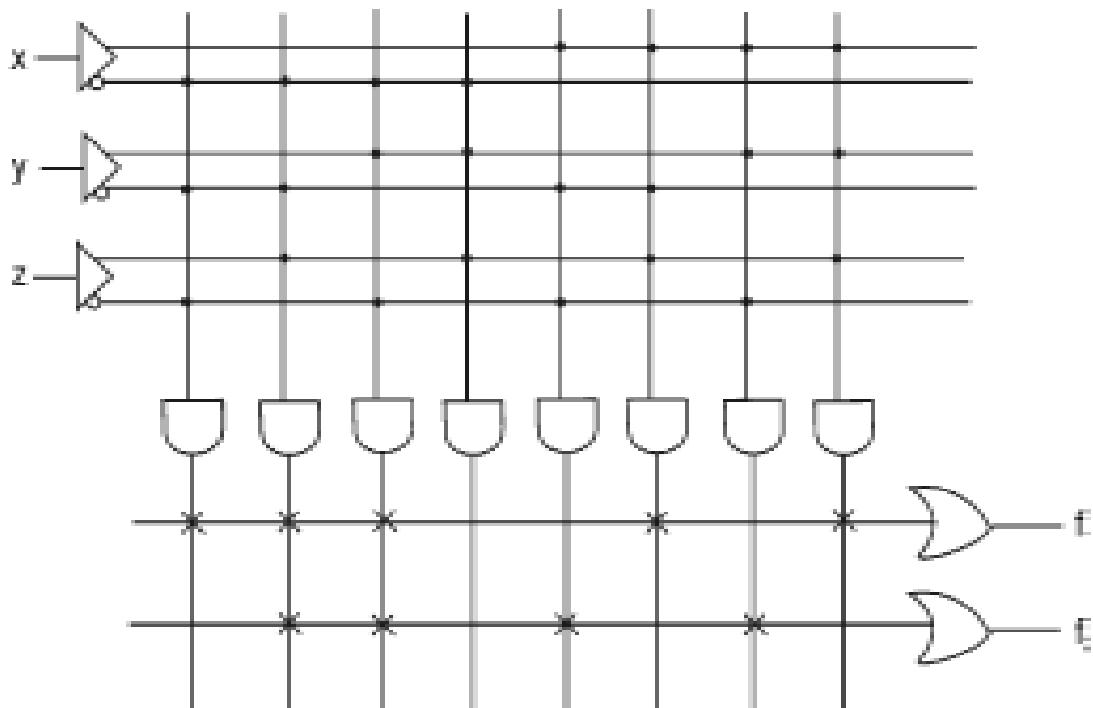


Fig. 5.47 : PROM circuit

5.31 PROGRAMMABLE LOGIC ARRAY

PLA is used in logic design where the number of don't care conditions is excessive, since it is more economical. In the PLA, the decoder (in ROM) is replaced by a group of AND gates, each of which can be programmed to generate a product term of the input variables. The AND and OR gates inside the PLA are initially fabricated with fuses among them. The specific Boolean functions are implemented in sum of products form by blowing appropriate fuses and leaving the desired connections.

The block diagram of the PLA is shown in **Figure 5.48**. It consists of 'n' inputs, 'm' outputs, 'k' product terms and 'm' sum terms. The product terms constitute a group of 'k' AND gates and the sum terms constitute a group of 'm' OR gates. Fuses are inserted between all 'n' inputs and their complement values to each of the AND gates. Fuses are also provided between the outputs of the AND gates and the inputs of the OR gates. Another set of fuses in the output inverters allow the output function to be generated either in the AND-OR form or in the AND-OR-INVERT form. With the inverter fuse in place, the inverter is bypassed, giving an AND-OR implementation. With the fuse blown, the inverter becomes part of the circuit and the function is implemented in the AND-OR-INVERT form.

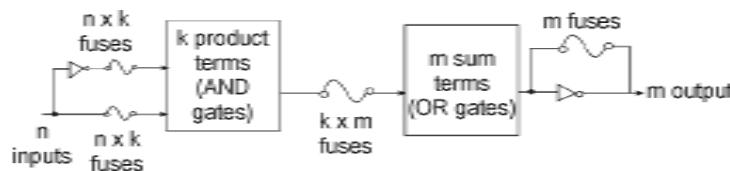


Fig. 5.48 : PLA block diagram

This PLA design is explained with the following example.

Example 5.9: Implement the combinational circuit for the functions

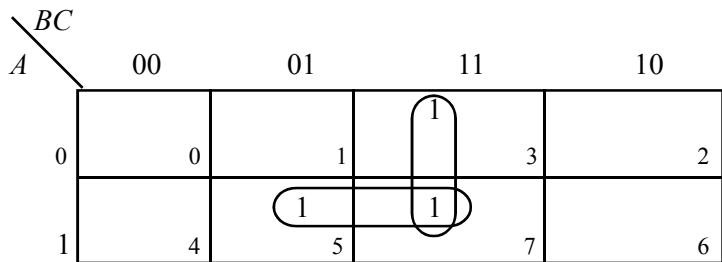
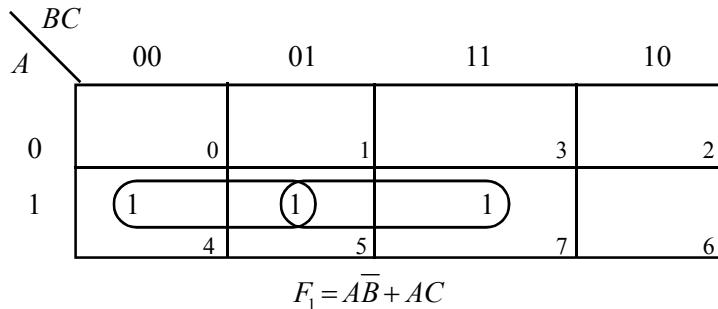
$$F_1 = \sum m(4, 5, 7); F_2 = \sum m(3, 5, 7)$$

Solution: The steps required in PLA implementation are:

- Step 1: Truth Table for given functions
- Step 2: K-map simplification
- Step 3: PLA program table
- Step 4: PLA diagram

Step 1: Truth Table:

A	B	C	F ₁	F ₂
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

Step 2: K map Simplification:**Step 3: PLA Program Table:**

Product Term	Inputs			Outputs	
	A	B	C	F_1	F_2
$A\bar{B}$	1	1	0	—	1
AC	2	1	—	1	1
BC	3	—	1	1	—
				T	T
				T/C	

In the PLA program table, first column lists the product terms numerically as 1, 2, 3. The second column (Inputs) specifies the required paths between AND gates and inputs. For each product term, the inputs are marked with 1, 0 or – (dash). If a variable in the product term appears in its normal form (unprimed), the corresponding input variable is marked with a 1. If it appears complemented (primed), the corresponding input variable is marked with a 0. If the variable is absent in the product term, it is marked with a dash.

The third column (outputs) specifies the paths between the AND gates and the OR gates. The output variables are marked with 1's for all those product terms that formulate the function. For example,

$$F_1 = A\bar{B} + AC$$

So F_1 is marked with 1's for product terms 1 and 2 and with a dash for product term 3. Each product term that has a 1 in the output column requires a path from the corresponding AND gate to the output OR gate. Those marked with a dash specify no connection.

Under each output variable, write T or C. The T (true) specifies that the fuse across the output inverter remains intact and a C (complement) specifies that the corresponding fuse be blown. The PLA circuit is shown in **Figure 5.49**.

Inputs	(A, B, C)	$n = 2$
Product terms	$(1, 2, 3)$	$k = 3$
Outputs	(F_1, F_2)	$m = 2$

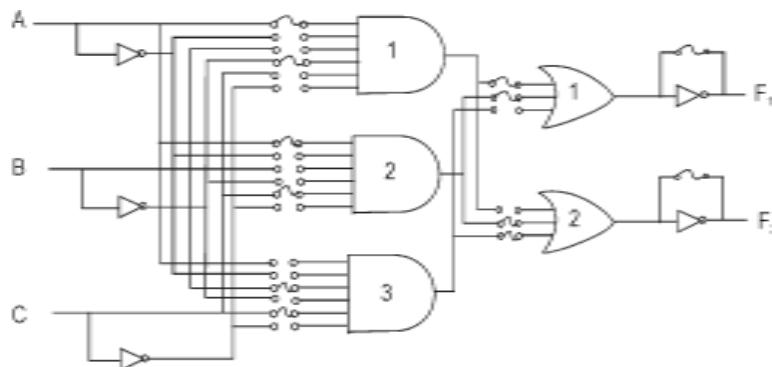


Fig. 5.49 : PLA circuit

5.32 IMPLEMENTATION OF COMBINATIONAL LOGIC CIRCUIT USING PLA

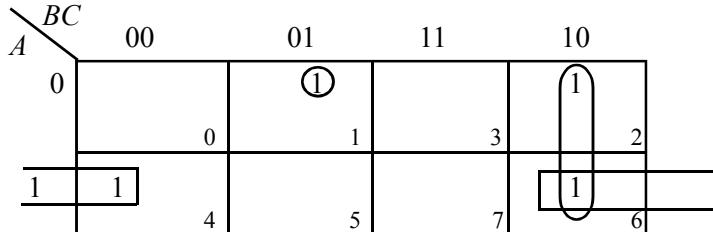
Example 5.10: Implement the following functions using PLA. (Dec. 2005)

$$F_1 = \sum m (1, 2, 4, 6); F_2 = \sum m (0, 1, 6, 7)$$

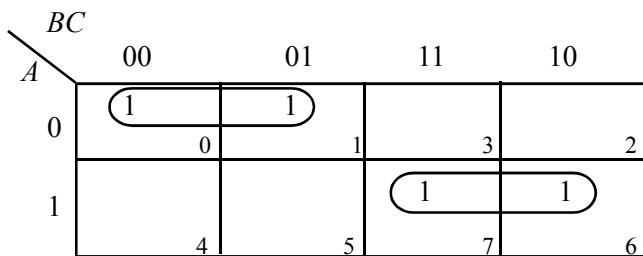
$$F_3 = \sum m (2, 6)$$

Solution: Step 1: Truth Table for given functions:

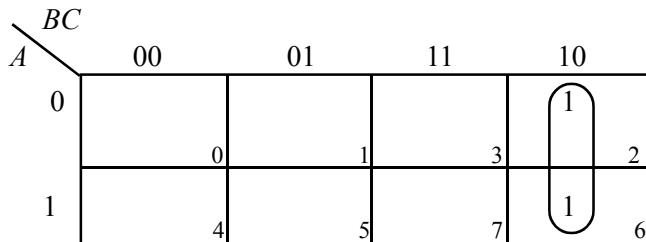
A	B	C	F_1	F_2	F_3
0	0	0	0	1	0
0	0	1	1	1	0
0	1	0	1	0	1
0	1	1	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	1	0

Step 2: K map Simplification:

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



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



$$F_3 = B\overline{C}$$

Step 3: PLA Program Table:

Product Term	Inputs			Outputs			
	A	B	C	F_1	F_2	F_3	
$\overline{A}\overline{B}C$	1	0	0	1	—	—	
$A\overline{C}$	2	1	—	0	1	—	
$B\overline{C}$	3	—	1	0	1	—	1
$\overline{A}\overline{B}$	4	0	0	—	—	1	—
AB	5	1	1	—	—	1	—
				T	T	T	T/C

Step 4: Draw the PLA circuit with

3 inputs
5 product terms
3 outputs

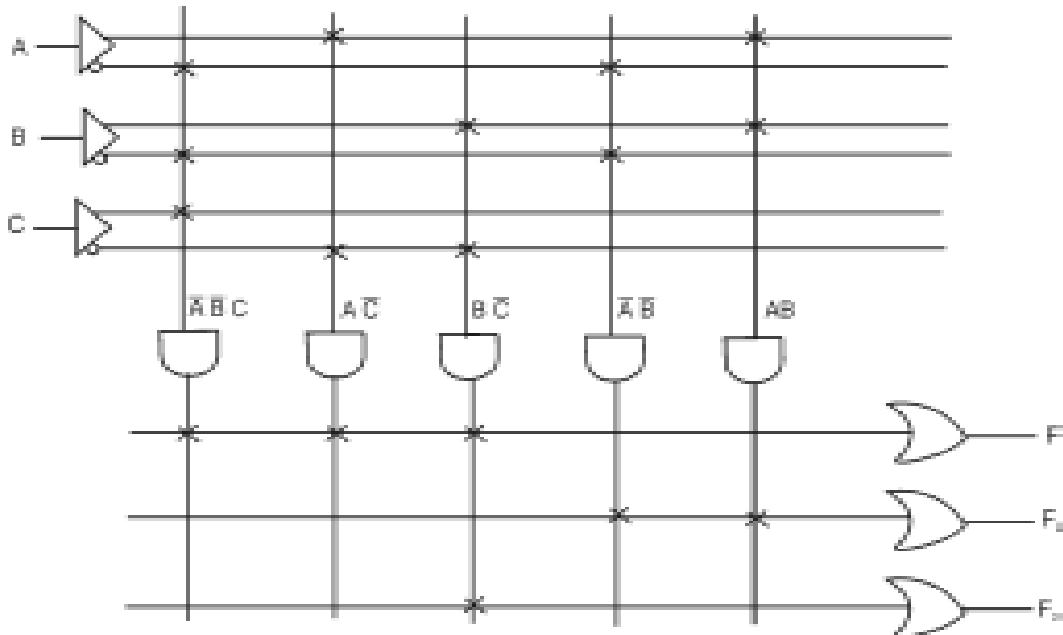


Fig. 5.50 : PLA circuit

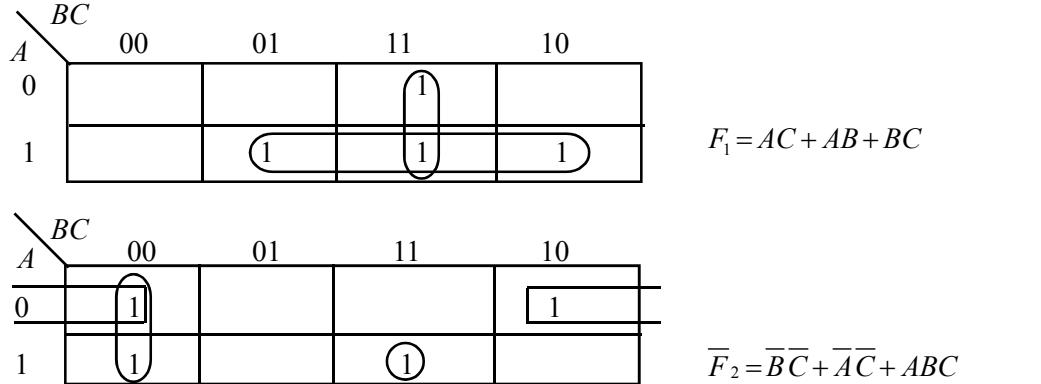
Example 5.11: Implement the combinational circuit with a PLA having 3 inputs, 4 product terms and 2 outputs for the functions:

$$F_1 = \sum (3, 5, 6, 7), F_2 = \sum (0, 2, 4, 7)$$

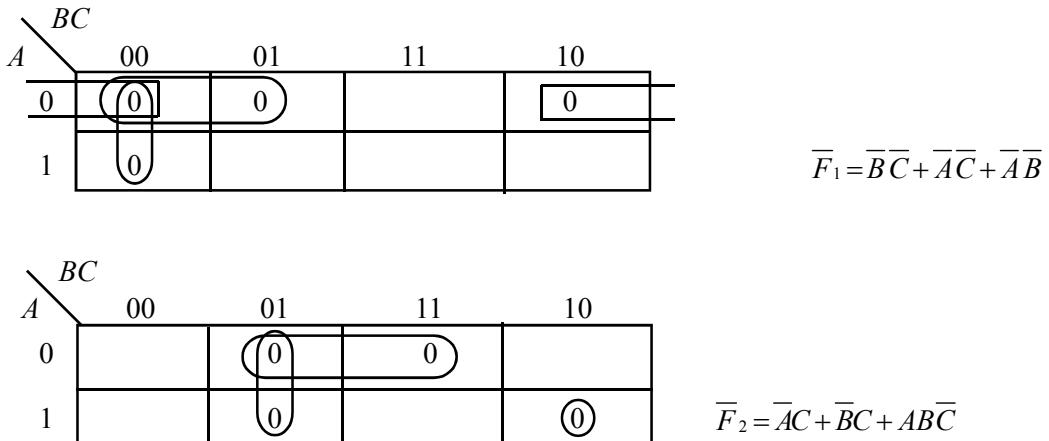
(Dec. 2005)

Solution: Step 1: Truth Table for Boolean Functions:

A	B	C	F_1	F_2
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Step 2: K map Simplification:

With this simplification, the total number of product terms is 6. But we require only 4 product terms. Therefore find out the K map simplification for \overline{F}_1 and \overline{F}_2 .



Now select the functions \overline{F}_1 and \overline{F}_2 , since the common product terms are $(\overline{BC}, \overline{AC}, \overline{AB}, ABC)$

Step 3: PLA Program Table:

Product Term	Inputs			Outputs	
	A	B	C	F_1	F_2
\overline{BC}	1	-	0	0	1
\overline{AC}	2	0	-	0	1
\overline{AB}	3	0	0	-	1
ABC	4	1	1	1	-
				C	T
					T/C

$$F_1 = (\overline{B}\overline{C} + \overline{A}\overline{C} + \overline{A}\overline{B}) ; F_2 = \overline{B}\overline{C} + \overline{A}\overline{C} + ABC$$

Step 4: Draw the PLA circuit with

3 inputs, 4 product terms and 2 outputs.

