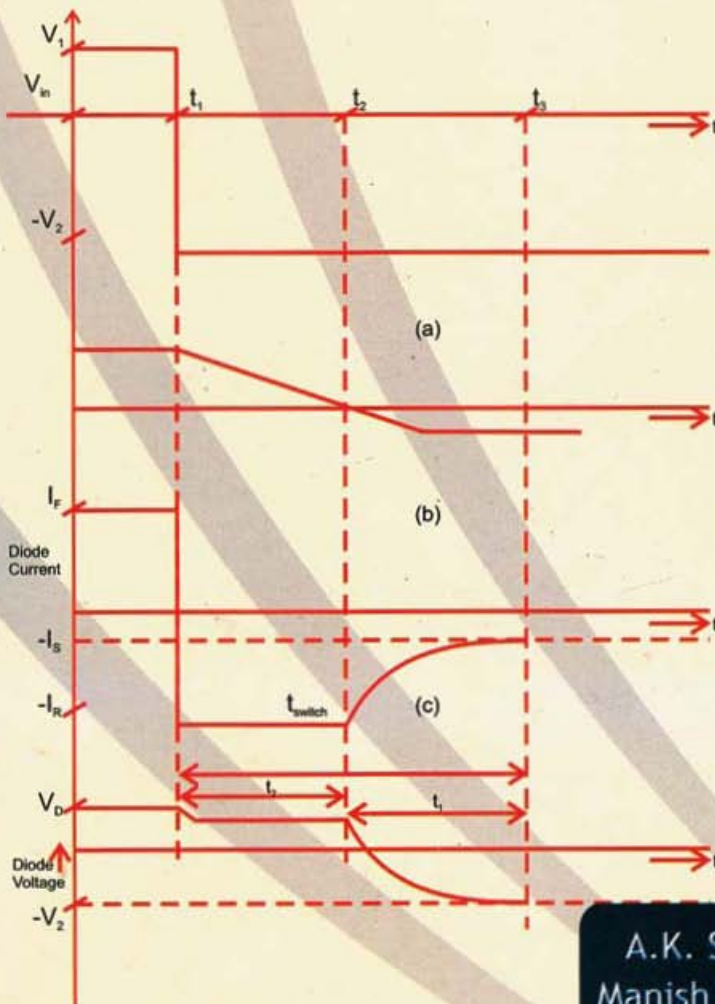


# Digital Principles Switching Theory



A.K. Singh  
Manish Tiwari  
Arun Prakash



**Digital Principles**  
**Switching Theory**

**THIS PAGE IS  
BLANK**

# DIGITAL PRINCIPLES

# SWITCHING

# THEORY

A.K. Singh

Manish Tiwari

Arun Prakash

*Department of Electronics*

*& Instrumentation Engineering*

*Northern India Engineering College*

*Lucknow.*



PUBLISHING FOR ONE WORLD

**NEW AGE INTERNATIONAL (P) LIMITED, PUBLISHERS**

New Delhi • Bangalore • Chennai • Cochin • Guwahati • Hyderabad  
Jalandhar • Kolkata • Lucknow • Mumbai • Ranchi

Visit us at [www.newagepublishers.com](http://www.newagepublishers.com)

Copyright © 2006, New Age International (P) Ltd., Publishers  
Published by New Age International (P) Ltd., Publishers

---

All rights reserved.

No part of this ebook may be reproduced in any form, by photostat, microfilm, xerography, or any other means, or incorporated into any information retrieval system, electronic or mechanical, without the written permission of the publisher. *All inquiries should be emailed to [rights@newagepublishers.com](mailto:rights@newagepublishers.com)*

**ISBN (10) : 81-224-2306-X**

**ISBN (13) : 978-81-224-2306-8**

**PUBLISHING FOR ONE WORLD**

**NEW AGE INTERNATIONAL (P) LIMITED, PUBLISHERS**

4835/24, Ansari Road, Daryaganj, New Delhi - 110002

Visit us at [www.newagepublishers.com](http://www.newagepublishers.com)

*Dedicated to*  
*Our Parents*

*Though no one can go back and make a brand new start,  
any one can start from now and make a brand new end.*

**THIS PAGE IS  
BLANK**

# PREFACE

Digital Electronic is intended as a comprehensive text for the courses in digital electronic circuits. The objective of this book is to develop in the reader the ability to analyze and design the digital circuits. The increased uses of digital technology in objects used for day-to-day life necessitate an in-depth knowledge of the subject for the professionals and engineers.

There are lots of references available on Switching Theory and Basic Digital Circuits, which discuss various topics separately. But through this text our notion was to discover the topics rather than to cover them. This comprehensive text fulfills the course requirement on the subject of digital circuit design for B. Tech degree course in Electronics, Electronics and Communication, Electronics and Electrical, Electronics and Instrumentation, Electronics Instrumentation and Control, Instrumentation and Control Engineering of different technical Universities. This text is also bound to serve as a useful reference book for various competitive examinations.

There is no special pre-requisite before starting this book. Each chapter of the book starts with simple facts and concepts, and traverse through the examples & figures it uncovers the advanced topics.

The book starts from chapter 0. It is very obvious because in the world of digital electronics the very first level is 0 and then comes the last level called 1. This is the reason why all the chapters of this book have subsections numbered starting from 0. The book has 11 well-organized chapters and 2 appendices.

Chapter 0 is introduction and is a must for all the beginners as it introduces the concept of digital signals and digital systems. It attempts to answer why and where the digital circuits are used what are their advantages. Chapter 1 deals with number systems and their arithmetic. It includes an exhaustive set of solved examples and exercise to clarify the concepts. Chapter 2 introduces the basic building blocks of digital electronics. It starts with basic postulates, Boolean algebra and then introduces logic gates. It also deals with several types of types of implementation using logic gates. For beginners we strongly recommend to work out this chapter twice before proceeding further.

Chapter 3 deals with the Boolean function minimization techniques using Postulates and Boolean Algebra, K-Map and Quine-McCluskey methods. Chapter 4 presents various combinational logic design using the discrete logic gates and LSI & MSI circuits. This chapter also deals with hazards and fault detection. Chapter 5 introduces the Programmable Logic



Devices. It also deals with basics of ROM, and then moves towards PLAs, PALs, CPLDs and FPGA.