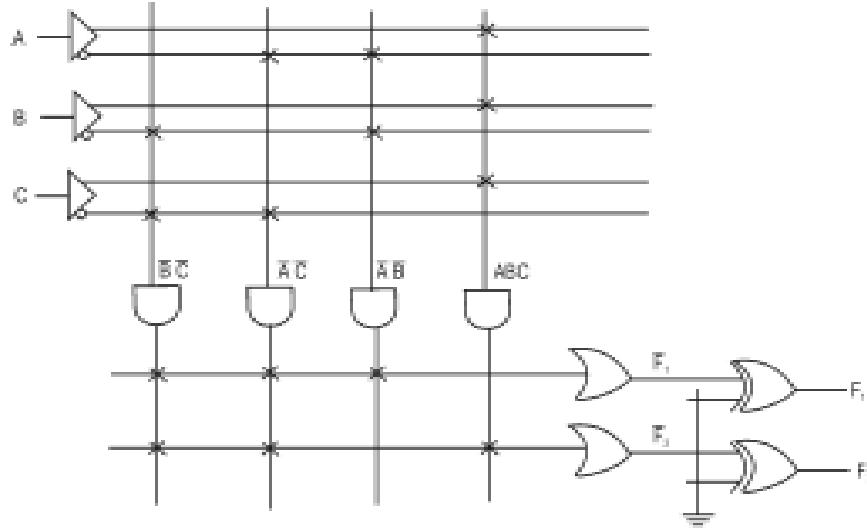


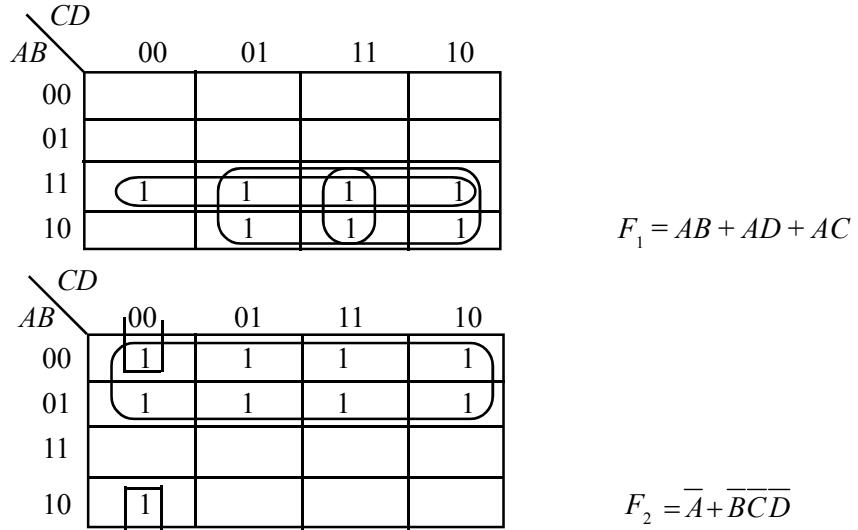
Fig. 5.51 : PLA circuit

It should be noted that $\overline{F_1}$ really occurs at one of the outputs of the OR-array. By programming the corresponding EX-OR gate fuse, $\overline{F_1} = F_1$ appears at the output of the PLA.

Example 5.12: A combinational logic circuit has 4 inputs and 2 outputs. The output 1 gives high output when the input combinational is greater than or equal to 1001 and the output 2 gives high output when the input combinational is less than 1001. Implement the circuit with PLA.

Solution: Step 1: Truth Table for given problem:

A	B	C	D	F_1	F_2
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	0

Step 2: K map Simplification:**Step 3: PLA Program Table:**

Product Term	Inputs				Outputs	
	A	B	C	D	F_1	F_2
AB	1	1	1	-	-	1
AC	2	1	-	1	-	1
AD	3	1	-	-	1	-
\bar{A}	4	0	-	-	-	1
$\bar{B}\bar{C}\bar{D}$	5	-	0	0	0	1
					T	T
						T/C

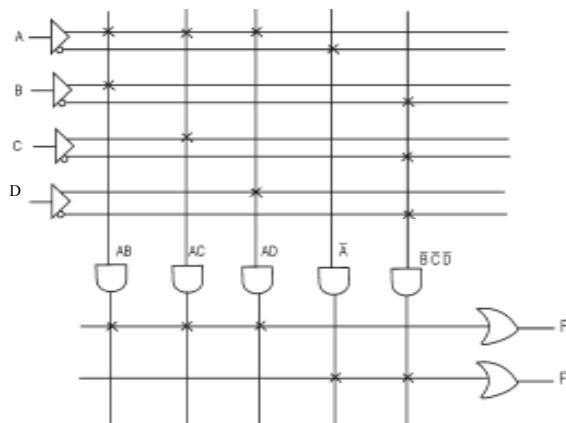
Step 4: Draw the PLA circuit with 4 inputs, 5 product terms and 2 outputs.

Fig. 5.52 : PLA circuit

Example 5.13: Design a BCD-to-Excess 3 code converter with PLA circuit.

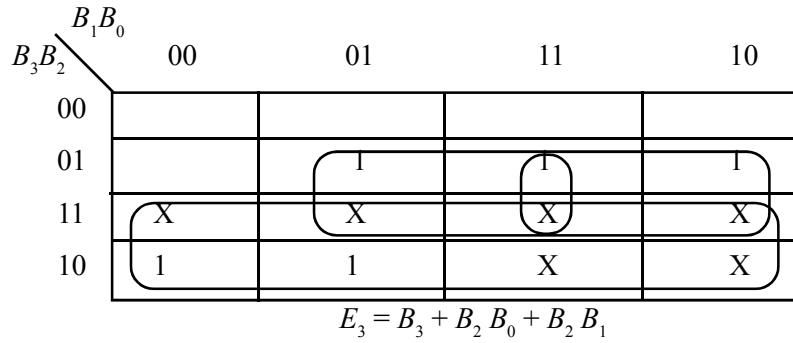
(May 2006)

Solution: Step 1: Truth Table for BCD to XS-3 code converter:

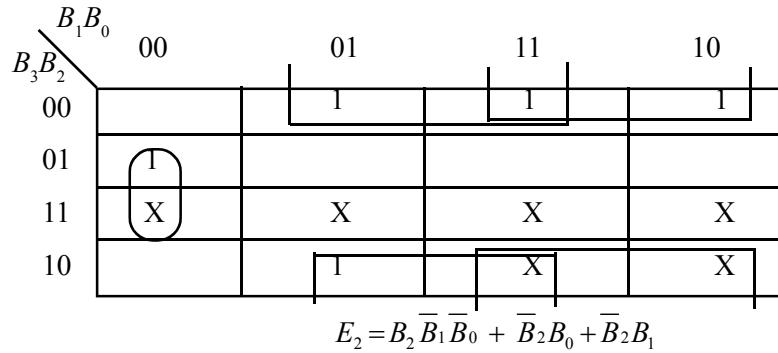
Decimal	BCD Code				Excess-3 Code			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Step 2: K map Simplification:

For E_3



For E_2



For E_1

		00	01	11	10
B_3B_2	B_1B_0	00	1		1
		01	1	1	
11	X	X	X	X	X
10	1		X		X

$$E_1 = \overline{B}_1 \overline{B}_0 + B_1 B_0$$

For E_0

		00	01	11	10
B_3B_2	B_1B_0	00	1		1
		01	1		1
11	X	X	X	X	X
10	1	X	X		X

$$E_0 = \overline{B}_0$$

Step 3: PLA Program Table

<i>Product terms</i>		<i>Inputs</i>				<i>Outputs</i>			
		B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
B_3	1	1	—	—		1	—	—	—
B_2B_0	2	—	1	—	1	1	—	—	—
B_2B_1	3	—	1	1	—	1	—	—	—
$B_2\overline{B}_1\overline{B}_0$	4	—	1	0	0	—	1	—	—
\overline{B}_2B_0	5	—	0	—	1	—	1	—	—
\overline{B}_2B_1	6	—	0	1	—	—	1	—	—

$\bar{B}_1 \bar{B}_0$	7	-	-	0	0	-	-	1	1
$\bar{B}_1 B_0$	8	-	-	1	1	-	-	1	-
B_0	9	-	-	1	0	-	-	-	1
						T	T	T	T/C

Step 4: Draw the PLA circuit with 4 inputs, 9 product terms and 4 outputs.

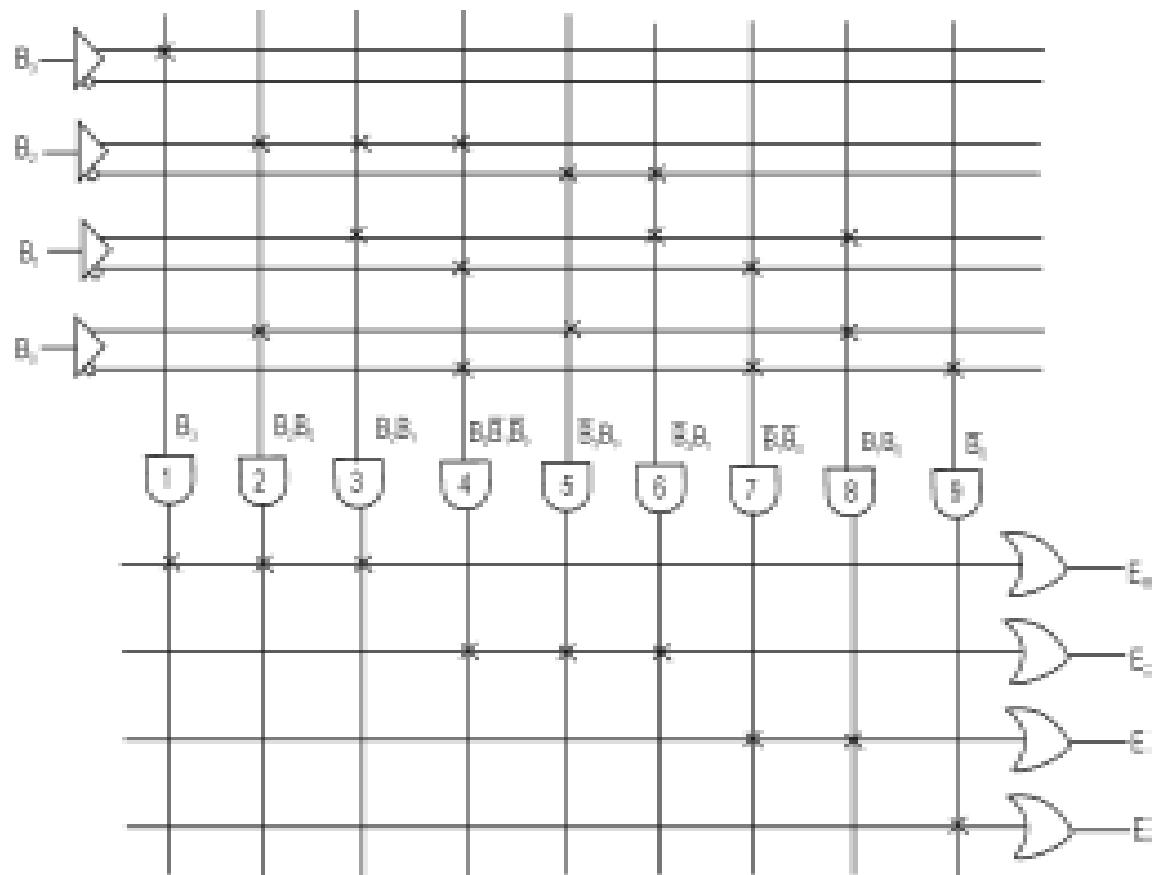


Fig. 5.53 : PLA circuit

Example 5.14: Implement the switching function.

(May 2013)

$$Z_1 = \overline{abde} + \overline{\overline{abcde}} + bc + de$$

$$Z_2 = \overline{a} \overline{c} e$$

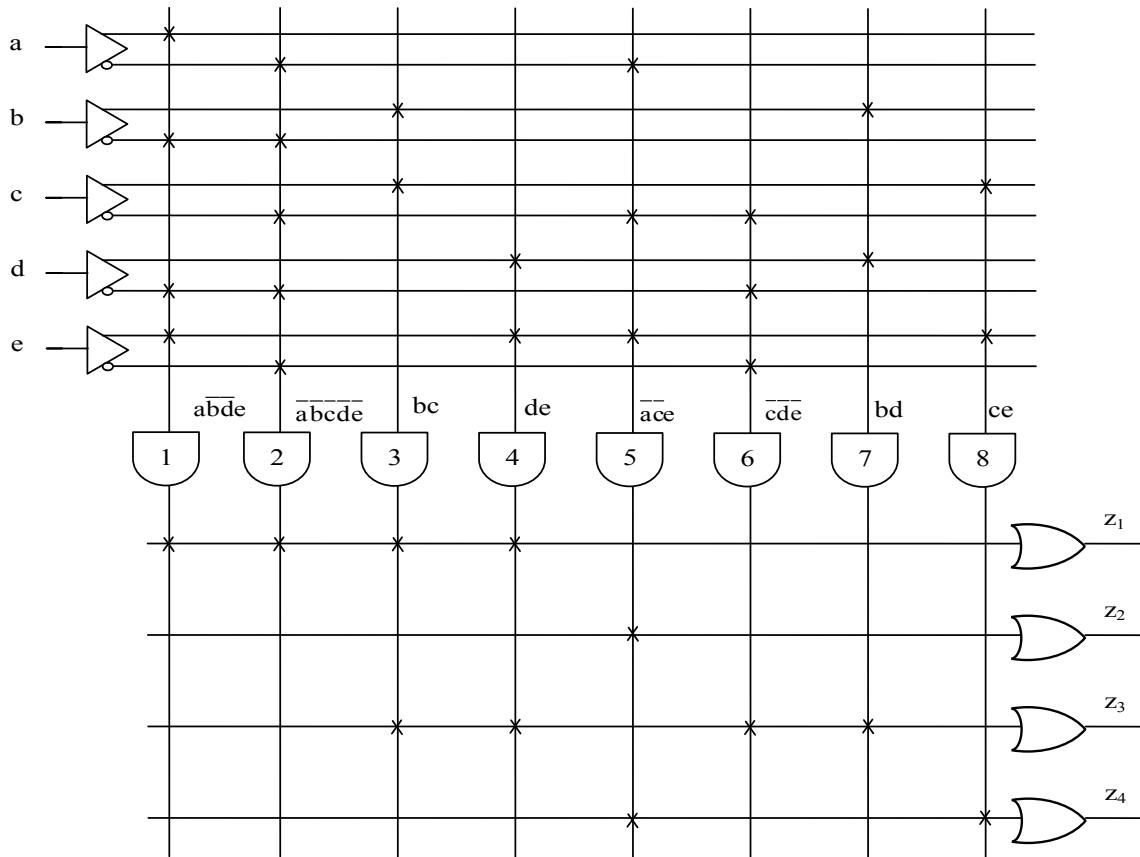
$$Z_3 = bc + de + \overline{\overline{cde}} + bd$$

$$Z_4 = \overline{ace} + ce \quad \text{using } 5 \times 8 \times 4 \text{ PLA}$$

Solution:

PLA Program Table

Product Terms	Inputs					Outputs			
	a	b	c	d	e	Z ₁	Z ₂	Z ₃	Z ₄
\overline{abde}	1	1	0	-	0	1	1	-	-
$\overline{\overline{abcde}}$	2	0	0	0	0	0	1	-	-
bc	3	-	1	1	-	-	1	-	1
de	4	-	-	-	1	1	1	-	1
\overline{ace}	5	0	-	0	-	1	-	1	-
$\overline{\overline{cde}}$	6	-	-	0	0	0	-	-	1
bd	7	-	1	-	1	-	-	-	1
ce	8	-	-	1	-	1	-	-	1



5.33 PROGRAMMABLE ARRAY LOGIC (PAL)

The PAL consists of a programmable array of AND gates that connects to a fixed array of OR gates. This structure allows any sum-of-products (SOP) logic expression with a defined number of variables to be implemented.

Example 5.15: Implement the circuit with PAL for the function $Y = AB + A\overline{B} + \overline{A}\overline{B}$.

Solution:

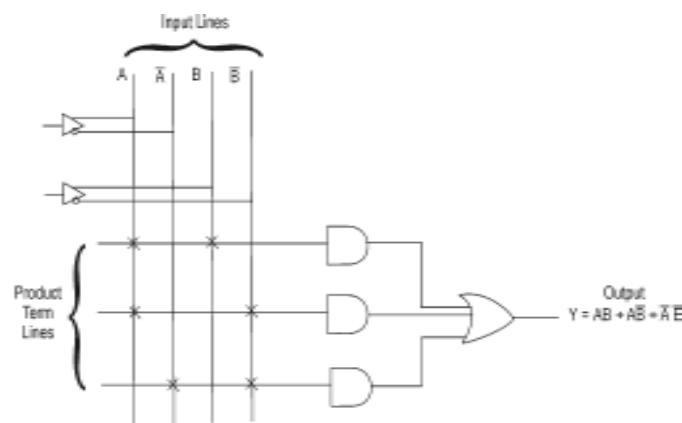


Fig. 5.54 : PAL circuit

5.34 IMPLEMENTATION OF COMBINATIONAL LOGIC CIRCUIT USING PAL

Example 5.16: Draw the PAL diagram for the following Boolean functions:

$$Y_3 = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}C\overline{D} + \overline{A}B\overline{C}\overline{D} + A\overline{B}C\overline{D}$$

$$Y_2 = \overline{A}\overline{B}C\overline{D} + \overline{A}B\overline{C}\overline{D} + A\overline{B}CD$$

$$Y_1 = \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + AC + A\overline{B}\overline{C}$$

$$Y_0 = ABCD$$

Solution:

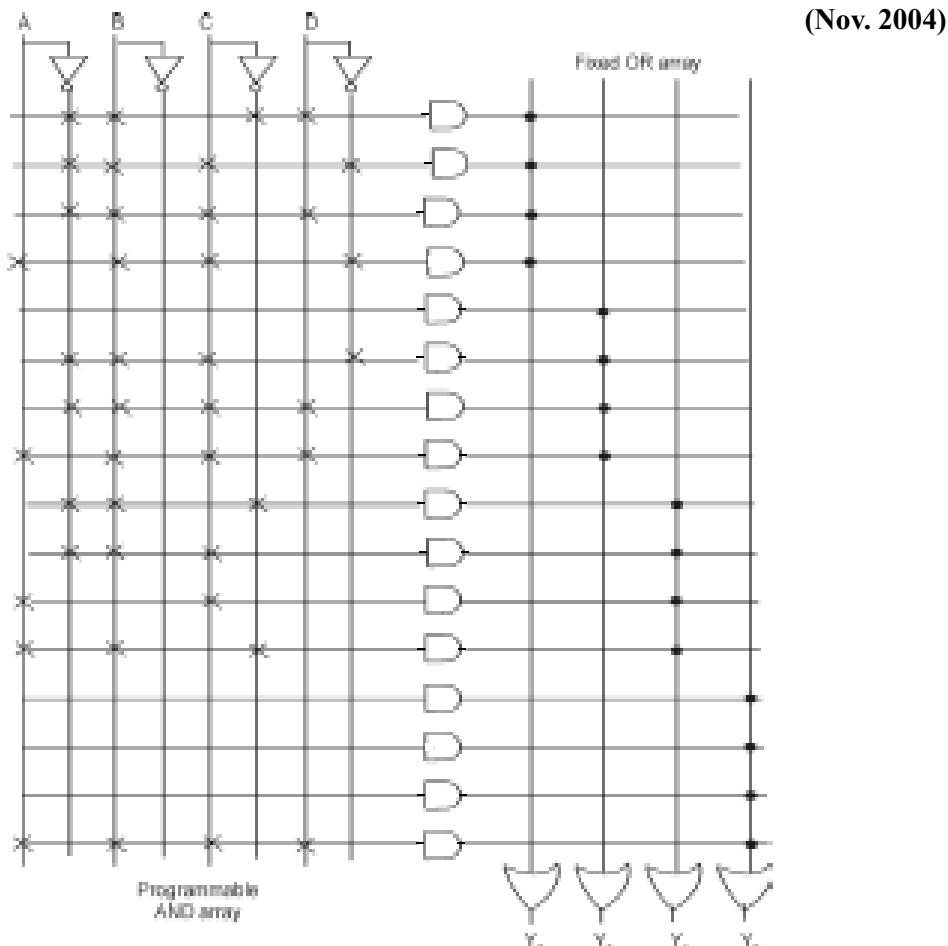


Fig. 5.55 : PAL circuit

Example 5.17: Implement the following Boolean functions using PAL.

$$W(A,B,C,D) = \sum(0,2,6,7,8,9,12,13)$$

$$X(A,B,C,D) = \sum(0,2,6,7,8,9,12,13,14)$$

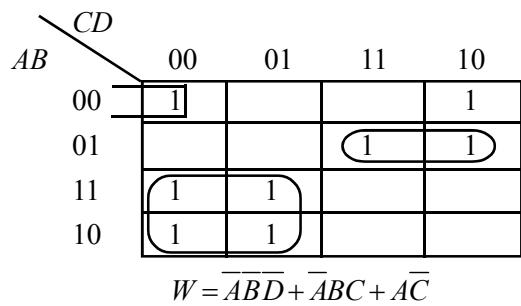
$$Y(A,B,C,D) = \sum(2,3,8,9,10,12,13)$$

$$Z(A,B,C,D) = \sum(1,3,4,6,9,12,14)$$

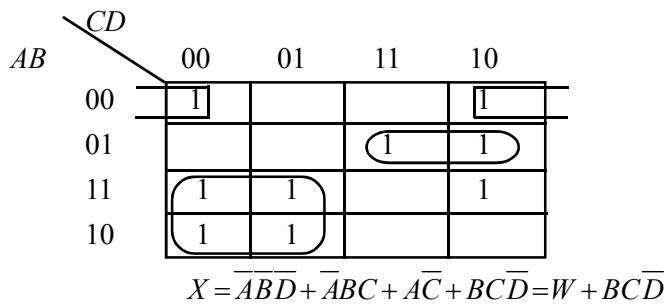
Solution:

1. Simplifying the given Boolean functions to a minimum number of terms using K-map.

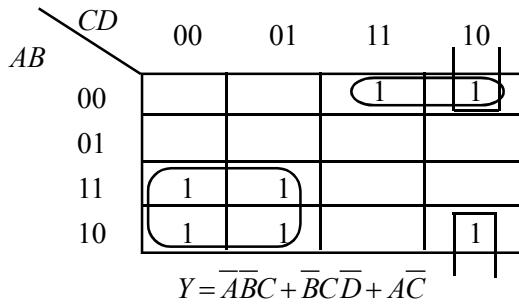
For W



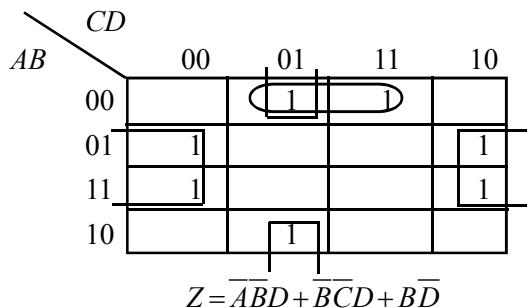
For X



For Y



For Z



II. PAL Program Table

Product Term		AND Inputs					Outputs
		A	B	C	D	W	
\overline{ABD}	1	0	0	-	0	-	W
\overline{ABC}	2	0	1	1	-	-	
$A\bar{C}$	3	1	-	0	-	-	
W	4	-	-	-	-	1	X
$BC\bar{D}$	5	-	1	1	0	-	
-	6	-	-	-	-	-	
\overline{ABC}	7	0	0	1	-	-	Y
$\overline{BC\bar{D}}$	8	-	0	1	0	-	
$A\bar{C}$	9	1	-	0	-	-	
$\overline{A\bar{B}D}$	10	0	0	-	1	-	Z
$\overline{BC\bar{D}}$	11	-	0	0	1	-	
$B\bar{D}$	12	-	1	-	0	-	

III. PAL Diagram

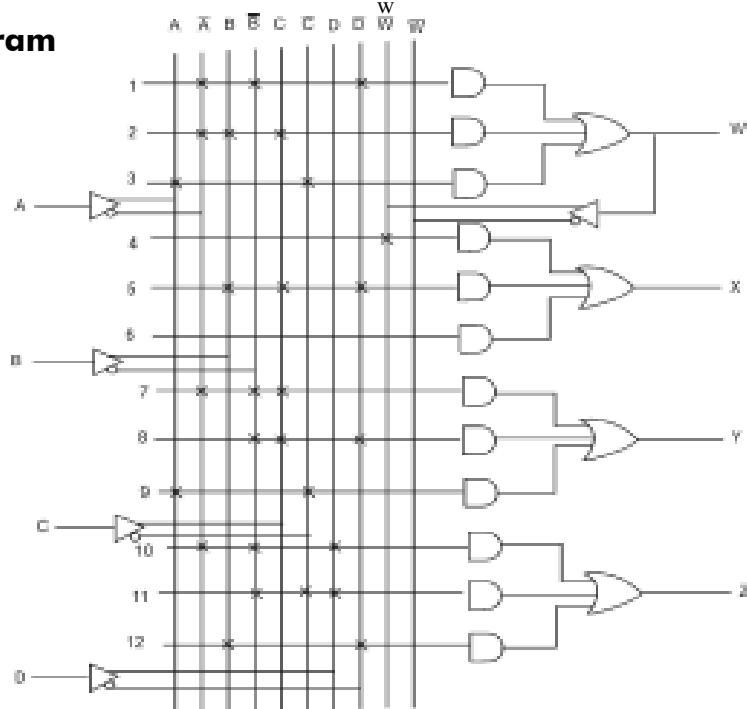


Fig. 5.56 : PAL circuit

Example 5.18: Implement the given function using PAL

$$A = \sum (0, 2, 6, 7, 8, 9, 12, 13)$$

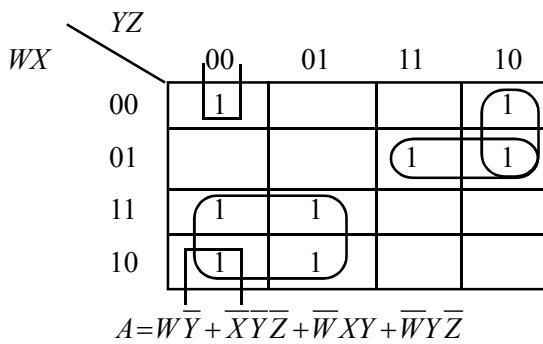
$$B = \sum (0, 2, 6, 7, 8, 9, 12, 13, 14)$$

$$C = \sum (1, 3, 4, 6, 10, 12, 13)$$

$$D = \sum (1, 3, 4, 6, 9, 12, 14)$$

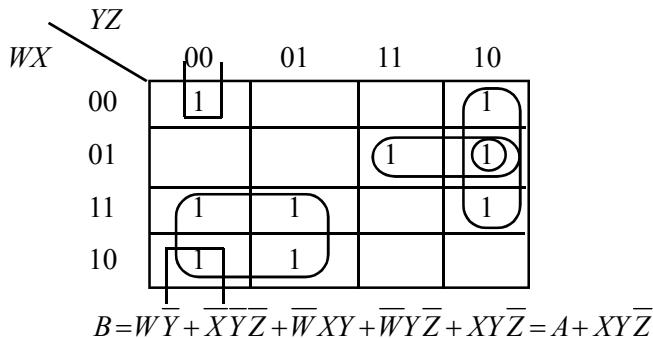
Solution: I. K map Simplification:

For A



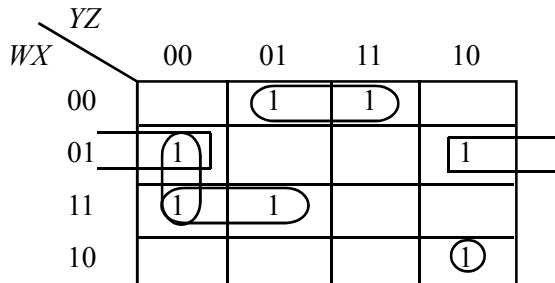
$$A = \bar{W}\bar{Y} + \bar{X}\bar{Y}\bar{Z} + \bar{W}XY + \bar{W}Y\bar{Z}$$

For B

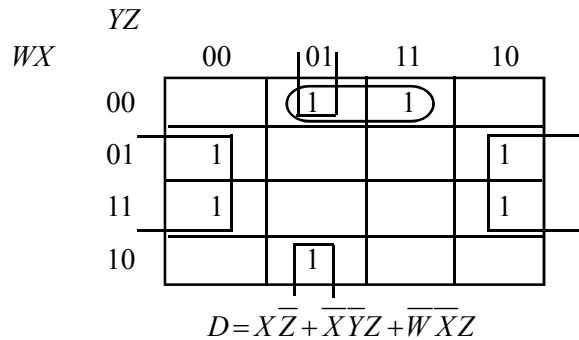


$$B = \bar{W}\bar{Y} + \bar{X}YZ + \bar{W}XY + \bar{W}Y\bar{Z} + XY\bar{Z} = A + XYZ$$

For C



$$C = X\bar{Y}\bar{Z} + WX\bar{Y} + \bar{W}X\bar{Z} + \bar{W}X\bar{Z} + \bar{W}\bar{X}Z + W\bar{X}Y\bar{Z}$$

For D**II. PAL Program Table:**

Product Term	Inputs					Outputs
	W	X	Y	Z	A	
$W\bar{Y}$	1	1	-	0	-	-
$\bar{X}YZ$	2	-	0	0	0	-
$\bar{W}XY$	3	0	1	1	-	-
$\bar{W}Y\bar{Z}$	4	0	-	1	0	-
A	5	-	-	-	-	1
$XY\bar{Z}$	6	-	1	1	0	-
$X\bar{Y}\bar{Z}$	7	-	1	0	0	-
$WX\bar{Y}$	8	1	1	0	-	-
$WX\bar{Z}$	9	0	1	-	0	-
$\bar{W}\bar{X}Z$	10	0	0	-	1	-
$W\bar{X}Y\bar{Z}$	11	1	0	1	0	-
$\bar{X}\bar{Z}$	12	-	1	-	0	-
$\bar{X}\bar{Y}\bar{Z}$	13	-	0	0	1	-
$\bar{W}\bar{X}Z$	14	0	0	-	1	-

III. PAL Diagram

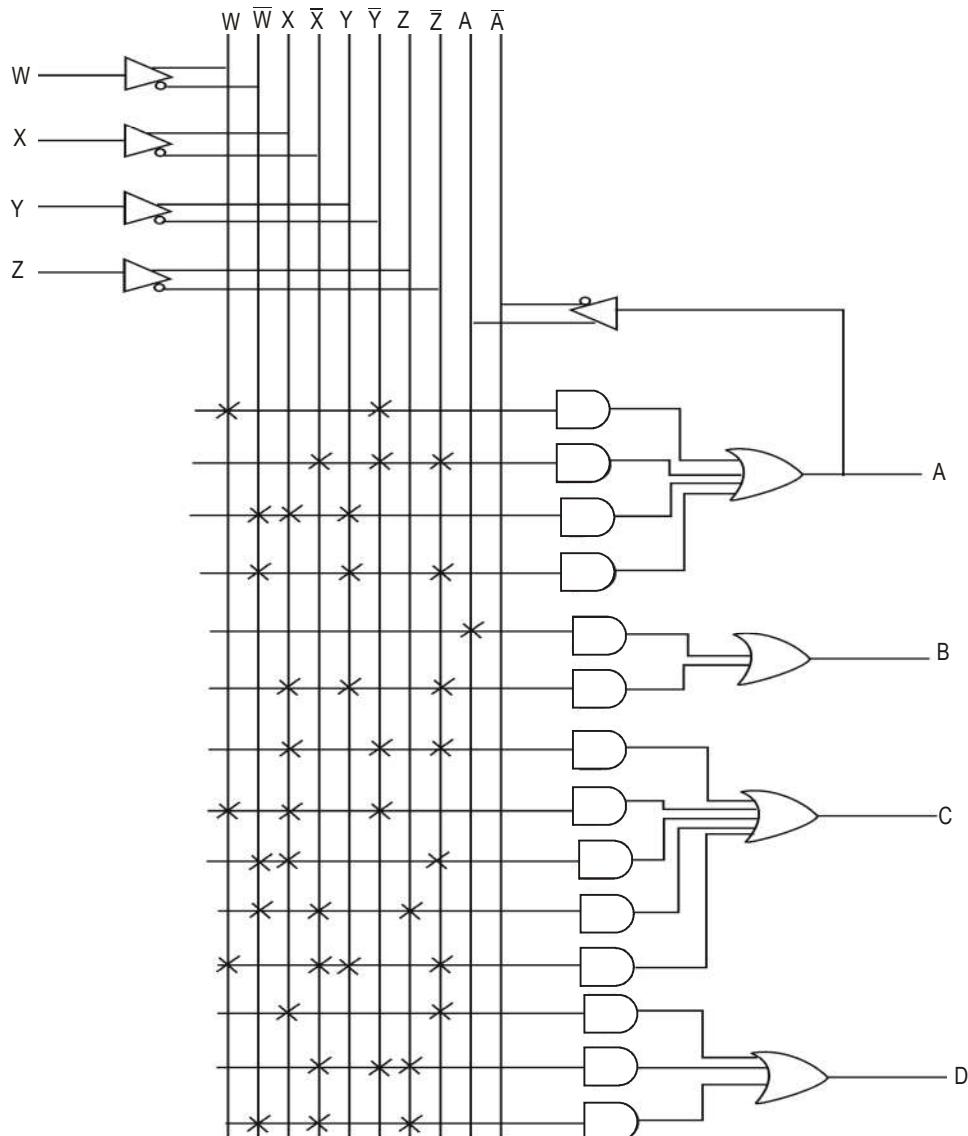


Fig. 5.57 : PAL circuit

5.35 FIELD PROGRAMMABLE GATE ARRAYS (FPGA)

The FPGA basically consists of an array of logic blocks with programmable row and column interconnecting channels surrounded by programmable I/O blocks as shown in **Figure 5.58**. Many FPGA architectures are based on a type of memory called LUT (look-up table) rather than on AND/OR arrays. Another approach is the use of multiplexers to generate logic functions. FPGA is a single VLSI (Very Large Scale Integrated) circuit constructed on a single piece of silicon.