Chapter 6 introduces the clocked (synchronous) sequential circuits. It starts with discussions on various flip-flops their triggering and flip-flop timings. It then deals with analysis and design of synchronous circuits and concludes with sequence detector circuits. Chapter 7 deals with shift registers and counters. It introduces the basic idea of shift registers and then discusses various modes and application of shift registers. It then introduces the various types and modes of counters and concludes with applications. Chapter 8 deals with asynchronous sequential circuits. It elaborates the analysis and design procedures with different considerations. Chapter 9 introduces the Algorithmic State Machine. It starts with basic concepts, design tools and concludes with design using multiplexers.

Chapter 10 introduces the fundamentals of digital integrated circuits- The Digital Logic Families. The chapter has an indepth analysis of semiconductor switching devices and various logic families with detailed qualitative and quantitative descriptions. Circuits are simplified topologically and equivalent circuits are drawn separately to clarify the concepts. Chapter 11 deals with the semiconductor memory devices to be used with computer system. It also includes memory system designs and introduces Magnetic and Optical memories also.

The text also includes two rich appendices giving information about ICs fabrication, various digital ICs, and lists various digital ICs.

All the topics are illustrated with clear diagram and simple language is used throughout the text to facilitate easy understanding of the concepts. The authors welcome constructive suggestion and comments from the readers for the improvement of this book at [singh\\_a\\_k@rediffmail.com](mailto:singh_a_k@rediffmail.com) or at [manishtiwari\\_me@rediffmail.com](mailto:manishtiwari_me@rediffmail.com)

**A.K. SINGH**  
**MANISH TIWARI**  
**ARUN PRAKASH**

## **ACKNOWLEDGEMENT**

This book is the result of the dedication and encouragement of many individuals. We are also thankful to our Directors and all our colleagues at BBD Group of Educational Institutions.

We would like to thank our family members especially wife and daughter for their patience and continuing support and our parents for their blessings.

We are indebted to our friend and colleague Arun Prakash, for his invaluable contribution and involvement in the project.

We thankfully acknowledge the contributions of various authors, data manuals, journals, reference manuals etc. from where materials have been collected to enrich the contents of the book.

Finally, We would like to thank the people at New Age International (P) Limited, especially Mr. L.N. Mishra, who continues support and encourages writing and who made the book a reality. Thanks are also due to Mr. Soumya Gupta, M.D. New Age International (P) Limited for his involvement in the project.

In last but not the least by the blessing of almighty and good fortune we get such a supporting and cooperative people around us who in one way or other help us to complete this project in time.

**A.K. SINGH  
MANISH TIWARI  
ARUN PRAKASH**

**THIS PAGE IS  
BLANK**

# CONTENTS

<i>Preface</i>	(v)
<i>Acknowledgement</i>	(vii)
<b>CHAPTER 0: INTRODUCTION TO DIGITAL ELECTRONICS</b>	<b>1</b>
<b>CHAPTER 1: NUMBERING SYSTEMS</b>	<b>12</b>
1.0 Introduction	12
1.1 Numbering Systems	12
1.1.1 A Review of the Decimal System	12
1.1.2 Binary Numbering System	12
1.1.3 Binary Formats	14
1.2 Data Organization	15
1.2.1 Bits	15
1.2.2 Nibbles	15
1.2.3 Bytes	16
1.2.4 Words	17
1.2.5 Double Words	17
1.3 Octal Numbering System	18
1.3.1 Octal to Decimal, Decimal to Octal Conversion	19
1.3.2 Octal to Binary, Binary to Octal Conversion	19
1.4 Hexadecimal Numbering System	20
1.4.1 Hex to Decimal, Decimal to Hex Conversion	21
1.4.2 Hex to Binary, Binary to Hex Conversion	21
1.4.3 Hex to Octal, Octal to Hex Conversion	21
1.5 Range of Number Representation	22
1.6 Binary Arithmetic	24
1.7 Negative Number & Their Arithmetic	26
1.7.1 1's & 2's Complement	27
1.7.2 Subtraction Using 1's & 2's Complement	29
1.7.3 Signed Binary Representation	31
1.7.4 Arithmetic Overflow	33
1.7.5 9's & 10's Complement	34

1.7.6	$r$ 's Complement and $(r-1)$ 's Complement	35
1.7.7	Rules for Subtraction using $r$ 's and $(r-1)$ 's Complement	35
1.8	Binary Coded Decimal (BCD) & Its Arithmetic	37
1.9	Codes	40
1.9.1	Weighted Binary Codes	40
1.9.2	Non-Weighted Codes	43
1.9.3	Error Detecting Codes	45
1.9.4	Error Correcting Codes	47
1.9.5	Hamming Code	49
1.9.6	Cyclic Codes	52
1.10	Solved Examples	54
1.11	Exercises	62
<b>CHAPTER 2: DIGITAL DESIGN FUNDAMENTALS–BOOLEAN ALGEBRA &amp; LOGIC GATES</b>		<b>63</b>
2.0	Introductory Concepts of Digital Design	63
2.1	Truth Table	63
2.2	Axiomatic Systems and Boolean Algebra	65
2.2.1	Huntington's Postulate	66
2.2.2	Basic Theorems and Properties of Boolean Algebra	67
2.3	Boolean Functions	69
2.3.1	Transformation of a Boolean Functions into Logic Diagram	71
2.3.2	Complement of a Function	71
2.4	Representation of a Boolean Function	72
2.4.1	Minterm & Maxterm Realization	73
2.4.2	Standard Forms—SOP & POS	75
2.4.3	Conversion between Standard Forms	77
2.5	Digital Logic Gates	77
2.5.1	Positive & Negative Logic Designation	77
2.5.2	Gate Definition	78
2.5.3	The and Gate	79
2.5.4	The or Gate	80
2.5.5	The Inverter & Buffer	82
2.5.6	Other Gates & Their Function	84
2.5.7	Universal Gates	84
2.5.8	The Exclusive OR (Ex-OR) Gate	88
2.5.9	The Exclusive NOR (Ex-NOR) Gate	91