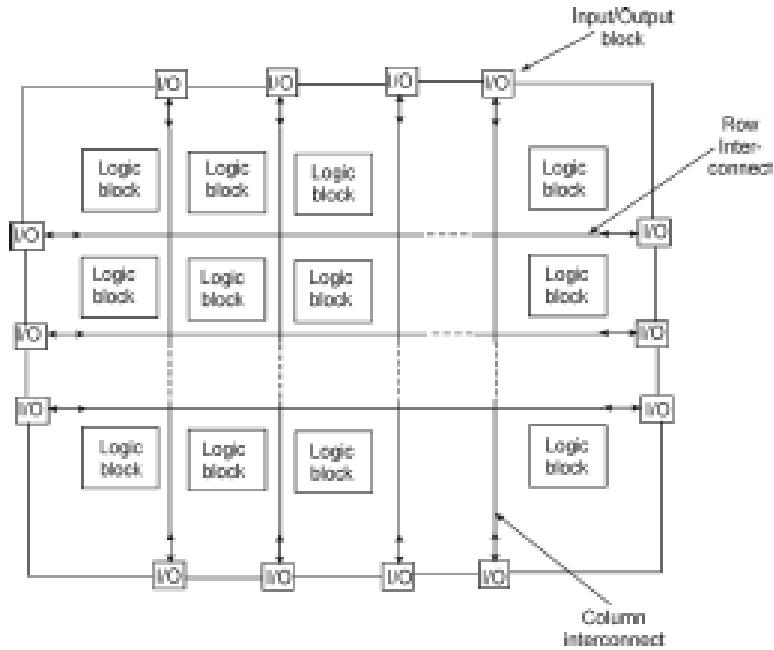


Fig. 5.58 : Basic Block Diagram of an FPGA

The module shown in **Figure 5.59** consists of 3 number of interconnected 2-to-1 multiplexer and an OR gate whose output selects the MUX 3.

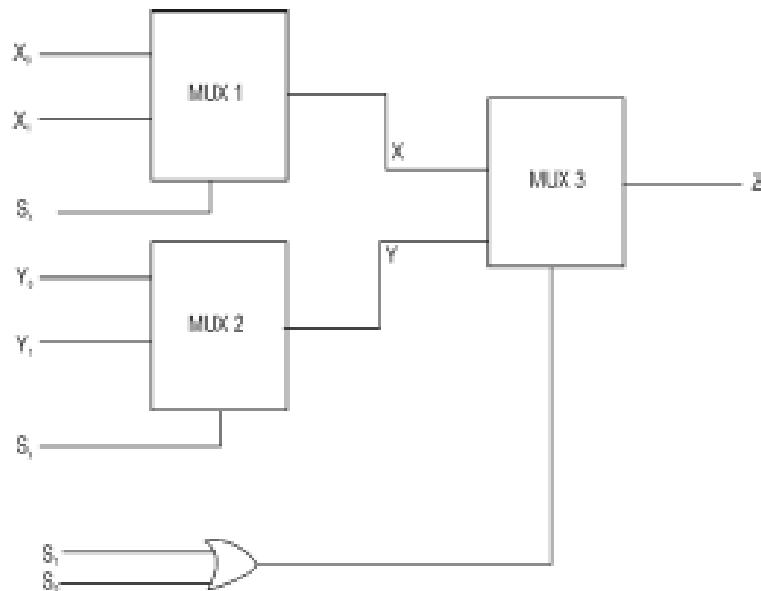


Fig. 5.59 : FPGA module

$$\text{Logic expression for MUX 1, } X = X_0 \bar{S}_X + X_1 S_X$$

$$\text{Logic expression for MUX 2, } Y = Y_0 \bar{S}_Y + Y_1 S_Y$$

By connecting the input of this module in different ways, the module can be programmed as a variety of 2 or 3 or 4 input gates. For example, by connecting X_0 , X_1 , Y_0 and S_1 to 0, the expression for output Z becomes $Z = Y_1 S_Y S_0$ and now this module acts like a 3 input positive logic AND gate. Similarly by connecting different inputs to high and low, the module acts like different gates. The logic circuit design procedure using FPGA involves the following steps:

1. Capture the logic circuit to be implemented with a suitable software package, using a library of logic elements which are various configuration of basic modules available in FPGA. In addition, many FPGA libraries also contain predesigned circuits for multiplexers, encoders, adders and so on. Predesigned circuits make design much easier.
2. Functional simulation simulates the circuit to determine whether it is function properly.
3. Configure and interconnect the modules of the FPGA to produce the desired logic circuit. This may be done automatically by a routing software called routes. Once the routing is over, it is now possible to determine the actual circuit delays which can now be introduced into the simulation model. Now an accurate simulation of the circuit can be available.
4. Programming is done by the FPGA interconnections. The routing of the devices determined in the previous step is now made into a fuse map. Then this fuse map is used in conjunction with a device programmer to make the internal device connections.
5. After programming, it must be tested. If the designed function is not fulfilled, it must be reprogrammed. With careful simulation, reprogramming can be minimized.

5.36 COMPARISON BETWEEN PROM, PLA AND PAL

Sl.No.	PROM	PLA	PAL
1.	Fixed AND array Programmable OR arrayProgrammable AND and Programmable OR array	Programmable AND and Programmable OR array	Programmable AND and Programmable OR array
2.	A11 minterms are decoded	Fixed OR array AND array can be programmed to get desired minterms.	AND array can be programmed to get desired minterms.
3.	Only Boolean functions in standard SOP form can be implemented.	Any Boolean functions in SOP form can be implemented	Any Boolean functions in SOP form can be implemented
4.	Cheaper	Costlier than PAL and PROM	Cheaper
5.	Simple to use	Complex than PAL and PROM	Simple to use

5.37 ERROR DETECTION AND CORRECTION

Data that is either transmitted over communication channel (bus) or stored in memory is not completely *error free*. Error can be caused by transmission errors and storage errors. Transmission error is caused by signal distortion or attenuation. Storage errors means DRAM memory cell contents can change spuriously due to some electromagnetic interference. In magnetic storage devices such as disks, magnetic flux density increases could cause one or more bits to flip (change that value).

- Error *detection* is the ability to detect errors
- Error *correction* has an additional feature that enables identification and correction of the errors
- Error detection always precedes error correction
- Both can be achieved by having *extra/redundant/check* bits in addition to data to deduce that there is an error
- Original Data is encoded with the redundant bit(s)
- New data formed is known as *code word*

Parity is the simplest and oldest error detection method. A binary digit called *parity* is used to indicate whether the number of bits with “1” in a given set of bits is even or odd. The parity bit is then appended to original data. *Sender* adds the parity bit to existing data bits before transmission. *Receiver* checks for the expected parity, If wrong parity found, the received data is discarded and retransmission is requested.

An error-correcting code uses multiple parity check bits that are stored with the data word in memory. Each check bit is a parity bit for a group of bits in the data word. When the word is read back from memory, the parity of each group, including the check bit, is evaluated. If the parity is correct for all groups, it signifies that no detectable error has occurred. If one or more of the newly generated parity values is incorrect, a unique pattern called a syndrome results that may be able to identify which bit is in error. A single error occurs when a bit changes in value from 1 to 0 or from 0 to 1 while stored or if it erroneously changes during a write or read operation. If the specific bit in error is identified, then the error can be corrected by complementing the erroneous bit.

5.37.1 Hamming Code

The most common types of error-correcting codes used in RAM are based on the codes devised by R. W. Hamming.

Hamming Code is used for error detection and error correction. This code uses a number of parity bits (dependent on the number of information bits), located at certain positions in the code group. If the number of information bit is designated 'M', then the number of parity bits 'P' is determinated as,

$$2^P \geq M + P + 1$$

For example, if we have 4 information bits, then P is found by trial and error method.

Let $P = 2$, then $2^P = 2^2 = 4$ and $M + P + 1 = 4 + 2 + 1 = 7$

Then equation $2^P \geq M + P + 1$ is not satisfied.

Let $P = 3$, then $2^P = 2^3 = 8$ and $M + P + 1 = 4 + 3 + 1 = 8$

The equation $2^P \geq M + P + 1$ is satisfied ($8 = 8$). So 3 bits are required to provide single error correction for four information bits.

The 3 parity bits (P_1, P_2, P_3) are located in the positions that are numbered corresponding to ascending powers of two (1, 2, 4, 8, ...).

bit 1,	bit 2,	bit 3,	bit 4,	bit 5,	bit 6,	bit 7
P_1 ,	P_2 ,	M_1 ,	P_3 ,	M_2 ,	M_3 ,	M_4

ASSIGNMENT OF PARITY BIT VALUES

The assignment of 0 or 1 value to each parity bit is very important one. Since each parity bit provides a check on certain other bits in the total code, we must know the value of these others in order to assign the parity bit value.

Step 1: Write the binary number for each decimal position number as 001, 010 ...

Step 2: Construct the bit position table and enter the information bits as shown in **Table 5.2**.

Table 5.2 : Bit Position Table for a 7 bit error correcting code

Bit designation	P_1	P_2	M_1	P_3	M_2	M_3	M_4
Bit position	1	2	3	4	5	6	7
Binary position number	001	010	011	100	101	011	111
Information bits (M_n)							
Parity bits (P_n)							

Step 3: (i) The binary position number of parity bit P_1 has a 1 for its right most digit (001). Thus parity bit checks all bit positions including itself, that have 1's in the same location in the binary position numbers (011, 101, 111). Therefore, parity bit P_1 checks bit positions 1, 3, 5 and 7.

(ii) The binary position number for parity bit P_2 has a 1 for its middle bit (010). It checks all bit positions, including itself, that have 1's in the middle position (011, 110, 111). Therefore, parity bit P_2 checks bit positions 2, 3, 6 and 7.

(iii) The binary position number for parity bit P_3 has a 1 for its left most digit (100). It checks all bit positions including itself, that have 1's in the left most bit (101, 110, 111). Therefore parity bit P_3 checks bit positions 4, 5, 6, 7.

Step 4: In each case, the parity bit is assigned a value to make the quantity of 1's in the set of bits that it checks odd or even, depending on which is specified.

This procedure is explained with the following example:

Example: Determine the single-error correcting code for the BCD number 1001 (information bits), using even parity.

Solution:

Step 1: Find the number of parity bits required

$$\text{Let } P = 3, \text{ then } 2^P = 2^3 = 8.$$

$$\text{Information bits, } M = 4$$

$$M + P + 1 = 4 + 3 + 1 = 8$$

The equation, $2^P \geq M + P + 1$ is satisfied, therefore 3 parity bits are sufficient.

$$\text{Total code bits} = M + P = 4 + 3 = 7.$$

Step 2: Construct a bit position table, and enter the information bits.

Bit designation	P_1	P_2	M_1	P_3	M_2	M_3	M_4
Bit position	1	2	3	4	5	6	7
Binary position number	001	010	011	100	101	011	111
Information bits			1		0	0	1
Parity bits							

Step 3: Parity bits are determined as follows:

For P_1 : Bit P_1 checks the bit positions 3, 5 and 7. They have two 1's and therefore to have an even parity P_1 must be 0.

For P_2 : Bit P_2 checks the bit positions 3, 6 and 7. They have two 1's and therefore to have an even parity P_2 must be 0. They have two 1's and therefore to have an even parity P_2 must be 0.

For P_3 : Bit P_3 checks the bit positions 5, 6 and 7. They have one '1' and therefore to have an even parity P_3 must be 1.

Step 4: These parity bits are entered in the table, and the resulting combined code is 0011001.

Bit designation	P_1	P_2	M_1	P_3	M_2	M_3	M_4
Bit position	1	2	3	4	5	6	7
Binary Position number	001	010	011	100	101	110	111
Information bits			1		0	0	1
Parity bits	0	0		1			

Result : 0011001

DETECTING AND CORRECTING THE ERROR

Step 1: Start with the group checked by P_1 .

Step 2: Check the group for proper parity. A '0' represents a good parity check and '1' represents a bad check.

Step 3: Repeat step 2 for each parity group.

Step 4: The binary number formed by the results of all the parity checks designates the position of the code bit that is in error. This is the error position code.

For example, the error position code is 011 means, the bit in position 3 is in error. Then correct the error by changing the value 0 or 1. Thus the error is corrected.

This procedure is explained with the following example:

Example

Assume that the code word (0011001) is transmitted and that 0010001 is received. The receiver does not know what was transmitted and must look for proper parities to determine if the code is correct. Designate any error that has occurred in transmission if even parity is used.

Solution:

Bit Designation	P_1	P_2	M_1	P_3	M_2	M_3	M_4
Bit Position	1	2	3	4	5	6	7
Binary position number	001	010	011	100	101	110	111
Received Code	0	0	1	0	0	0	1

For P_1 : Bit P_1 checks positions 1, 3, 5 and 7.

There are two 1 s in this group.

Parity check is good. → 0 (LSB)

For P_2 : Bit P_2 checks positions 2, 3, 6 and 7.

There are two 1 s in this group.

Parity check is good. → 0

For P_3 : Bit P_3 checks positions 4, 5, 6 and 7.

There are one 1 in this group.

Parity check is bad. → 1 (MSB)

The error position code = 100 i.e., 4.

This says that the bit in position 4 is in error. It is a '0' and should be a '1'. Now the corrected code is 0011001, which agrees with the transmitted code.

5.38 SEQUENTIAL PROGRAMMABLE DEVICES

For some digital design, external flip-flops are included with PLDs. Sequential programmable devices include both gates and flip flops. The device can be programmed to perform a variety of sequential circuit functions. Fig. 5.60 shows the basic sequential circuit and Fig. 5.61 shows the block diagram of a sequential programmable device.

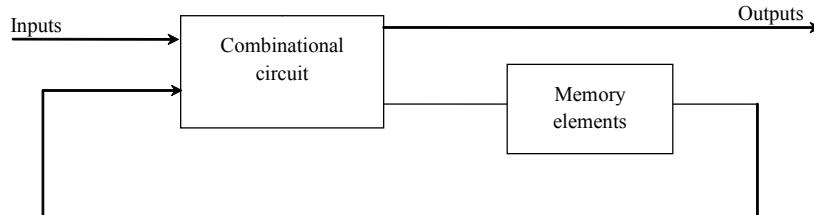


Fig. 5.60 Basic sequential circuit

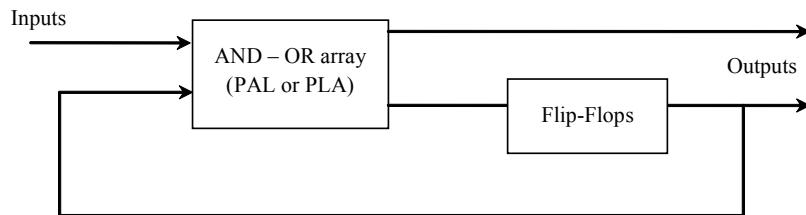


Fig. 5.61 Sequential programmable device

Types of sequential programmable devices

1. Sequential (or simple) programmable logic device (SPLD)
2. Complex programmable logic device (CPLD)
3. Field programmable gate array (FPGA)

5.38.1 Sequential programmable logic device (SPLD)

SPLD includes flip-flops and AND-OR array

- Flip-flops connected to form a register
- Flip-flops outputs could be included in product terms of AND array
- Field-programmable logic sequencer (FPLS)
 - first programmable device developed, flip-flops may be of D or JK type
 - not succeed commercially due to too many programmable connections
- Combinational PAL together with D flip-flops are mostly used

Each section of an SPLD is called a *macrocell*, which is a circuit that contains a sum of products combinational logic function and an optional flip flop. A typical SPLD contains 8-10 macrocells within one IC package.

Features:

- programming AND array
- use or bypass the flip-flop
- select clock edge polarity
- preset or clear for the register
- complement an output

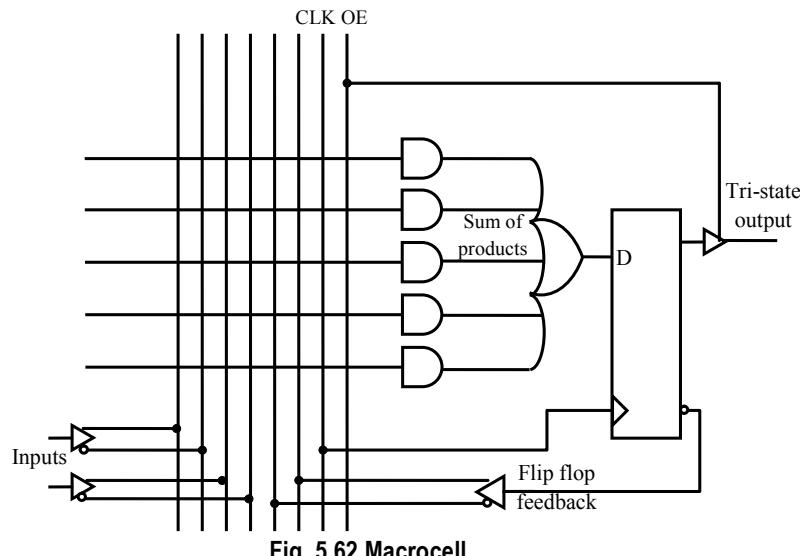


Fig. 5.62 Macrocell

Figure 5.62 shows the logic of a basic macrocell. The output is driven by an edge triggered *D* flip flop connected to a common clock input and changes state on a clock edge. The output of the flip flop is connected to a three state buffer (or inverter) controlled by an *OE* (output enable) signal. The output of the flip flop is fed back into one of the inputs of the programmable AND gates to provide the present state condition for the sequential circuit. All the flip flops are connected to the common *CLK* input, and all three state buffers are controlled by the *OE* input.