2.5.10	Extension to Multiple Inputs in Logic Gates	92
2.6	NAND-NOR Implementation (Two Level)	97
2.6.1	Implementation of a Multistage (Or Multilevel) Digital Circuit Using NAND Gates Only	97
2.6.2	Implementation of a Multistage (Or Multilevel) Digital Circuits Using NOR Gates Only	99
2.7	Exercises	101
<b>CHAPTER 3: BOOLEAN FUNCTION MINIMIZATION TECHNIQUES</b>		<b>112</b>
3.0	Introduction	112
3.1	Minimization using Postulates & Theorems of Boolean Algebra	112
3.2	Minization using Karnaugh Map (K-Map) Method	113
3.2.1	Two and Three Variable K-Map	114
3.2.2	Boolean Expression Minization Using K-Map	116
3.2.3	Minimization in Products of Sums Form	119
3.2.4	Four Variable K-Map	120
3.2.5	Prime and Essential Implicants	123
3.2.6	Don't Care Map Entries	124
3.2.7	Five Variable K-Map	125
3.2.8	Six Variable K-Map	127
3.2.9	Multi Output Minimization	129
3.3	Minimization Using Quine-McCluskey (Tabular) Method	130
3.4	Exercises	136
<b>CHAPTER 4: COMBINATIONAL LOGIC</b>		<b>141</b>
4.0	Introduction	141
4.1	Arithmetic Circuits	143
4.1.1	Adders	143
4.1.2	Subtractors	146
4.1.3	Code Converters	149
4.1.4	Parity Generators & Checkers	153
4.2	MSI & LSI Circuits	155
4.2.1	Multiplexers	156
4.2.2	Decoders (DeMultiplexers)	159
4.2.3	Encoders	167
4.2.4	Serial & Parallel Adders	169
4.2.5	Decimal Adder	174
4.2.6	Magnitude Comparators	177

4.3	Hazards	179
4.3.1	Hazards in Combinational Circuits	179
4.3.2	Types of Hazards	181
4.3.3	Hazard Free Realizations	182
4.3.4	Essential Hazards	184
4.3.5	Significance of Hazards	185
4.4	Fault Detection and Location	185
4.4.1	Classical Method	185
4.4.2	The Fault Table Method	186
4.4.3	Fault Direction by Path Sensitizing	189
4.5	Exercises	192
<b>CHAPTER 5: PROGRAMMABLE LOGIC DEVICES</b>		<b>196</b>
5.0	Introduction	196
5.1	Read Only Memory (ROM)	196
5.1.1	Realizing Logical Functions with ROM	198
5.2	PLAs : Programmable Logical Arrays	199
5.2.1	Realizing Logical Functions with PLAs	201
5.3	PALs : Programmable Array Logic	202
5.3.1	Commercially Available SPLDS	204
5.3.2	Generic Array Logic	204
5.3.3	Applications of PLDs	205
5.4	Complex Programmable Logic Devices (CPLD)	206
5.4.1	Applications of CPLDs	207
5.5	FPGA : Field Programmable Gate Arrays	207
5.5.1	Applications of FPGAs	209
5.6	User-Programmable Switch Technologies	210
5.7	Exercises	211
<b>CHAPTER 6: SYNCHRONOUS (CLOCKED) SEQUENTIAL LOGIC</b>		<b>213</b>
6.0	Introduction	213
6.1	Flip Flops	214
6.1.1	RS Flip Flop	216
6.1.2	D-Flip Flop	220
6.1.3	Clocked of Flip Flop	222
6.1.4	Triggering of Flip Flops	231
6.1.5	JK & T Flip Flop	232
6.1.6	Race Around Condition & Solution	235

6.1.7	Operating Characteristics of Flip-Flop	236
6.1.8	Flip-Flop Applications	237
6.2	Flip-Flop Excitation Table	238
6.3	Flip Flop Conversions	239
6.4	Analysis of Clocked Sequential Circuits	241
6.5	Designing of Clocked Sequential Circuits	246
6.6	Finite State Machine	250
6.7	Solved Examples	256
6.7	Exercises	262
<b>CHAPTER 7: SHIFT REGISTERS AND COUNTERS</b>		<b>265</b>
7.0	Introduction	265
7.1	Shift Registers	265
7.2	Modes of Operation	268
7.2.1	Serial IN – Serial Out Shift Registers (SISO)	268
7.2.2	Serial IN – Parallel Out Shift Registers (SIPO)	269
7.2.3	Parallel IN – Serial Out Shift Registers (PISO)	270
7.2.4	Parallel IN – Parallel Out Shift Registers (PIPO)	270
7.2.5	Bidirectional Shift Registers	270
7.3	Applications of Shift Registers	271
7.3.1	To Produce Time Delay	271
7.3.2	To Simplify Combinational Logic	271
7.3.3	To Convert Serial Data to Parallel Data	272
7.4	Counters	272
7.4.1	Introduction	272
7.4.2	Binary Ripple Up-Counter	272
7.4.3	4-Bit Binary Ripple Up-Counter	275
7.4.4	3-Bit Binary Ripple Down Counter	277
7.4.5	Up-Down Counters	278
7.4.6	Reset and Preset Functions	279
7.4.7	Universal Synchronous counter Stage	280
7.4.8	Synchronous Counter ICs	282
7.4.9	Modulus Counters	287
7.4.10	Counter Reset Method (Asynchronous Counter)	288
7.4.11	Logic Gating Method	295
7.4.12	Design of Synchrons Counters	299
7.4.13	Lockout	305



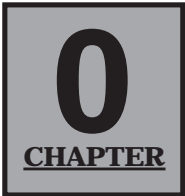
7.4.14	MSI Counter IC 7490 A	307
7.4.15	MSI Counter IC 7492 A	313
7.4.16	Ring Counter	316
7.4.17	Johnson Counter	318
7.4.18	Ring Counter Applications	322
7.5	Exercises	328
<b>CHAPTER 8: ASYNCHRONOUS SEQUENTIAL LOGIC</b>		<b>331</b>
8.0	Introduction	331
8.1	Difference Between Synchronous and Asynchronous	333
8.2	Modes of Operation	334
8.3	Analysis of Asynchronous Sequential Machines	335
8.3.1	Fundamental Mode Circuits	335
8.3.2	Circuits Without Latches	335
8.3.3	Transition Table	338
8.3.4	Flow Table	339
8.3.5	Circuits with Latches	340
8.3.6	Races and Cycles	345
8.3.7	Pulse-Mode Circuits	346
8.4	Asynchronous Sequential Circuit Design	349
8.4.1	Design Steps	349
8.4.2	Reduction of States	351
8.4.3	Merger Diagram	352
8.5	Essential Hazards	353
8.6	Hazard-Free Realization Using S-R Flip-Flops	354
8.7	Solved Examples	357
8.8	Exercises	361
<b>CHAPTER 9: ALGORITHMIC STATE MACHINE</b>		<b>362</b>
9.0	Introduction	362
9.1	Design of Digital System	362
9.2	The Elements and Structure of the ASM Chart	363
9.2.1	ASM Block	365
9.2.2	Register Operation	365
9.2.3	ASM Charts	366
9.2.4	MOD-5 Counter	368
9.2.5	Sequence Detector	369