5.38.2 Complex Programmable Logic Device (CPLD)

The design of a digital system using PLDs often requires the connection of several devices to produce the complete specification. For this type of application, it is more economical to use a complex programmable logic device (CPLD), which is a collection of individual PLDs on a single

integrated circuit. A programmable interconnection structure allows the PLDs to be connected to each other in the same way that can be done with individual PLDs. Figure 5.63 shows the general configuration of a CPLD.

- CPLD is a collection of PLDs to be connected to each other through a programmable switch matrix
- input/output blocks provide connections to IC pins
- each I/O pin is driven by a three-state buffer and can be programmed to act as input or output
- switch matrix receives inputs from I/O block and directs it to individual macrocells
- selected outputs from macrocells are sent to the outputs as needed
- each PLD typically contains from 8 to 16 macrocells

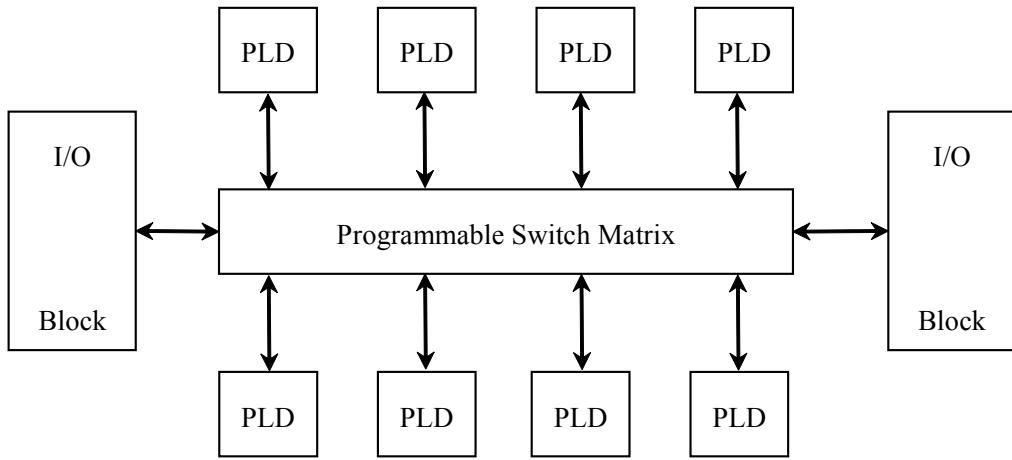


Fig. 5.63 CPLD configuration

5.38.3 Field programmable gate array (FPGA)

The field-programmable gate array (FPGA) is a semiconductor device that can be programmed after manufacturing. Instead of being restricted to any predetermined hardware function, an FPGA allows to program product features and functions, adapt to new standards, and reconfigure hardware for specific applications even after the product has been installed in the field—hence the name “field-programmable”.

FPGA is used to implement any logical function that an application-specific integrated circuit (ASIC) could perform, but the ability to update the functionality after shipping offers advantages for many applications.

FPGAs consist of various mixes of configurable embedded SRAM, high-speed transceivers, high-speed I/Os, logic blocks, and routing. Specifically, an FPGA contains programmable logic components called logic elements and a hierarchy of reconfigurable interconnects that allow the logic elements to be physically connected. Logic elements are configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flipflops or more complete blocks of memory.

As FPGAs continue to evolve, the devices have become more integrated. Hard intellectual property blocks built into the FPGA fabric provide rich functions while lowering power and cost and freeing up logic resources for product differentiation. Newer FPGA families are being developed with hard embedded processors, transforming the devices into systems on a chip (SoC).

FPGA:

- an array of hundreds or thousands of logic blocks
- surrounded by programmable input and output blocks
- connected together via programmable interconnections
- a logic block consists of look-up tables (truth tables stored in a SRAM and providing combinational circuit functions for the logic block), multiplexers, gates, and flip-flops
- SRAM instead of ROM
- Complexity: 100 - 1000(s) of Gate Equivalents. For example, Xilinx FPGA family characteristics are given in Table 5.2.

Table 5.2 Xilinx FPGA Family Characteristics

Family series	Number of I/O Blocks	Number of CLBs	Number of FFs	Estimated typical Usable Gates
XC2000	58 – 74	64 – 100	122 – 174	800 – 1800
XC3000	64 – 176	64 – 484	256 – 1320	1300 – 9000
XC4000	64 – 256	64 – 1024	256 – 2569	1600 – 25000
XC5000	148 – 244	196 – 484	784 – 1936	6000 – 15000

Applications of FPGAs include

- digital signal processing,
- software-defined radio,
- ASIC prototyping,

- medical imaging,
- computer vision,
- speech recognition,
- cryptography,
- bio informatics,
- computer hardware emulation,
- radio astronomy,
- metal detection.

Field Programmable Gate Arrays (FPGA)

The FPGA basically consists of an array of logic blocks with programmable row and column interconnecting channels surrounded by programmable I/O blocks as shown in **Figure 5.64**. Many FPGA architectures are based on a type of memory called LUT (look-up table) rather than on AND/OR arrays. Another approach is the use of multiplexers to generate logic functions. FPGA is a single VLSI (Very Large Scale Integrated) circuit constructed on a single piece of silicon.

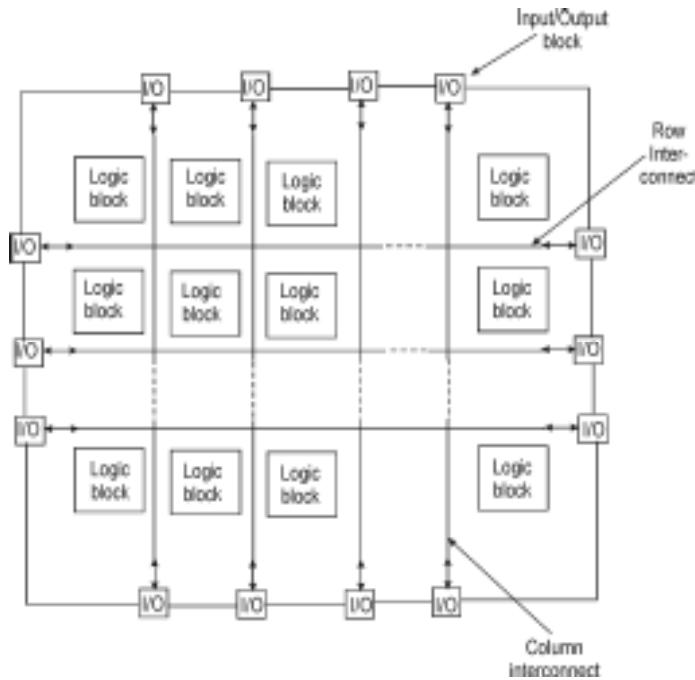


Fig. 5.64 : Basic Block Diagram of an FPGA

The module shown in **Figure 5.64** consists of 3 number of interconnected 2-to-1 multiplexer and an OR gate whose output selects the MUX 3.

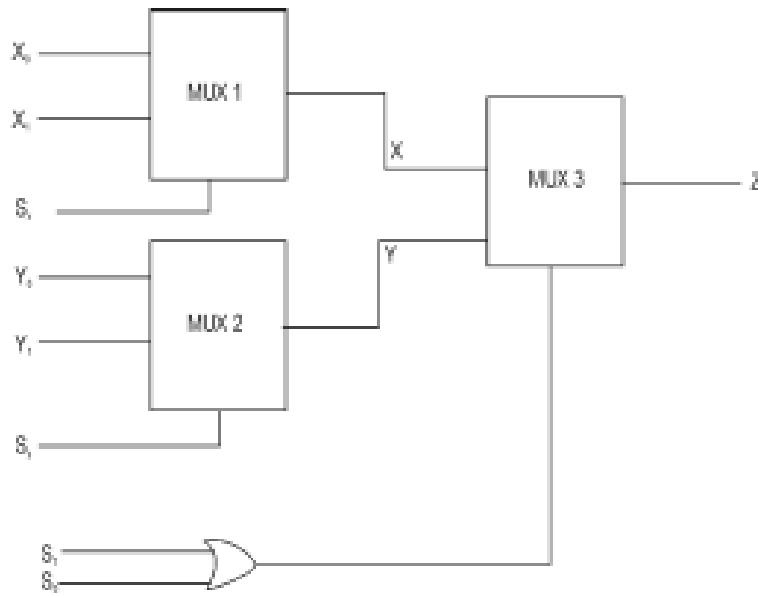


Fig. 5.65 : FPGA module

Logic expression for MUX 1, $X = X_0 \bar{S}_X + X_1 S_X$

Logic expression for MUX 2, $Y = Y_0 \bar{S}_Y + Y_1 S_Y$

By connecting the input of this module in different ways, the module can be programmed as a variety of 2 or 3 or 4 input gates. For example, by connecting X_0 , X_1 , Y_0 and S_1 to 0, the expression for output Z becomes $Z = Y_1 S_Y S_0$ and now this module acts like a 3 input positive logic AND gate. Similarly by connecting different inputs to high and low, the module acts like different gates. The logic circuit design procedure using FPGA involves the following steps:

1. Capture the logic circuit to be implemented with a suitable software package, using a library of logic elements which are various configuration of basic modules available in FPGA. In addition, many FPGA libraries also contain predesigned circuits for multiplexers, encoders, adders and so on. Predesigned circuits make design much easier.
2. Functional simulation simulates the circuit to determine whether it is function properly.
3. Configure and interconnect the modules of the FPGA to produce the desired logic circuit. This may be done automatically by a routing software called routes. Once the routing is over, it is now possible to determine the actual circuit delays which can now be introduced into the simulation model. Now an accurate simulation of the circuit can be available.
4. Programming is done by the FPGA interconnections. The routing of the devices determined in the previous step is now made into a fuse map. Then this fuse map is used in conjunction with a device programmer to make the internal device connections.
5. After programming, it must be tested. If the designed function is not fulfilled, it must be reprogrammed. With careful simulation, reprogramming can be minimized.

5.39 APPLICATION SPECIFIC INTEGRATED CIRCUITS

ASIC (Application Specific Integrated Circuit) is a chip that is custom designed for a specific application rather than a general purpose chip such as a microprocessor. The use of ASICs improve performance over general purpose CPUs, because ASICs are ‘hardwired’ to do a specific job and do not incur the overhead of fetching and interpreting stored instructions. However, a standard cell ASIC may include one or more microprocessor cores and embedded software, in which case, it may be referred to as a ‘System on Chip’ (SoC).

Programmable Logic Devices (PLDs) and FPGAs are merged as cost effective ASIC solutions because they provide low-cost prototype with nearly instant manufacturing. This class of device consist of an array of uncommitted logic elements whose interconnect structure and/or logic structure can be personalized on-site according to the user’s specification.

Types of ASIC

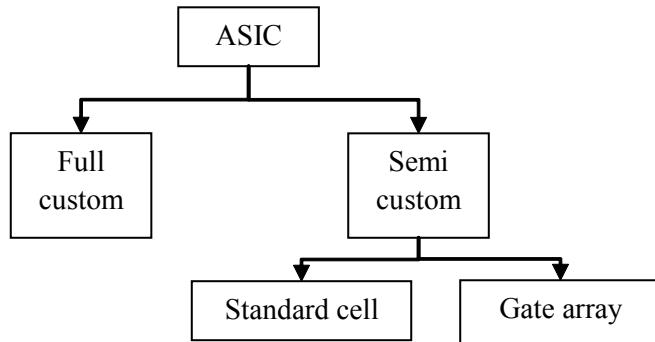


Fig. 5.66 Types of ASIC

5.39.1 Full Custom ASIC

In a full custom ASIC, an engineer designs some or all of the logic cells circuits or layout specifically for one ASIC. It makes sense to take this approach only if there are no suitable existing cell libraries available that can be used for the entire design.

- All aspects of a circuit are tailored for a particular application
- Circuit fully optimized
- Design extremely complex
- Very time consuming design
- Intel and AMD are partly full-custom

- Example: DVD, CD, Digital camera

5.39.2 Standard cell ASIC

A cell based ASIC (CBIC) uses predesigned logic cells known as standard cells. The standard cell areas also called flexible blocks in a CBIC are built of rows of standard cells. The ASIC designer defines only the placement of standard cells and the interconnect in a CBIC. All the mask layers of a CBIC are customized and are unique to a particular customer.

- Circuit made of a set pre-defined logic, known as standard cells.
- Layout of a cell is pre-determined; but layout of the complete circuit is customized.
- Example: Mobile phone digital IC

5.39.3 Gate array ASIC

A gate array approach using a prefabricated chip with active devices like AND gates, etc. that is later interconnected according to a custom order by adding metal layers in the factory environment.

5.39.4 Difference between FPGA and ASIC

Difference between ASICs and FPGAs mainly depends on cost, tool availability, performance and design flexibility. They have their own pros and cons, but it is designer's responsibility to find the advantages of the each and use either FPGA or ASIC for the product. However recent developments in the FPGA domain are narrowing down the benefits of the ASICs.

FPGA Design Advantages

- Faster time-to-market
- No NRE (Non-Recurring Expenses)
- Simpler design cycle
- More predictable project cycle
- Field reprogrammability
- Reusability
- FPGA synthesis is much more easier than ASIC

FPGA Design Disadvantages

- Power consumption in FPGA is more
- FPGA limits the design size
- Good for low quantity production. When quantity increases, cost per product increases compared to the ASIC implementation.

ASIC Design Advantages

- Full custom capability
- Low power consumption
- Smaller form factor

ASIC Design Disadvantages

- Time-to-market. Some large ASICs can take long time
- Design issues - Complexity
- Design tools are very much expensive

ANNA UNIVERSITY QUESTIONS**MEMORY DEVICES****PART-A**

1. Define Mask Programmable PLA and Field Programmable PLA. (April 2003)
2. Give the logic table of a ROM which will multiply two 2 bit binary numbers. (April 2003)
3. Distinguish between a PAL and PLA. (Nov. 2003)
4. Why is it that unlike PROMs, PALs are not a universal logic solution? (April 2004)
5. Name any two random access memories. (April 2004)
6. What is an EPROM? (April 2004)
7. Distinguish between PAL and PLA. (April 2004)
8. What is a memory cell? Give an example. (Nov. 2004)
9. What is FPLA? (Nov. 2004)
10. List any two programming technologies used in field programmable devices. (Nov. 2004)
11. How many address inputs, data inputs and data outputs are required for a 16Kx 12 memory? (Nov. 2004)
12. What is a PROM? (Nov. 2004)
13. What is a (p, n, m) PLA? (Nov. 2004)
14. What is the difference between PAL and PLA? (April 2005)
15. How is the data stored in a single DRAM cell. (April 2005)
16. Implement the following two functions using PLA.
 - (a) $f_1(A, B, C, D) = \overline{B}D + \overline{A}\overline{B}\overline{C} + A\overline{B}C + A\overline{B}\overline{C}$
 - (b) $f_2(A, B, C, D) = (\overline{A} + \overline{B} + \overline{D})(A + \overline{B} + \overline{C})(\overline{A} + B + \overline{D})(B + \overline{C} + \overline{D})$ (April 2005)
17. What makes PAL is different from PLA? (April 2005)
18. How does ROM retain information? (April 2005)
19. What is PLA? (April 2005)
20. Whether PAL is same as PLA? Explain. (Dec. 2005)
21. What is the difference between PLA and PAL? (Dec. 2005)
22. How is the read operation performed in the single transistor DRAM cell? (Dec. 2005)
23. Draw the basic dynamic memory cell. (Dec. 2005)
24. Which memory is called volatile? Why? (Dec. 2005)
25. Draw a RAM cell. (Dec. 2005)
26. Is the PAL same as the PLA? Justify. (Dec. 2005)

27. What is meant by ‘static’ and ‘dynamic’ memories? **(May 2006)**
28. How is individual location in a EEPROM programmed or erased? **(May 2006)**
29. Distinguish between PLA and PAL. **(May 2007)**
30. Determine the number of address lines required for 512 bytes of memory and for a 2 KB memory. **(May 2011)**
31. What is the difference between programmable array logic (PAL)? **(Dec 2011)**
32. What is a combinational PLD? **(Dec 2012)**
33. Compare SRAM and DRAM. **(May 2013)**

PART-B

1. List the PLA program table for the BCD to excess 3 code converter circuit and show its implementation for any two output functions. **(16) (April 2003, April 2005)**
2. Write notes on : ROM, EPROM, PLA. **(6) (April 2003)**
3. Implement a binary serial adder using SR flipflop programmable logic array. **(16) (April 2003)**
4. Design a switching circuit that converts a 4 bit binary code into a 4 bit Gray code using ROM memory. **(16) (April 2003)**
5. Explain EPROM and PLA. **(8) (Nov. 2003)**
6. Draw a RAM cell and explain its working. **(8) (Nov. 2003)**
7. Draw a neat sketch showing implementation of

$$z_1 = \bar{a}\bar{b}\bar{d}e + \bar{a}\bar{b}\bar{c}\bar{d}\bar{e} + bc + de$$

$$z_2 = \bar{a}\bar{c}e$$

$$z_3 = bc + de + \bar{c}\bar{d}\bar{e} + bd$$

$$z_4 = \bar{a}\bar{c}e + ce$$

using a $5 \times 8 \times 4$ PLA.