9.3	Timing Consideration	371
9.4	Data Processing Unit	375
9.5	Control Design	376
9.5.1	Multiplexer Control	377
9.5.2	PLA Control	379
9.6	Exercises	379
<b>CHAPTER 10: SWITCHING ELEMENTS &amp; IMPLEMENTATION OF LOGIC GATES</b>		<b>382</b>
10.0	Introduction	382
10.1	Fundamentals of Semiconductors and Semiconductor Switching Device	382
10.1.1	Semiconductors	382
10.1.2	Semiconductor Diode or PN Junction	385
10.1.3	Bipolar Junction Transistor (BJTs)	391
10.2	Characteristics of Logic Families	403
10.2.1	Classifications of Logic Families	403
10.2.2	Characteristics of Digital ICs and Families	404
10.3	Implementation of Logic Families	407
10.3.1	Basic Diode Logic	407
10.3.2	Resistor Transistor Logic (RTL)	410
10.3.3	Direct Coupled Transistor Logic (DCTL)	415
10.3.4	Diode Transistor Logic (DTL)	415
10.3.5	High Threshold Logic (HTL)	422
10.3.6	Transistor-Transistor Logic (TTL)	423
10.3.7	Emitter Coupled Logic (ECL)	431
10.3.8	MOS Logic	438
10.3.9	Three State Logic (TSL)	444
10.4	Interfacing of Logic Gates	446
10.4.1	TTL to CMOS Interfacing	446
10.4.2	CMOS to TTL Interfacing	447
10.5	Comparison of Logic Families	448
10.8	Exercises	448
<b>CHAPTER 11: MEMORY FUNDAMENTALS</b>		<b>452</b>
11.0	Introduction	452
11.1	Memory Basics	452
11.2	Memory Characteristics	453

11.3	Mass Storage Devices	455
11.3.1	Magnetic Memory	455
11.3.2	Optical Memory	457
11.4	Semiconductor Memory	458
11.4.1	Basic Memory Unit	458
11.4.2	Basic Memory Organization	459
11.4.3	Cell Organization (Memory Addressing)	460
11.4.3.1	Matrix Addressing	461
11.4.3.2	Address Decoding	461
11.4.4	Organizing Word Lengths (Different Memory Organization)	464
11.4.5	Classification of Semiconductor Memory	468
11.4.6	Semiconductor Memory Timing	469
11.4.6.1	Memory Write Operation	470
11.4.6.2	Memory Read Operation	471
11.4.7	Read Only Memory	472
11.4.7.1	Some Simple ROM Organizations	473
11.4.7.2	Mask Programmed ROMs	475
11.4.8	Programmable Read Only Memory (PROM)	476
11.4.8.1	Bi-Polar PROMS	477
11.4.8.2	MOS PROMS	478
11.4.8.3	PROM Programming	478
11.4.9	Programmable Read Only Memory (EPROM)	479
11.4.9.1	EPROM Programming	480
11.4.9.2	The 27XXX EPROM Series	480
11.4.10	Electrically Erasable Programmable Read Only Memory (EEPROM)	481
11.4.11	Random Access Memory (RAM)	482
11.4.12	Static Random Access Memory (SRAM)	482
11.4.12.1	The Bi-Polar SRAM Cell	483
11.4.12.2	The MOS SRAM Cell	484
11.4.12.3	SRAM ICs	485
11.4.13	Dynamic Random Access Memory (DRAM)	486
11.4.13.1	Basic DRAM Cell	486
11.4.13.2	One MOS Transistor DRAM Cell	487
11.4.13.3	DRAM Organization	488
11.4.14	SRAMS and DRAMS	489

11.4.15 Memory System Design	492
11.4.15.1 Determining Address Lines and Address Range	492
11.4.15.2 Parallel and Series Connections of Memory	493
11.4.15.3 Address Space Allocation	494
11.4.15.4 Formation of Memory System	495
11.5 Exercises	505
<b>APPENDICES</b>	<b>517</b>
A: Integrated Circuits Fabrication Fundamentals	
B: Digital ICs	
<b>REFERENCES</b>	<b>517</b>
<b>INDEX</b>	<b>518</b>

**THIS PAGE IS  
BLANK**



# INTRODUCTION TO DIGITAL ELECTRONICS

---

## 0.1 INTRODUCTION

Engineers generally classify electronic circuits as being either analog or digital in nature. Historically, most electronic products contained electronic circuitry. Most newly designed electronic devices contain at least some digital circuitry. This chapter introduces you to the world of digital electronics.

What are the clues that an electronic product contains digital circuitry? Signs that a device contains digital circuitry include:

1. Does it have an alphanumeric (shows letters and numbers) display?
2. Does it have a memory or can it store information?
3. Can the device be programmed?

If the answer to any one of the three questions is yes, then the product probably contains digital circuitry.

Digital circuitry is quickly becoming pervasive because of its advantages over analog including:

1. Generally, digital circuits are easier to design using modern integrated circuits (ICs).
2. Information storage is easy to implement with digital.
3. Devices can be made programmable with digital.
4. More accuracy and precision is possible.
5. Digital circuitry is less affected by unwanted electrical interferences called noise.

The very basic digital design can be defined as the science of organizing arrays of simple switches into what is called a discrete system that perform transformations on two-level (binary) information in a meaningful and predictable way. Certainly this is true, but digital design as we know it today is much more exciting than this definition pretends. Digital design has matured since 1938 when Claude Shannon systemized the earlier theoretical work of George Boole (1854). Since then, design methods and devices have been developed, tried and proven, leading the way into one of the most fascinating and challenging fields of study.

Keep in mind that seldom will you find a field of study as vast as that of digital design and its related applications, yet seldom will you find a field in which you can become more productive in as short a period of time.

In short, with a limited background in other basic sciences and a desire to create, you can start designing digital circuits in a limited period of time.

## 2 Switching Theory

Digital design is contrasting yet complementary to yet another developmental science we call ANALOG DESIGN, which over the years has produced systems such as radio, analog computers, stereo, and all sorts of other conveniences that we classify as ANALOG or CONTINUOUS systems. However, it is interesting to note that it is becoming increasingly difficult to delineate the two technologies because of the inevitable integration of the two. For example, you can now purchase a DIGITAL STEREO POWER AMPLIFIER, capable of delivering some 250 watts per channel. Until recently linear amplifier design has been one of the strongest bastions of the analog world, but now we have this component that incorporates the advantages of two technologies, resulting in a superior product. This same sort of mix is witnessed each day in all areas of measurement instrument design where we see digital voltmeters, digital oscilloscopes, switching power supplies etc.