8. Design a combinational circuit using a ROM, that accepts a 3-bit number and generates an output binary number equal to the square of the given input number. **(16) (April 2004)**
9. Draw a diode ROM which translates from BCD 8421 code to Gray code. **(12) (Nov. 2004)**
10. Write short notes on Programmable Logic Devices. **(8) (Nov. 2004)**
11. Derive the PLA implementation of a serial binary adder. **(10) (Nov. 2004)**
12. Generate the following Boolean functions with a PAL with 4 inputs and 4 outputs.

$$Y_3 = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D} + AB\bar{C}\bar{D}$$

$$Y_2 = \overline{ABCD} + \overline{ABC}\overline{D} + ABCD$$

$$Y_1 = \overline{ABC} + \overline{AB}\overline{C} + A\overline{B}\overline{C} + A\overline{B}C$$

$$Y_0 = ABCD \quad (8) \text{ (Nov. 2004)}$$

13. Give the architecture of a typical PAL and explain with suitable example how it can be used to implement combinational and sequential circuits. **(16) (Nov. 2004)**
14. Distinguish between ROM, PAL and PLA. **(6) (Nov. 2004)**
15. Sketch the functional diagram of typical dual port SRAM and clearly explain the functions of each block. **(16) (Nov. 2004)**
16. Draw the schematic diagram of a MOS SRAM cell. Explain the read and write operations with suitable timing diagrams. **(16) (Nov. 2004)**
17. What are the classification of PLDs? Explain with example. **(16) (April 2005)**
18. Explain the structure and characteristics of the following memories:
- (i) PROM, (ii) EPROM, (iii) EEPROM **(16) (April 2005)**
19. (i) Compare SRAM and DRAM in terms of density, speed and overall circuit complexity. **(6)**
(ii) Draw the basic circuit in SRAM and explain how the read and write operation is performed. **(6)**
(iii) What is two dimensional decoding? Why is it preferred? **(4) (April 2005)**
20. A combinational logic circuit is defined by the following function: **(10) (April 2005)**

$$f_1(a, b, c) = \sum (0, 1, 6, 7); f_2(a, b, c) = \sum (2, 3, 5, 7)$$

Implement the circuit with a PLA having 3 inputs, 3 product terms and 2 outputs.

21. Compare the following PLDs: PROM, PLA and PAL. **(6) (April 2005)**
22. Write notes on EPROM. **(4) (April 2005)**
23. What is PAL? Show how a PAL is programmed for the following logic function.

$$X = \overline{ABC} + \overline{AB}\overline{C} + \overline{A}\overline{B} + AC$$

24. Write notes on semi conductor memories and methods of memory decoding. **(16) (Dec.2005)**
25. Implement the following functions using PLA.

$$F_1 = \sum m(1, 2, 4, 6)$$

$$F_2 = \sum m(0, 1, 6, 7)$$

$$F_3(x, y, z) = \sum m(2, 6)$$

26. A combinational circuit is defined by the functions **(16) (Dec. 2005)**

$$F_1 = \sum m(3, 5, 7)$$

$$F_2 = \sum m(4, 5, 7)$$

Implement the circuit with a PLA having 3 inputs, 3 product terms and two outputs.

27. (i) Using ROM, design a combinational circuit which accepts 3 bit number and generates an output binary number equivalent to square of input number. (8)
(ii) Explain the operation of bipolar RAM cell with suitable diagrams. (8)
28. (i) Draw the CMOS RAM cell and explain the read and write operation performed in the cell. (8) (Dec. 2005)
(ii) Compare the features of PROM, EPROM memory and EEPROM. (8) (Dec. 2005)
29. (i) Describe the principle of operation of PAL's. (5)
(ii) Bring the relative merits and demerits of PLD's. (5)
(iii) Discuss the applications of PAL's and PLA's. (6)
30. (i) Describe the RAM organization. (12)
(ii) A bipolar RAM chip is arranged as 16 words. How many bits are stored in the chip? (4) (Dec. 2005)
31. Write a note on: (i) MOSFET RAM cell (ii) Dynamic RAM cell. (16) (Dec. 2005)
32. A combinational circuit is defined by the functions

$$F_1(A, B, C) = \sum (3, 5, 6, 7)$$

$$F_2(A, B, C) = \sum (0, 2, 4, 7)$$

- Implement the circuit with PLA. (6) (Dec. 2005)
33. Explain the Read and Write Cycles of RAM. (8) (May 2006)
34. Design a BCD to Excess-3 code converter with (i) PLA, (ii) PAL devices. (16) (May 2006)
35. (i) How can one make 64×8 ROM using four 32×4 ROM's? Draw such a circuit and explain. (8) (May 2006)
(ii) Implement binary to excess 3 code converter using ROM. (8) (May 2006)
36. Draw a dynamic RAM cell and explain its operation. Compare its simplicity with that of NMOS static RAM Cell, by way of diagram and operation. (16) (May 2006)
37. A combinational logic circuit is defined by the functions $F = \sum m(3, 4, 5, 7, 10, 14, 15)$ and $G = \sum m(1, 5, 7, 11, 15)$. Implement the circuit using a programmable logic array with 4 inputs, 6 product terms and 2 outputs. (16) (May 2006)
38. What is micro programmed control unit? Explain the different types of ROM?. (8) (Dec. 2006)
39. Implement the given functions using PROM and PAL
 $F_1 = \sum m(0, 1, 3, 5, 7, 9)$ $F_2 = \sum m(1, 2, 4, 7, 8, 10, 11)$ (16) (May 2007)
40. Write short notes on semiconductor memories. (6) (May 2007)
41. Implement the given functions using PAL and PLA
 $F_1 = \sum m(0, 1, 2, 4, 6, 7)$ $F_2 = \sum m(1, 3, 5, 7)$ $F_3 = \sum m(0, 2, 3, 6)$ (16) (May 2007)

42. What are advantages of PLA over ROM? Explain the internal construction of PLA.
(8) (May 2007)

43. Explain the different types of ROM.
(16) (Dec. 2007)

44. Explain the operation of DRAM with suitable diagram. Also explain how Read/Write operations are performed in DRAM with timing diagram.
(16) (May 2009)

45. Implement the switching functions

$$z_1 = abde + abcde + bc + de$$

$$z_2 = ace$$

$$z_3 = bc + de + cde + bd$$

$z_4 = ace + ce$ Using a $5 \times 8 \times 4$ programmable logic array.
(16) (Dec 2010)

46. (i) Write short notes on the basic configuration of the three types of programmable Logic Devices.
(6) (May 2011)

(ii) Draw the signals of a 32×8 RAM with control input. Show the external connections necessary to have a 128×8 RAM using a decoder and replication of this RAM.

(10) (May 2011)

47. With suitable timing diagram explain how Read operation is performed in Random access memory.
(6) (Nov 2011)

48. Implement the following function using PLA

$$A(x,y,z) = \sum m(1,2,4,6)$$

$$B(x,y,z) = \sum m(0,1,6,7)$$

$$C(x,y,z) = \sum m(2,6)$$

(16) (May 2012)

49. i) Write notes on PLA and PAL.
(Nov 2012)

ii) Write notes on RAM, its operations and its types.
(Nov 2012)

50. Implement the switching functions

$$Z_1 = \overline{ab}\overline{d} + \overline{\overline{abc}}\overline{e} + bc + de$$

$$Z_2 = \overline{ace}$$

$$Z_3 = bc + de + \overline{c}\overline{d}\overline{e} + bd$$

$$Z_4 = \overline{ace} + ce \quad \text{using } 5 \times 8 \times 4 \text{ PLA}$$

(May 2013)

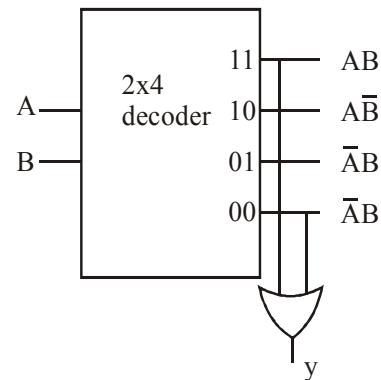
B.E/B.TECH. DEGREE EXAMINATIONS MAY / JUNE 2014**SECOND SEMESTER****COMPUTER SCIENCE AND ENGINEERING****DIGITAL PRINCIPLES AND SYSTEM DESIGN****Time: Three Hours****Max.Marks:100****ANSWER ALL QUESTIONS****PART – A (10 x 2 = 20)****1. Find the octal equivalent of hexadecimal numbers AB.CD.**

$$AB.CD_{16} = ?8$$

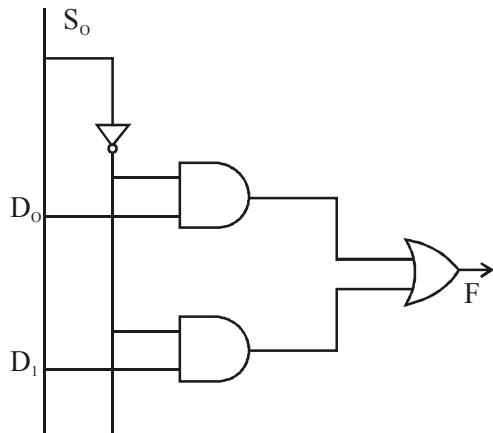
A	B	C	D
1010	1011	1100	1101
Binary \Rightarrow	101 010 110 .	110 011 010	
Octab \Rightarrow	5 2 6 .	6 3 2	
$(AB.CD)_{16} = (526.632)_8$			

2. State and prove the consensus theorem.

- In Boolean algebra, consensus theorem is the identify $x y v \bar{x} z v y z = x y v \bar{x} z$
- The consensus of terms $\bar{x} z$ and is yz
- It is the conjunction of all the unique literals of the terms, excluding the literal which appears unnegated in one term and negated in other.

3. Implement the function $G = \sum M(0,3)$ using 2x4 decoder.

4. Draw the circuit for 2:1 line multiplexer.



5. Write the characteristic table and equation of JK flip flop.

Q	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$$\text{Characteristic Equation : } Q_{n+1} = J\bar{Q} + \bar{K}Q$$

6. Write any two applications of shift register.

- * Serial to parallel converters
- * Parallel to serial converter

7. Define Race condition.

- Races exist in asynchronous sequential circuits when two or more binary state variables change during a state transition.
- Races are classified as (i) Critical races (ii) Non-critical races

8. What are the types of hazards?

- * Static hazard
- * Dynamic hazard
- * Essential hazard

9. What is memory decoding?

In addition storage units, decoding circuits are needed to select the memory word specified by input address. It is done by techniques as absolute decoding, linear decoding.

10. Define ASIC.

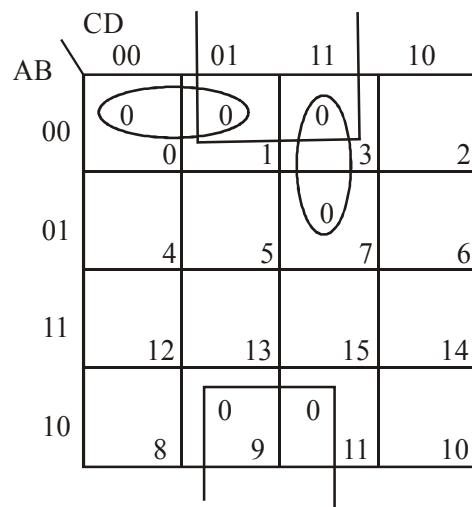
- * Application specific integrated circuit is a chip that is custom designed for specific application rather than a general purpose chip such as microprocessor.
- * Types of ASIC includes full custom and semi custom.
- * Semicustom has standard cell and Gate array types.

PART B- (5 * 16=80marks)

11. (a) Simplify the functions.

(16)

$$G = \prod M(0, 1, 3, 7, 11)$$



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

$$(ii) f(k_1, x, y, z) = \Sigma m(0, 7, 8, 9, 10, 12) + \Sigma d(2, 5, 13)$$

Refer page number 1.89 Problem No.1.141

(b) Minimize the expression using Quine McCluskey (Tabulation) method.

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

Refer page number 1.107 Example 1.158

12. (a) Design a circuit that converts 8421 BCD code to Excess-3 code.

Refer page number 2.33 to 2.35

(b) Implement the following Boolean function using 8 to 1. Multiplexer $F(A,B,C,D) = A'BD' + ACD + B'CD + A'C'D$. Also implement the function using 16 to 1 multiplexer.

Refer page number 2.56 & Page 2.49

13. (a) Implement T flip flop using D flip flop and JK flip flop using D flip flop.

Refer page number 3.17 to 3.18

(OR)

(b) Design a synchronous counter which counts in the sequence 000, 001, 010, 011, 100, 101, 110, 111, 000 using D.F.F.

Refer page number 3.84 to 3.86

14. (a) Explain the steps for the design of asynchronous sequential circuits.

Refer page number 4.3 to 4.6 & 4.7, Sec : 4.7

(OR)

(b) Implement the switching function $F = \Sigma m(1, 3, 5, 7, 8, 9, 14, 15)$ by a static hazard free two level AND OR gate network.

Refer page number 4.42

15. (a) Implement the following function using PLA.

$$A(x, y, z) = \Sigma m(1, 2, 4, 6)$$

$$B(x, y, z) = \Sigma m(01, 6, 7)$$

$$C(x, y, z) = \Sigma m(2, 6)$$

Refer page number 5.36 Problem 5.10

(OR)

(b) The following messages have been coded in the even parity Hamming code and transmitted through a noisy channel. Decode the messages assuming that at most a single error has occurred in each codeword.

(i) 1001001

(ii) 0111001

(iii) 1110110

(iv) 0011011

Refer page number: 5.55

B.E/B.Tech. DEGREE EXAMINATIONS MAY / JUNE 2014**Second Semester****Computer Science and Engineering****CS 6201 - DIGITAL PRINCIPLES OF SYSTEM DESIGN****Time : Three Hours****Maximum : 100 Marks****ANSWER ALL QUESTIONS****PART – A (10 x 2 = 20)**

1. Find the octal equivalent of hexadecimal numbers AB.CD.
2. State and prove the consensus theorem.
3. Implement the function $G = \sum m(0, 3)$ using a 2×4 decoder.
4. Draw the circuit for 2-to-1 line multiplexer.
5. Write the characteristics table and equation of JK flip flop.
6. Write any two applications of shift register.
7. Define Race condition.
8. What are teh types of hazards.
9. What is memory decoding?
10. Define ASIC.

PART B- (5 * 16=80marks)

11. (a) Simplify the following functions using K-Map technique.

(i) $G = \overline{\sum} M(0, 1, 7, 9, 11)$

(ii) $f(W, X, Y, Z) = \sum m(0, 7, 8, 9, 10, 12) + \sum d(2, 5, 13)$

(OR)

- (b) Minimize the expression using Quine McCluskey (Tabulation) method

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

12. (a) Design a circuit that converts 8421 BCD code to Excess-3 code.

(OR)

- (b) Implement the following Boolean function using 8 to 1. Multiplexer $F(A, B, C, D) = A'B'D' + ACD + B'CD + A'C'D$. Also implement the function using 16 to 1 multiplexer.

13. (a) Implement T flip flop using D flip flop and JK flip flop using D flip flop.

(OR)

(b) Design a synchronous counter which counts in the sequence 000, 001, 010, 011, 100, 101, 110, 111, 000 using D-FF.

14. (a) Explain the steps for the design of asynchronous sequential circuits.

(OR)

(b) Implement the switching function $F = Sm(1, 3, 5, 7, 8, 9, 14, 15)$ by static hazard free two level AND OR gate network.

15. (a) Implement the following function using PLA.

$$A(x, y, z) = Sm(1, 2, 4, 6)$$

$$B(x, y, z) = Sm(0, 1, 6, 7)$$

$$C(x, y, z) = Sm(2, 6)$$

(OR)

(b) The following messages have been coded in the even parity Hamming code and transmitted through a noisy channel. Decode the messages, assuming that at most a single error has occurred in each codeword.

(i) 1001001

(ii) 0111001

(iii) 1110110

(iv) 0011011

B.E/B.Tech. DEGREE EXAMINATIONS NOVEMBER / DECEMBER 2014**Second Semester****Computer Science and Engineering****CS 6201 - DIGITAL PRINCIPLES OF SYSTEM DESIGN****Time : Three Hours****Maximum : 100 Marks****ANSWER ALL QUESTIONS****PART – A (10 x 2 = 20)**

1. State the principle of duality.
2. Implement AND gate using only NOR gates.
3. Implement the following Boolean function using 8:1 multiplexer $F(A, B, C) = \Sigma m(1, 3, 5, 6)$.
4. Define hazard.
5. Distinguish Moore and Mealy circuit.
6. With reference to a JK flip flop, what is racing?
7. How many states are there in a 3-bit ring counter? What are they?
8. What is a Priority Encoder?
9. Whether PAL is same as PLA? Explain.
10. What is a volatile memory? Give example.

PART B- (5 * 16=80marks)

11. (a) Simplify the function $F(w, x, y, z) = \Sigma m(2, 3, 12, 13, 14, 15)$ using tabulation method, Implement the simplified function using gates. using K-Map technique.

(OR)

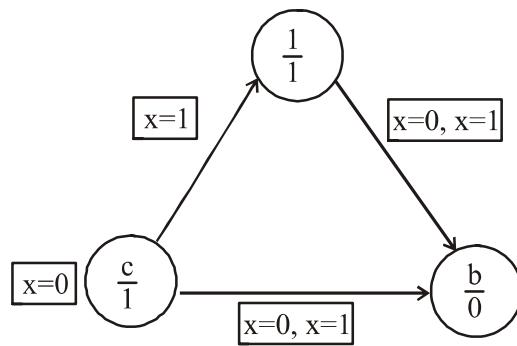
- (b) (i) Simplify the Boolean function in Sum of Products (SOP) and Product of Sums (POS) $F(A, B, C, D) = \Sigma m(0, 2, 5, 8, 9, 10)$

- (ii) Plot the following Boolean function in Karnaugh map and simplify it. $F(w, x, y, z) = \Sigma m(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

12. (a) Design and implement a 8421 to gray code converter. Realize the converter using only NAND gates.