The next five sections are intended to familiarize you with some of the basics of both sciences so that you can better appreciate the applications of both and how they relate to each other. The rest of the text is devoted to helping you develop an in-depth understanding of digital design. The methodology of the text is step by step learning. We proceed using a rather poignant statement made by Rudyard Kipling as a guide to rest of our studies:

I had six honest serving men

Who taught me all they knew.

Their names were WHERE, and WHAT, and WHEN, and WHY, and HOW, and WHO.

### Where are Digital Circuits Used?

Digital electronics is a fast growing field, as witnessed by the widespread use of microcomputers. Microcomputers are designed around complex ICs called microprocessors. In addition many IC semiconductor memories makes up the microcomputer. Microcomputers with microprocessors, interface chips and semiconductor memories have started the PC revolution. Small computers that used to cost tens of thousands of dollars now cost only hundreds. Digital circuits housed in ICs are used in both large and small computers.

Other examples include:

Calculator

Peripheral devices

Robotics

Digital timepiece

Digital capacitance meter

Frequency counters

Function generator

### What and When Digital?

A system can be defined mathematically as a unique transformation or operator that maps or transforms a given input condition into a specific output.

We classify systems in one of the two ways:

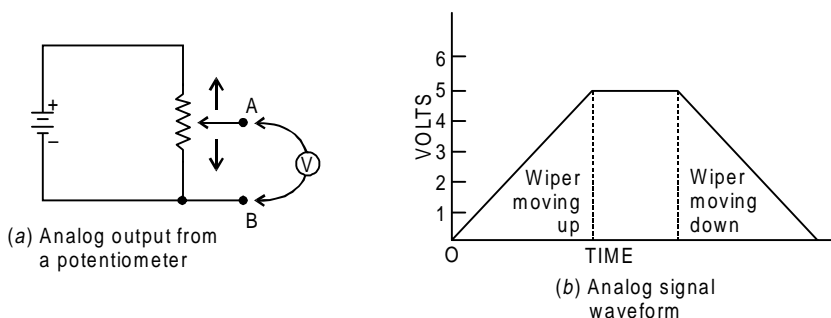
(i) Analog or continuous

(ii) Digital or discrete

An analog system operates with an analog or continuous signal and a digital system operates with a digital or discrete signal. A signal can be defined as useful information transmitted within, to or from electronic circuits.

## Analog or Continuous Signal

The circuit of Fig. 1 puts out an analog signal or voltage.



**Fig. 1**

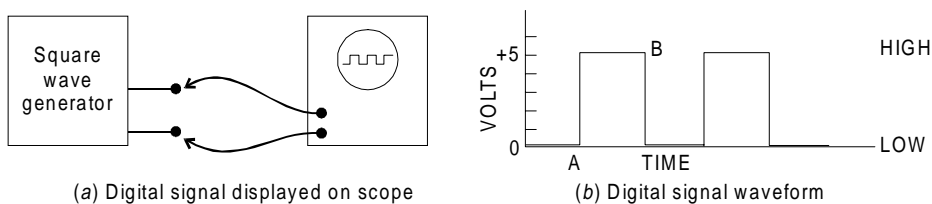
As the wiper on the potentiometer is moved upward, the voltage, from points A to B gradually increases. When the wiper is moved downward, the voltage gradually decreases from 5 to 0 volts (V). The waveform diagram in (b) is a graph of the analog output. On the left side the voltage from A to B is gradually increasing from 0 to 5 V; on the right side the voltage is gradually decreasing from 5 to 0 V. By stopping the potentiometer wiper at any mid-point we can get an output voltage anywhere between 0 to 5 V.

As analog system, then, is one that has a signal which varies continuously in step with the input.

Continuous is defined in many sophisticated ways for a wide variety of reasons and purposes. However, for the purpose here, 'continuous signals or events or processes which change from one condition to yet another condition in such a manner that no detector can perceive any disjoint or quantized behaviour brought about by this change.' For example the temperature of the air around us changes from time to time, and at times it changes quite rapidly, but never does it change so rapidly that some specialized electronic thermometer cannot track its change.

## Digital or Discrete Signal

Fig. 2(a) pictures a square wave generator. The generator produces a square waveform that is displayed on oscilloscope. The digital signal is only at +5 V or at 0 V as diagrammed in 2(b). The voltage at point A moves from 0 to +5 V. The voltage then stays at +5 V for a time. At point B the voltage drops immediately from +5 V to 0 V. The voltage then stays at 0 V for a time. Only two voltages are present in a digital electronic circuit. In the waveform diagram in Fig. 2(b). These voltages are labeled HIGH and LOW. The HIGH voltage is +5 V and the LOW voltage is 0V. Latter we shall call the HIGH voltage (+5 V) a logical 1 and the LOW voltage (0 V) a logical 0.



**Fig. 2**



#### 4 Switching Theory

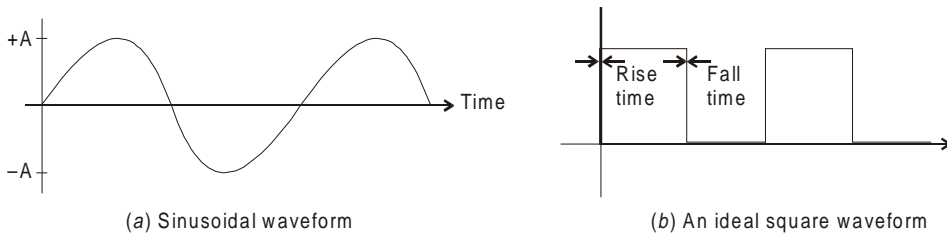
Systems, that handle only HIGH and LOW signals are called digital systems.

Discrete signals or processes are defined as those processes that change from one condition to yet another condition in a perceived disjoint or quantized manner. Two explanations could be that:

- (i) there exists no detector that can trace the quantum transitions in a continuous fashion or, may be,
- (ii) it is best described to be discontinuous for specific reasons.

What is implied here is that there are processes that are continuous in every sense of the word; however their changes are dominated by rapid transitions from one condition to the next. Thus, it makes more sense to define it as a discrete process. For example, consider the signal waveforms shown in Fig. 3. In Fig. 3(a) we have a sinusoidal waveform that is defined by the continuous mathematical expression.

$$V(t) = A \sin 2\pi ft.$$



**Fig. 3**

While in Fig. 3(b) we have an ideal discrete signal, called a square wave signal, that is defined by an infinite series of sinusoidal expressions called a Fourier series. This ideal square wave is characterized by its square corners and infinitely short rise and fall times, and thus is classified distinctly as discrete.