(OR)

- (b) Design 2-bit Magnitude Comparator and write a Verilog HDL code.
13. (a) Design a MOD-10 Synchronous counter using JK flip-flops. Write execution table and state table.
- (OR)
- (b) (i) How race condition can be avoided in a flip flop?
- (ii) Realize the sequential circuit for the state diagram shown below.



14. (a) An asynchronous sequential circuit is described by the following excitation and output function.

$$Y = X_1 X_2 + (X_1 + X_2)Y$$

$$Z = Y$$

- (i) Draw the logic diagram of the circuit.
(ii) Derive the transition table and output map.
(iii) Describe the behaviour of the circuit.

(OR)

- (b) Design a synchronous counter using JK flip-flop to count the following sequence 7, 4, 3, 1, 5, 0, 7...

15. (a) Design a BCD to Excess-3 code converter and implement using suitable PLA.

(OR)

- (b) Discuss on the concept of working and applications of semiconductor memories.

B.E/B.Tech. DEGREE EXAMINATIONS APRIL / MAY 2015**Second Semester****Computer Science and Engineering****CS 6201 - DIGITAL PRINCIPLES OF SYSTEM DESIGN****Time : Three Hours****Maximum : 100 Marks****ANSWER ALL QUESTIONS****PART – A (10 x 2 = 20)**

1. Convert $(0.6875)_{10}$ to binary.
2. Prove the following using DeMorgan's theorem.
 $[(x+y)'+(x+y)'] = x+y$
3. Implement the full adder with 4 x 1 Multiplexer.
4. Write the Data flow description of a 4-bit Comparator.
5. Give the block diagram of Master-Slave D flip-flop.
6. What is Ring counter?
7. Compare asynchronous and synchronous sequential circuit.
8. What is critical race condition? Give example.
9. Difference between EEPROM and PROM.
10. How to detect double error and correct single error?

PART B- (5 * 16=80marks)

11. (a) Simplify the following Boolean expression in

- (i) Sum-of-product
- (ii) Product-of-sum using Karnaugh-map.

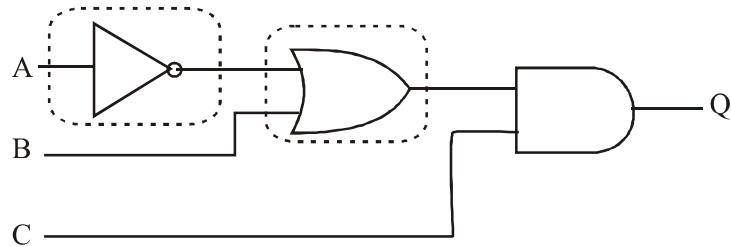
$$AC' + B'D + A'CD + ABCD$$

(OR)

- (b) (i) Express the following function in sum of min-terms and product of max-terms

$$F(x, y, z) = x = yz.$$

- (ii) Convert the following logic system into NAND gates only.



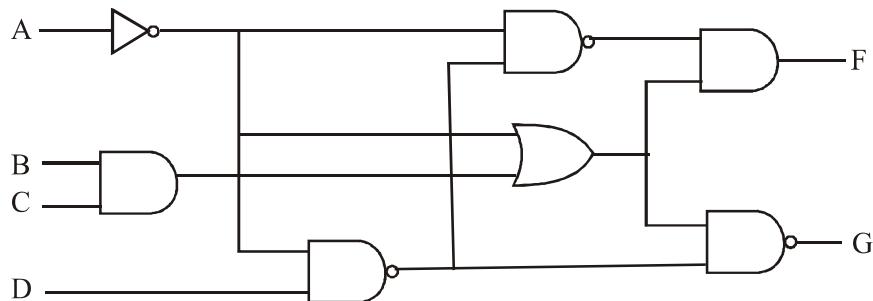
12. (a) (i) Implement the following Boolean functions with a multiplexer :

$$F(w, x, y, z) = \Sigma(2, 3, 5, 6, 11, 14, 15)$$

(ii) Construct a 5 to 32 line decoder using 3 to 8 line decoders and 2 to 4 line decoder.

(OR)

(b) (i) Explain the Analysis procedure. Analyze the following logic diagrams.



(ii) With neat diagram the 4-bit adder with carry lookahead.

13. (a) (i) A sequential circuit with two D flip-flops A and B, one input x, and one output z is specified by the following next-state and output equations.

$$A(t=1) = A' + B, B(t+1) = B'x, z = A+B'$$

(1) Draw the logic diagram of the circuit.

(2) Derive the state table.

(3) Draw the state diagram of the circuit.

(ii) Explain the difference between a state table, characteristic table and an excitation table.

(OR)

(b) Consider the design of a 40bit BCD counter that counts in the following way :

0000, 0010, 0011,, 1001 and back to 0000.

(i) Draw the state diagram.

(ii) List the next state table.

(iii) Draw the logic diagram of the circuit.

14. (a) (i) Explain the Race-free state assignment procedure.

(ii) Reduce the number of states in the following state diagram. Tabulate the reduced state table and draw the reduced state diagram.

Present state	Next state	Output
	$x = 0 \quad x = 1$	$x = 0 \quad x = 1$
a	a b	0 0
b	c d	0 0
c	a d	0 0
d	e f	0 1
e	a f	0 1
f	g f	0 1
g	a f	0 1

(OR)

(b) Explain the hazards in combinational circuit and sequential circuit and also demonstrate a hazard and its removal with example.

15. (a) (i) Write short note on Address multiplexing.

(ii) Briefly discuss the sequential programmable devices.

(OR)

(b) (i) Implement the following two Boolean functions with a PLA.

$$F1 = A B' + A C + A' B C'$$

$$F2 = (AC + BC)'$$

(ii) Give the Internal block diagram of 4 x 4 RAM.

Question paper Code : 25059

B.E/B.Tech Degree Examination, November/December 2018.

Third Semester.

Computer Science and Engineering

CS 8351 – DIGItal Principles and System Design

(Common to Electronics and Telecommunication Engineering / information Technology)

(Regulations 2017)

Time: Three hours

Maximum: 100 marks

Answers all questions

Part –A (10×2=20 marks)

1. State the classification of binary codes?
2. Define Associative law.
3. Draw 1:8 Demultiplexer using two 1:4 Demultiplexers.
4. What is propagation delay?
5. State the operation of T flip-flop.
6. Mention the different types of shift registers.
7. What is race around condition?
8. Define state table.
9. List the major differences between PLA and PAL.
10. What is field programmable logic array?

Part B – (5×13=65 marks)

11. (a) Write short notes on Demorgan's theorem, Absorption law and Consensus law. (13)

Or

- (b) (i) Convert the following Boolean expression into standard SOP form:

$$AB'C + A'B' + ABC'D \quad (6)$$

- (ii) Express the Boolean function $F = A + B'C$ in a sum of minterms (SOP) (7)

12. (a) Explain in detail about encoders and decoders. (13)

Or

(b) Design 32 to 1 multiplexer using four 8 to 1 multiplexer and 2 to 4 decoder. (13)

13. (a) Design and implementation of SR Flip-flop using NOR gate. (13)

Or

(b) Explain in detail about 4 bit Johnson counter. (13)

14. (a) Find the circuit that has no static hazard and implement the Boolean function

$$F(A, B, C, D) = \sum m(1, 5, 6, 7). \quad (13)$$

Or

(b) What are called as essential hazard? How does the hazard occur in sequential circuits? How can the same be eliminated using SR latches? Give an example? (13)

15. (a) Illustrate with neat sketch and describe the categories of RAM. (13)

Or

(b) With neat diagrams describe the working principle of programmable array logic. (13)

Part C – (1×15=15 marks)

16. (a) Design a decade counter using JK flip flops using IC 74LS112D. (15)

Or

(b) Declare a module that describe a circuit that is specified with the following two Boolean expressions: (15)

$$E = A + BC + B'D$$

$$F = B'C + BC'D$$

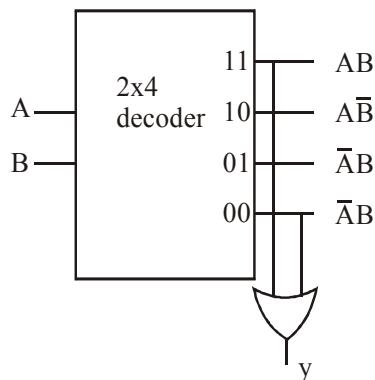
B.E/B.TECH. DEGREE EXAMINATIONS MAY / JUNE 2014**SECOND SEMESTER****COMPUTER SCIENCE AND ENGINEERING****CS 6201 - DIGITAL PRINCIPLES AND SYSTEM DESIGN (Reg. 2013)****Time: Three Hours****Max.Marks:100****ANSWER ALL QUESTIONS****PART – A (10 x 2 = 20)****1. Find the octal equivalent of hexadecimal numbers AB.CD.**

$$AB.CD_{16} = ?_8$$

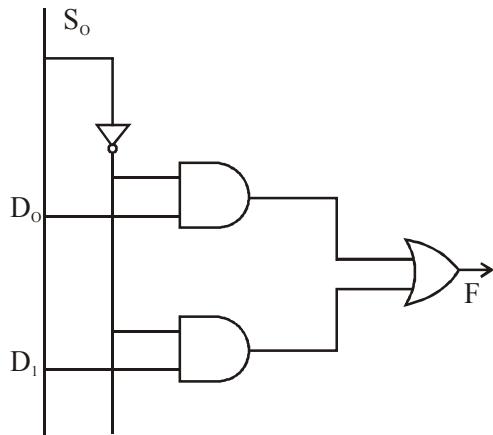
	A	B	.	C	D
	1010	1011	.	1100	1101
Binary \Rightarrow	101	010	110	110	011 010
Octab \Rightarrow	5	2	6	6	3 2
	$(AB.CD)_{16} = (526.632)_8$				

2. State and prove the consensus theorem.

- In Boolean algebra, consensum theorem is the identify $x y \vee \bar{x} z \vee y z = x y \vee \bar{x} z$
- The consensus of terms $\bar{x} z$ and is yz
- It is the conjunction of all the unique litexals of the terms, excluding the literal which appears innegated in one term and regated in other.

3. Implement the function $G = \sum M(0,3)$ using 2x4 decoder.

4. Draw the circuit for 2:1 line multiplexer.



5. Write the characteristic table and equation of JK flip flop.

Q	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$$\text{Characteristic Equation : } Q_{n+1} = J\bar{Q} + \bar{K}Q$$

6. Write any two applications of shift register.

- * Serial to parallel converters
- * Parallel to serial converter

7. Define Race condition.

- Races exist in asynchronous sequential circuits when two or more binary state variables change during a state transition.
- Races are classified as (i) Critical races (ii) Non-critical races

8. What are the types of hazards?

- * Static hazard
- * Dynamic hazard
- * Essential hazard

9. What is memory decoding?

In addition storage units, decoding circuits are needed to select the memory word specified by input address. It is done by techniques as absolute decoding, linear decoding.

10. Define ASIC.

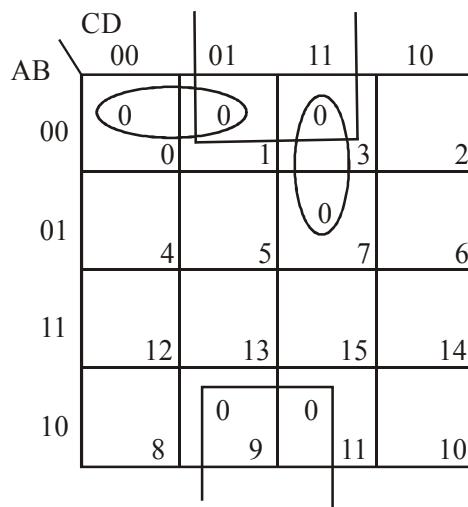
- * Application specific integrated circuit is a chip that is custom designed for specific application rather than a general purpose chip such as microprocessor.
- * Types of ASIC includes full custom and semi custom.
- * Semicustom has standard cell and Gate array types.

PART B- (5 * 16=80marks)

11. (a) Simplify the functions.

(16)

$$G = \prod M(0, 1, 3, 7, 11)$$



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

(ii) $f(k_1, x, y, z) = \Sigma m(0, 7, 8, 9, 10, 12) + \Sigma d(2, 5, 13)$

Refer page number 1.89 Problem No. 1.141

(b) Minimize the expression using Quine McCluskey (Tabulation) method.

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

Refer page number 1.107 Example 1.158

12. (a) Design a circuit that converts 8421 BCD code to Excess-3 code.

Refer page number 2.33 to 2.35

(b) Implement the following Boolean function using 8 to 1. Multiplexer $F(A,B,C,D) =$

$$A'BD' + ACD + B'CD + A'C'D. \text{ Also implement the function using 16 to 1 multiplexer.}$$

Refer page number 2.56 & Page 2.49

13. (a) Implement T flip flop using D flip flop and JK flip flop using D flip flop.

Refer page number 3.17 to 3.18

(OR)

(b) Design a synchronous counter which counts in the sequence 000, 001, 010, 011, 100, 101, 110, 111, 000 using D.F.F.

Refer page number 3.84 to 3.86

14. (a) Explain the steps for the design of asynchronous sequential circuits.

Refer page number 4.3 to 4.6 & 4.7, Sec : 4.7

(OR)

(b) Implement the switching function $F = \Sigma m(1, 3, 5, 7, 8, 9, 14, 15)$ by a static hazard free two level AND OR gate network.

Refer page number 4.42

15. (a) Implement the following function using PLA.

$$A(x, y, z) = \Sigma m(1, 2, 4, 6)$$

$$B(x, y, z) = \Sigma m(01, 6, 7)$$

$$C(x, y, z) = \Sigma m(2, 6)$$

Refer page number 5.36 Problem 5.10

(OR)

(b) The following messages have been coded in the even parity Hamming code and transmitted through a noisy channel. Decode the messages assuming that at most a single error has occurred in each codeword.

(i) 1001001

(ii) 0111001

(iii) 1110110

(iv) 0011011

Refer page number: 5.55

ASCII Characters

binary	MSN	0000		0001		0010		0011		0100		0101		0110		0111	
LSN	hex	0		1		2		3		4		5		6		7	
0000	0	NUL	00	DLE	16 10	SP	32 20	0	48 30	@	64 40	P	80 50		96 60	p	112 70
0001	1	SOH	101	XON (DC1)	17 11	!	33 21	1	49 31	A	65 41	Q	81 51	a	97 61	q	113 71
0010	2	STX	202	DC2	18 22	"	34 22	2	50 32	B	66 42	R	82 52	b	98 62	r	114 72
0011	3	ETX	303	XOFF (DC2)	19 13	#	35 23	3	51 33	C	67 43	S	83 53	c	99 63	s	115 73
0100	4	EOT	404	DC4	20 14	\$	36 24	4	52 34	D	68 44	T	84 54	d	100 64	t	116 74
0101	5	ENQ	505	NAK	21 15	%	37 25	5	53 35	E	69 45	U	85 55	e	101 65	u	117 75
0110	6	ACK	606	SYN	22 16	&	38 26	6	54 36	F	70 46	V	86 56	f	102 66	v	118 76
0111	7	BEL	707	ETB	23 17	'	39 27	7	55 37	G	71 47	W	87 57	g	103 67	w	119 77
1000	8	BS	808	CAN	24 18	(40 28	8	56 38	H	72 48	X	88 58	h	104 68	x	120 78
1001	9	HT	909	EM	25 19)	41 29	9	57 39	I	73 49	Y	89 59	i	105 69	y	121 79
1010	A	LF	100A	SUB	26 1A	*	42 2A	:	58 3A	J	74 4A	Z	90 5A	j	106 6A	z	122 7A
1011	B	VT	110B	ESC	27 1B	+	43 2B	;	59 3B	K	75 4B	[91 5B	k	107 6B	{	123 7B
1100	C	FF	12OC	FS	28 1C	,	44 2C	<	60 3C	L	76 4C	\	92 5C	l	108 6C	l	124 7C
1101	D	CR	13OD	GS	29 1D	-	45 2D	=	61 3D	M	77 4D]	93 5D	m	109 6D	}	125 7D
1110	E	SO	14OE	RS	30 1E	.	46 2E	>	62 3E	N	78 4E	^	94 5E	n	110 6E	~	126 7E
1111	F	SI	15OF	US	31 1F	/	47 2F	?	63 3F	O	79 4F	-	95 5F	o	111 6F	DEL	127 7F

Hex	Name	Description	Hex	Name	Description
00	NUL	Null ^@	10	DLE	Data Link Escape
01	SOH	Start of Heading	11	XON	Xmit ON ^Q
02	STX	Start of TeXt	12	DC2	Device Control 2
03	ETX	End of TeXt	13	XOFF	Xmit OFF ^S
04	EOT	End of Transmission	14	DC4	Device Control 4
05	ENQ	ENQuiry	15	NAK	Negative Acknowledge
06	ACK	ACKnowledge	16	SYn	SYNchronous idle
07	BEL	BELI (beep) ^G	17	ETB	End of Transmission
		Block			
08	BS	BackSpace ^H	18	CAN	CANcel
09	TAB	Horizontal TAB ^I	19	EM	End of Medium
0A	LF	Line Feed ^J	1A	SUB	SUBstitute
0B	VT	Vertical Tab	1B	ESC	ESCAPE ^l
0C	FF	Form Feed ^L	1C	FS	File Separator
0D	CR	Carriage Return	1D	GS	Group Separator
0E	SO	Shift Out	1E	RS	Record Separator
0F	SI	Shift In	1F	US	Unit Separator

The values to the *right* of the ASCII character are that character's ASCII values in decimal and hexadecimal. The LSN column signifies the *least significant nibble* (4 bits) of the binary value, the MSN row is the *most significant nibble* of the binary value, thus the commercial-at (@) sign has an LSN of 0000 and an MSN of 0100, making 01000000. The second row and column show the LSN and MSN in their hexadecimal (base 16) values.