Its changes from one condition (HIGH voltage level) to the other (LOW voltage level) are dominated by a series of rapid transitions.

Thus, it is supposed that some reasonable criteria could be developed for classifying processes and signals as continuous or discrete by determining the time it takes to move from one condition in relation to the time spent in the new condition before the next condition must be moved to.

### 0.2 CLASSIFICATION OF SIGNALS

There are several classes of signals. Here we are considering only the following classes, which are suitable for the scope of this book:

1. Continuous time and discrete time signals.
2. Analog and digital signals.

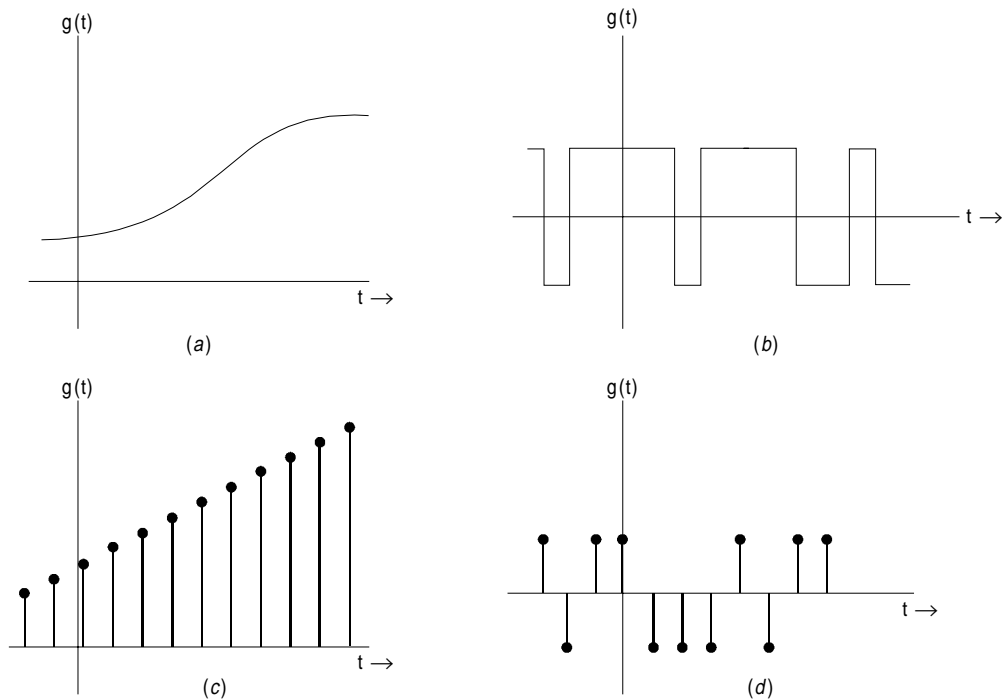
#### 1. Continuous Time and Discrete Time Signals

A signal that is specified for every value of time  $t$  is a continuous time signal (Fig. 4(a) and (b)) and a signal that is specified only at discrete values of  $t$  (Fig. 4(d)) is a discrete time signal. Telephone and video camera outputs are continuous time signals, whereas the monthly sales of a corporation, and stock market daily averages are discrete time signals.

## 2. Analog and Digital Signals

The concept of continuous time is often confused with that of analog. The two are not the same. The same is true of the concepts of discrete and digital. A signal whose amplitude can take on any value in continuous range is an analog signal. This means that an analog signal amplitude can take on an infinite number of values. A digital signal, on the other hand, is one whose amplitude can take on only a finite number of values. Signals associated with a digital computer are digital because they take on only two values (binary signals). For a signal to qualify as digital, the no. of values need not be restricted to two. It can be any finite number. A digital signal whose amplitudes can take on  $M$  values is an  $M$ -ary signal of which binary ( $M = 2$ ) is a special case.

The term continuous time and discrete time qualify the nature of a signal along the time (horizontal) axis. The terms analog and digital on the other hand qualify the nature of the signal amplitude (vertical axis). Figures 4(a, b, c, d) shows examples of various type of signals. It is clear that analog is not necessary continuous time and digital need not be discrete time. Fig. 4(c) shows an example of an analog but discrete time signal.



**Fig. 4.** Examples of signals

- (a) analog, continuous time
- (b) digital, continuous time
- (c) analog, discrete time
- (d) digital, discrete time.

## Why Use Digital Circuits ?

There are several advantages that digital (two-valued discrete) systems have over the analog (continuous) systems. Some commonly named are:

1. Inexpensive ICs can be used with few external components.
2. Information can be stored for short periods or indefinitely.
3. Systems can be programmed and show some manner of “intelligence”.
4. Alphanumeric information can be viewed using a variety of electronic displays.
5. Digital circuits are less affected by unwanted electrical interference called ‘noise’.

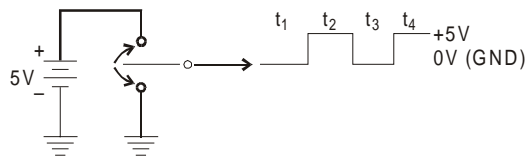
Both digital and analog approaches have pitfalls. However, the pitfalls of digital are at times easier to identify and resolve than the associated pitfalls in the analog world. This advantage as well as those mentioned above, answer much of the question, “why digital?”

## How Digital

The rest of the text is devoted to the fifth of the Kipling’s honest men—How digital. However at this introductory stage, we are giving some idea that how do you generate a digital signal.

Digital signals are composed of two well defined voltage levels. Most of the voltage level used in this class will be about +3 V to +5 V for HIGH and near 0 V (GND) for LOW.

A digital signal could be made manually by using a mechanical switch. Consider the simple circuit shown in Fig. 5.



**Fig. 5**

As the switch is moved up and down, it produces the digital waveform shown at right. At time period  $t_1$ , the voltage is 0V, or LOW. At  $t_2$  the voltage is +5V, or HIGH. At  $t_3$ , the voltage is again 0 V, or LOW, and at  $t_4$ , it is again +5 V, or HIGH. The action of the switch causing the LOW, HIGH, LOW, HIGH waveform is called toggling. By definition, to toggle means to switch over to an opposite state. As an example, in figure, if the switch moves from LOW to HIGH we say the output has toggled. Again if the switch moves from HIGH to LOW we say the output has again toggled.

## Digital and the Real World

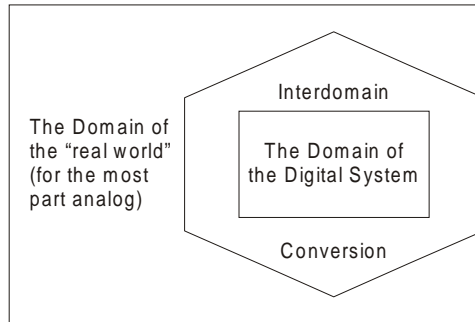
The use of digital practices can be a viable method for solving design problems in the real world. The reason that design problems are emphasized stems from the fact that the major areas of involvement for electrical and digital engineers are:

1. Measurement
2. Control and,
3. Transmission of information and energy.

Thus, if we look at the “what we do”, we find that we are continually trying to find solutions to problems related to the measurement, control, and transmission of information

or energy in the domain of the real world. However, the real world tends to have a continuous nature. Because of this, the discrete domain needs to be buffered in some way.

As a result of this buffering requirement, we should view the digital domain in the perspective shown in Fig. 6.



**Fig. 6**

However, figure does not completely describe the relation between the two domains because there are several important processes in the real world that are at least modelled as discrete processes. For example the electrical signals governing the human nervous system which is, most definitely discrete phenomena. But for the most part, figure does depict the typical relation between the outside world and the digital domain.

The interdomain converter depicted in figure is a specialized system that converts or translates information of one domain into information of another domain. For example, you will serve as a special interdomain converter shortly when you are asked to convert a decimal number into its binary equivalent. This operation we define as an **ANALOG-TO-DIGITAL CONVERSION**.

The pure digital systems are made up of arrays of simple and reliable switches with only two positions, that are either open or closed. (These switches can exist as either mechanical, electromechanical, or electronic devices.)

A numerical system that already existed was adopted to serve as the "tool" needed for utilizing the basic concept. This numerical math system, called the binary system, is based on the two symbols "0" and "1" in contrast to the decimal system which has ten symbols: 0, 1, 2, . . . . , 9.

We should now see that in order to use a digital system, such as a digital computer for mathematical computations, we must first convert our mathematical symbolisms (decimal in this case) into binary symbolisms to allow the computer to perform the mathematical operation. Once this is done, the inverse process must be performed to convert the binary result into a readable decimal representation.

The obvious question: "Is digital worth all of the conversion? The answer can not be simply stated in Yes/No terms, but must be left to the individual and the particular situation. In certain instances, it may not infact be worth the bother. Such would be the case if we were able to create and use multiple valued logic systems to create a "totally decimal machine". Obviously if there were ten unique discriptors usable for our "decimal computer", there would be no need to convert any information into the now required two valued binary system. However, practically speaking, binary systems presently dominate and will continue to be the dominant system for some years to come.

Since, such is the case, and man must learn how to communicate with his machine, it is necessary that we study the processes involved in number conversion and the different codes used to represent and convey information.

## Binary Logic

Binary logic deals with variables that take on two discrete values and with operations that assume logical meaning. The two values the variables take may be called by different names (*e.g.*, true and false, high and low, asserted and not asserted, yes and no etc.), but for our purpose it is convenient to think in terms of numerical values and using the values of 1 and 0. The variables are designated by letters of the alphabet such as A, B, C,  $x$ ,  $y$ ,  $z$ , etc., with each variable having two and only two distinct possible values: 1 and 0. There are three basic logical operations: AND, OR and NOT.

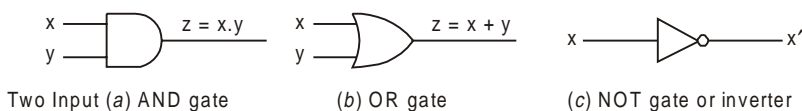
1. **AND:** This operation is represented by a dot or by the absence of an operator. For example  $x.y = z$  or  $xy = z$  is read “ $x$  AND  $y$  is equal to  $z$ ”. The logical operation AND mean that  $z = 1$  if and only if  $x = 1$  if and  $y = 1$ ; otherwise  $z = 0$ .
2. **OR:** This operation is represented by a plus sign. For example  $x + y = z$  is read “ $x$  OR  $y$  is equal to  $z$ ” meaning that  $z = 1$ ,  $x = 1$ , or if  $y = 1$  or if both  $x = 1$  and  $y = 1$ . If both  $x = 0$ , and  $y = 0$ , then  $z = 0$ .
3. **NOT:** This operation is represented by a prime (sometimes by a bar). For example  $x' = z$  (or  $\bar{x} = z$ ) is read “ $x$  not is equal to  $z$ ” meaning that  $z$  is what  $x$  is not. In other words, if  $x = 1$ , then  $z = 0$ ; but if  $x = 0$ , then  $z = 1$ .

Binary logic should not confused with binary arithmetic. (Binary arithmetic will be discussed in Chapter 2). One should realize that an arithmetic variable designates a number that may consist of many digits. A logic variable is always either a 1 or 0. For example, in binary arithmetic we have  $1 + 1 = 10$  (read: “one plus one is equal to 2”), while in binary logic, we have  $1 + 1 = 1$  (read: “one OR one is equal to one”).

For each combination of the values of  $x$  and  $y$ , there is a value of  $z$  specified by the definition of the logical operation. These definitions may be listed in a compact form using ‘truth tables’. A truth table is a table of all possible combinations of the variables showing the relation between the values that the variables may take and the result of the operation. For example, the truth tables for the operations AND and OR with, variables  $x$  and  $y$  are obtained by listing all possible values that the variables may have when combined in pairs. The result of the operation for each combination is then listed in a separate row.

## Logic Gates

Logic circuits that perform the logical operations of AND, OR and NOT are shown with their symbols in Fig. 7. These circuits, called gates, are blocks of hardware that produce a logic 1 or logic 0. Output signal if input logic requirements are satisfied. Not that four different names have been used for the same type of circuits. Digital circuits, switching circuits, logic circuits, and gates. We shall refer to the circuits as AND, OR and NOT gates. The NOT gate is sometimes called an inverter circuit since it inverts a binary signal.



**Fig. 7**

The input signals  $x$  and  $y$  in the two-input gates of Fig. 8 may exist in one of four possible states: 00, 01, 10, or 11. These input signals are shown in Fig. 8 together with the output signals for the AND and OR gates. The timing diagrams of Fig. 8 illustrate the response of each circuit to each of the four possible input binary combinations.