ASCII values 32 decimal to 126 decimal are printable ASCII characters.

The position of the Pound Sign (£) varies significantly between different interpretations of the ASCII set. In older versions it can sometimes be found at 35 decimal, in place of the hash symbol (#), but is now more often found at 163 decimal in the extended ASCII set.

ASCII is used in almost all computer systems except for some IBM midrange and mainframe systems that use EBCDIC.

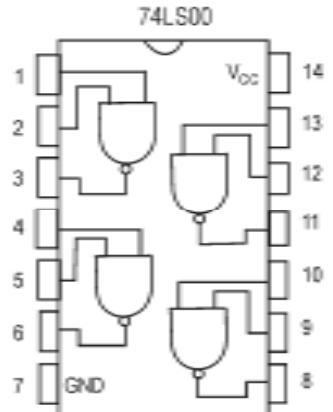
TTL DEVICES

Number	Function
7400	Quad 2-input NAND gates
7401	Quad 2-input NAND gates (open collector)
7402	Quad 2-input NOR gates
7403	Quad 2-input NOR gates (open collector)
7404	Hex inverters
7405	Hex inverters (open collector)
7406	Hex inverter buffer-driver
7407	Hex buffer-drivers
7408	Quad 2-input AND gates (open collector)
7409	Quad 2-input AND gates (open collector)
7410	Triple 3-input NAND gates
7411	Triple 3-input AND gates
7412	Triple 3-input NAND gates (open collector)
7413	Dual Schmitt triggers
7414	Hex Schmitt triggers
7416	Hex inverter buffer-drivers
7417	Hex buffer-drivers
7420	Dual 4-input NAND gates
7422	Dual 4-input AND gates
7423	Expandable dual 4-input NOR gates
7425	Dual 4-input NOR gates
7426	Quad 2-input TTL-MOS interface NAND
7427	Triple 3-input NOR gates
7428	Quad 2-input NOR buffer
7430	8-input NAND gate
7432	Quad 2-input OR gates
7437	Quad 2-input NAND buffers
7438	Quad 2-input NAND buffers (open collector)
7439	Quad 2-input NAND buffers (open collector)
7440	Dual 4-input NAND buffers
7441	BCD-to-decimal decoder-Nixie driver
7442	BCD-to-decimal decoder
7443	Excess 3-to-decimal decoder
7444	Excess Gray-to-decimal
7445	BCD-to-decimal decoder-driver
7446	BCD-to-seven segment decoder-drivers
7447	BCD-to-seven segment decoder-drivers
7448	BCD-to-seven segment decoder-drivers
7450	Expandable dual 2-input 2-wide AND-OR-
7451	Dual 2-input 2-wide AND-OR-INVERT gates
7452	Expandable 2-input 4-wide AND-OR gates
7453	Expandable 2-input 4-wide AND-OR gates
7454	2-input 4-wide AND-OR-INVERT gates
7455	Expandable 4-input 2-wide AND-OR-INVERT gates
7459	Dual 2-3 input 2-wide AND-OR-INVERT gates
7460	Dual 4-input expanders

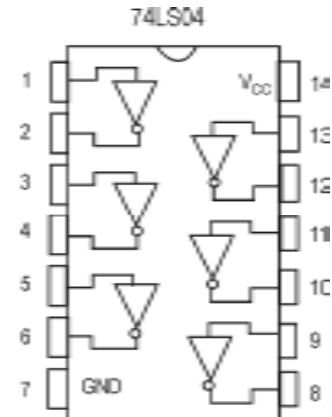
Number	Function
7461	Triple 3-input expanders
7462	2-2-3-3 input 4-wide expanders
7464	2-2-3-4 input 4-wide AND-OR-INVERT gates
7465	4-wide AND-OR-INVERT gates
7470	Edge-triggered JK flip-flop
7472	JK master-slave flip-flop
7473	Dual JK master-slave flip-flop
7474	Dual D flip-flop
7475	Quad latch
7476	Dual JK master-slave flip-flop
7480	Gates full adder
7482	2-bit binary full adder
7483	4-bit binary full adder
7485	4-bit magnitude comparator
7486	Quad EXCLUSIVE-OR gate
7489	64-bit random-access read-write memory
7490	Decade counter
7491	8-bit shift register gates
7492	Divide-by-12 counter
7493	4-bit binary counter
7494	4-bit shift register
7495	4-bit right-shift-left-shift register
7496	5-bit parallel-in-parallel-out shift register
74100	4-bit bistable latch
74104	JK master-slave flip-flop
74105	JK master-slave flip-flop
74107	Dual JK master-slave flip-flop
74109	Dual JK positive-edge-triggered flip-flop
74116	Dual 4-bit latches with clear
74121	Monostable multivibrator
74122	Monostable multivibrator with clear
74123	Monostable multivibrator
74125	Three-state quad bus buffer (30-V output)
74126	Three-state quad bus buffer
74132	Quad Schmitt trigger (15-V output)
74136	Quad 2-input EXCLUSIVE-OR gate
74141	BCD-to-decimal decoder-driver
74142	BCD counter-latch-driver INVERT gates
74145	BCD-to-decimal decoder-driver
74147	10/4 priority encoder
74148	Priority encoder
74150	16-line-to-1-line multiplexer
74151	8-channel digital multiplexer
74152	8-channel data selector-multiplexer
74153	Dual 4/1 multiplexer
74154	4-line-to-16-line decoder-demultiplexer
74155	Dual 2/4 demultiplexer
74156	Dual 2/4 demultiplexer

Number	Function
74157	Dual 2/1 data selector
74160	Decade counter with asynchronous clear
74161	Synchronous 4-bit counter
74162	Synchronous 4-bit counter
74163	Synchronous 4-bit counter
74164	8-bit serial shift register
74165	Parallel-load 8-bit serial shift register
74166	8-bit shift register
74173	4-bit three-state register
74174	Hex F flip-flop with clear
74175	Quad D flip-flop with clear
74176	35-MHz presettable decade counter
74177	35-MHz presettable binary counter
74179	4-bit parallel-access shift register
74180	8-bit odd-even parity generator-checker
74181	Arithmetic-logic unit
74182	Look-ahead carry generator
74184	BCD-to-binary converter
74185	Binary-to-BCD converter
74189	Three-state 64-bit random-access memory
74190	UP-down decade counter
74191	Synchronous binary up-down counter
74192	Binary up-down counter
74193	Binary up-down counter
74194	4-bit directional shift register
74195	4-bit parallel-access shift register
74196	Presettable decade counter
74197	Presettable binary counter
74198	8-bit shift register
74199	8-bit shift register
74221	Dual one-shot Schmitt trigger
74251	Three-state 8-channel multiplexer
74259	8-bit addressable latch
74276	Quad JK flip-flop
74279	Quad debouncer
74283	4-bit binary full adder with fast carry
74284	Three-state 4-bit multiplexer
74285	Three-state 4-bit multiplexer
74365	Three-state hex buffers
74366	Three-state hex buffers
74367	Three-state hex buffers
74368	Three-state hex buffers
74390	Individual clocks with flip-flops
74393	Dual 4-bit binary counter

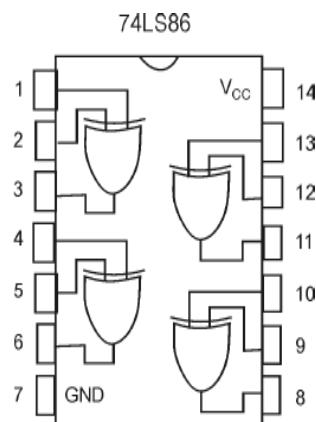
TTL CIRCUITS



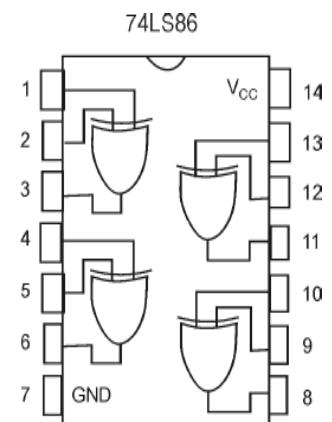
Quad Two input NAND



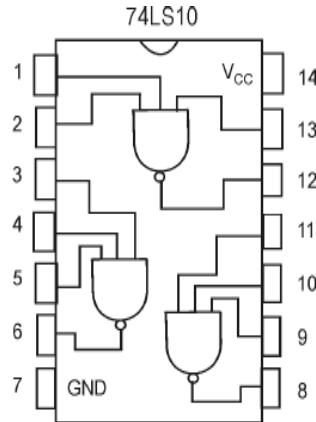
Hex Inverters



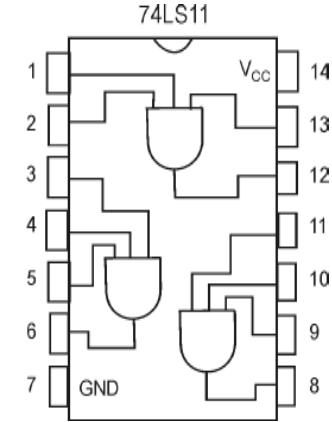
Quad-XOR



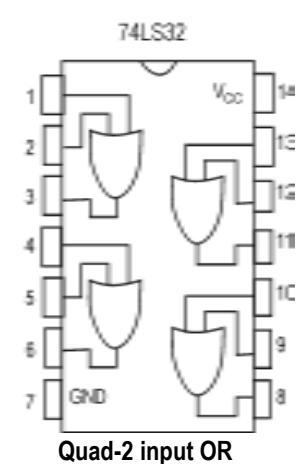
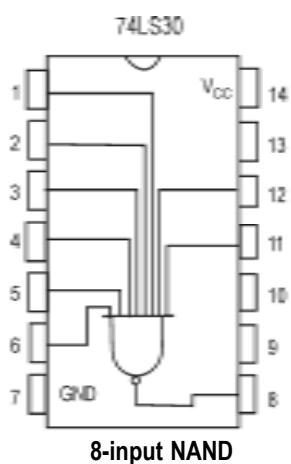
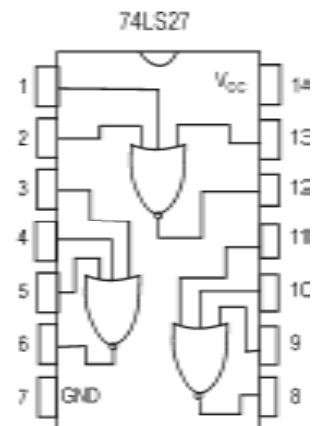
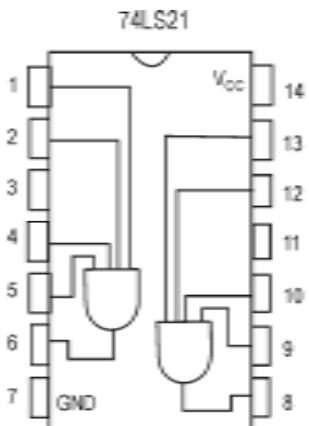
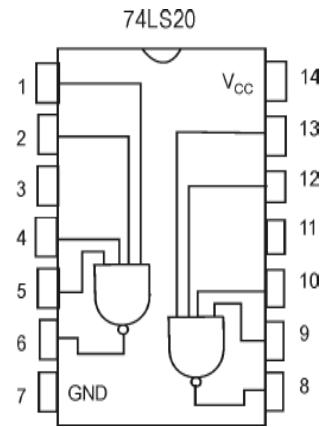
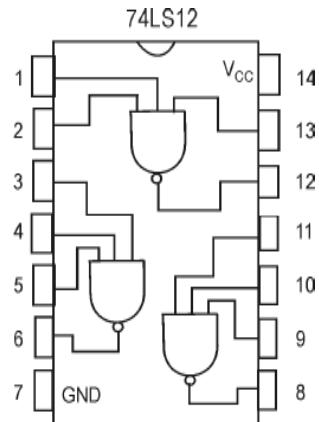
Quad-2 input AND



Triple-3 input NAND



Triple-3 input AND



CMOS DEVICES

74HC00 Series

Part No.	Pins	Function
74HC00	14	Quad 2-input NAND gate
74HC02	14	Quad 2-input NOR gate
74HC04	14	Hex inverter (buffered)
74HC08	14	Quad 2-input AND gate
74HC14	14	Hex inverting Schmitt trigger
74HC20	14	Dual 4-input NAND gate
74HC30	14	8-input NAND gate
74HC32	14	Quad 2-input OR gate
74HC42	16	BCD-to-decimal decoder
74HC74	14	Dual D flip-flop with preset and clear
74HC85	16	4-bit magnitude comparator
74HC123	16	Dual monostable multivibrator
74HC132	14	Quad 2-input NAND Schmitt trigger
74HC138	16	3-to 8-line decoder
74HC139	16	Expandable dual 2- to 4-line decoder
74HC154	24	4- to 16-line decoder (use 24SLP socket)
74HC161	16	Synchronous binary counter
74HC163	16	Synchronous binary counter
74HC164	14	8-bit serial in-parallel out shift register
74HC165	16	8-bit parallel in-serial out shift register
74HC174	16	Hex D Flip-Flop with clear
74HC175	16	Quad D type flip-flop with clear
74HC191	16	Up-down binary counter
74HC192	16	Synchronous decade up-down counter
74HC193	16	Synchronous binary up-down counter
74HC221	16	Dual monostable multivibrator
74HC240	20	Inverting octal tri-state buffer
74HC244	20	Octal tri-state buffer
74HC245	20	Octal tri-state transceiver
74HC257	16	Quad 2-channel tri-state multiplexer
74HC273	20	Octal D flip-flop
74HC367	16	Tri-state hex buffer
74HC373	20	Tri-state octal D-type latch
74HC374	20	Tri-state octal D-type flip-flop
74HC390	16	Dual 4-bit decade counter
74HC393	14	Dual 4-bit binary counter
74HC541	20	Octal buffer-line driver (tri-state)
74HC573	20	Tri-state octal D-type latch
74HC574	20	Tri-state octal D-type flip-flop
74HC595	16	8-bit serial-to-parallel shift register latch
74HC688	20	8-bit magnitude comparator (equality detector)
74HC942	20	Full duplex low-speed 300-baud modem chip
74HC943	20	Full duplex 300-baud modem chip
74HC4017	16	Decade counter-divider with 10 decoded outputs
74HC4020	16	14-stage binary counter
74HC4040	16	12-stage binary counter
74HC4046	16	CMOS phase-lock loop
74HC4060	16	14-stage binary counter
74HC4066	14	Quad analog switch
74HC4514	24	4- to 16-line decoder with latch (use 24SLP socket)
74HC4538	16	Dual retriggerable monostable multivibrator

REFERENCES

1. Morris Mano.M, Digital Design, Prentice Hall of India Pvt. Ltd., New Delhi, 2003.
2. John.M.Yarbrough, Digital Logic Applications and Design, Thomson-Vikas Publishing House, New Delhi, 2002.
3. John.F.Wakerly, Digital Design Principles and Practices, 3rd Edition, Pearson Education, New Delhi, 2002.
4. Thomas.L.Floyd, Digital Fundamentals, 8th Edition, Pearson Education, Inc, New Delhi, 2003.
5. Richard.S.Sandige, Modern Digital Design, McGRAW-HILL Publishing Company, 1990.
6. Charles.H.Roth, Jr, Fundamentals of Logic Design, 4th edition, Jaico Publishing House, 2002.
7. Donald.P.Leach, Albert Paul Malvino, Digital Principles and Applications, 5th Edition, Tata McGraw-Hill, 2003.
8. Taub.H, Schilling.D, Digital Integrated Electronics, McGraw Hill, 1977.
9. Millman.J, Jalkias.C.C, Integrated Electronics, McGraw Hill, 1972.
10. Fletcher.W.I, An Engineering Approach to Digital Design, Prentice Hall, 1996.
11. Jain. R.P, Modern Digital Electronics, Tata McGraw Hill, 2003.
12. Sedha.R.S, Digital Electronics, S.Chand & Company Ltd, 2004.
13. Salivahanan.S, Arivazhagan.S, Digital Circuits and Design, Vikas Publishing House Pvt. Ltd., 2004.
14. Bhasker.J, Verilog HDL Synthesis, BS Publications, 2001.
15. Millman.J, Taub.H, Pulse, Digital and Switching Waveforms, McGraww Hill, 1965.
16. Hall.D.V, Micro Processor and Digital Systems, McGraw Hill, 1980.
17. Gothmann.W.H, Digital Electronics-Introduction Theory and Practice, PHI, 1992.
18. Theodore, F, Bogart. JR, Introduction to Digital Circuits, McGraw-Hill, 1992.
19. Donald D. Givone, Digital Principles and Design, Tata McGraw-Hill, 2003.
20. National Semi-Conductor, Data Book, 1976.