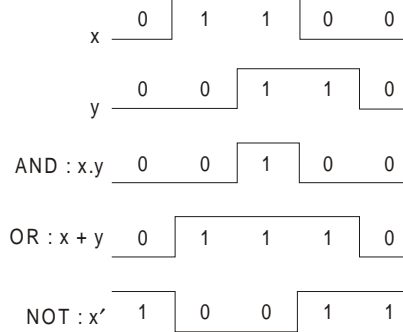


Fig. 8

The mathematic system of binary logic is known as Boolean, or switching algebra. This algebra is used to describe the operation of networks of digital circuits. Boolean algebra is used to transform circuit diagrams to algebraic expressions and vice versa. Chapter 3 is devoted to the study a Boolean algebra where we will see that these function (AND, OR, NOT) Make up a sufficient set to define a two valued Boolean algebra.

The truth tables for AND, OR and NOT are listed in following Table.

AND			OR			NOT	
$x$	$y$	$x \cdot y$	$x$	$y$	$x + y$	$x$	$x'$
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

These tables clearly demonstrate the definitions of the operations.

### Switching Circuits and Binary Signals

The use of binary variables and the application of binary logic are demonstrated by the simple switching circuits of Fig. 9(a) and (b).

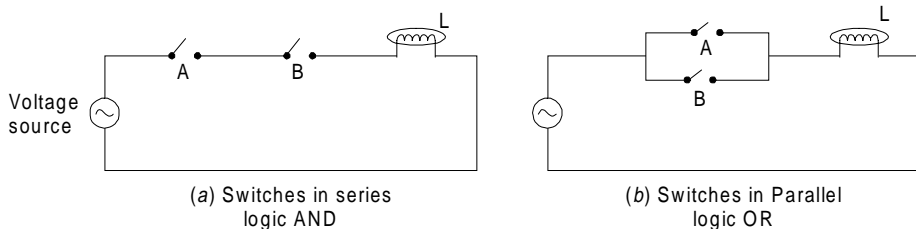


Fig. 9

Let the manual switches A and B represent two binary variables with values equal to 0 when the switch is open and 1 when switch is closed. Similarly, let the lamp L represent a

third binary variable equal to 1 when the light is on and 0 when off. For the switches in series, the light turns on if A and B are closed. For the switches in parallel, this light turns on if A or B is closed.

Thus

$$L = A.B \text{ for the circuit of Fig. 9(a)}$$

$$L = A + B \text{ for the circuit of Fig. 9(b)}$$

Electronic digital circuits are sometimes called switching circuits because they behave like a switch, with the active element such as a transistor either conducting (switch closed) or not conducting (switch open). Instead of changing the switch manually, an electronic switching circuit uses binary signals to control the conduction or non-conduction state of the active element.

### 0.3 INTEGRATED CIRCUITS

An integrated circuit (IC) is a small silicon semiconductor crystal, called a chip, containing electrical components such as transistors, diodes, resistors and capacitors. The various components are interconnected inside the chip to form an electronic circuit. The chip is mounted on a metal or plastic package, and connections are welded to external pins to form the IC.

The individual components in the IC cannot be separated or disconnected and the circuit inside the package is accessible only through the external pins.

Besides a substantial reduction in size, ICs offer several other advantages and benefits compared to electronic circuits with discrete components. These are:

1. The cost of ICs is very low, which makes them economical to use.
2. Their reduced power consumption makes the digital system more economical to operate.
3. They have a high reliability against failure, so the digital system needs less repairs.
4. The operating speed is higher, which makes them suitable for high-speed operations.

Small scale integration (SSI) refers to ICs with fewer than 10 gates on the same chip.

Medium scale integration (MSI) includes 10 to 100 gates per chip. Large scale integration (LSI) refers to more than 100 upto 5000 gates per chip. Very large scale integration (VLSI) devices contain several thousand gates per chip.

Integrated chips circuits are classified into two general categories: (i) Linear and (ii) Digital.

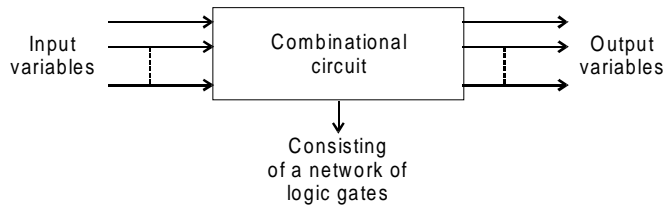
### 0.4 CLASSIFICATION OF DIGITAL CIRCUITS

Digital circuits can be broadly classified into two general categories:

1. Combination logic circuits
2. Sequential logic circuits.

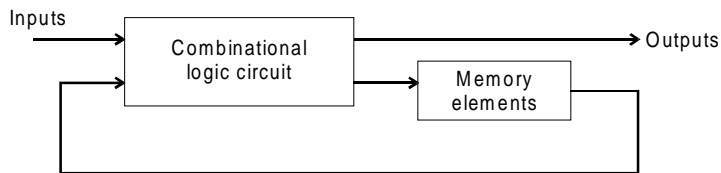
#### 1. Combination Logic Circuits

A combinational circuit consists of logic gates whose outputs at any time are determined directly from the present combination of inputs without regard to previous inputs. A combinational circuit consist of input variables, logic gates and output variables. A block diagram of combinational logic circuit is shown in Fig. 10.

**Fig. 10**

## 2. Sequential Logic Circuits

Many systems encountered in practice also include memory elements which require that the system be described in terms of sequential logic. A block diagram of sequential logic circuit is shown in Fig. 11.

**Fig. 11**

It consists of a combinational circuit to which memory elements are connected to form a feedback path. The memory elements are devices capable of storing binary information within them. Thus the external outputs in a sequential circuit are a function of not only external inputs but also of the present states of memory elements. This is why, these circuits are also known as History sensitive circuits.

There are two types of sequential circuits depending upon the timing of their input signals of the memory elements used.

- (i) **Synchronous sequential circuit.** If the transition of sequential circuit from one state to another are controlled by a clock (*i.e.*, depending upon time), the circuit is known as synchronous. The memory elements are clocked flip flops.
- (ii) **Asynchronous sequential circuit.** When the circuit is not controlled by clock, (*i.e.*, independent of time) the transition from one state to another occur whenever there is a change in the input of circuit. The memory elements are either unclocked FFs (latches) or time delay elements.



# 1

CHAPTER

## NUMBERING SYSTEMS

---

### 1.0 INTRODUCTION

This chapter discusses several important concepts including the binary, octal and hexadecimal numbering systems, binary data organization (bits, nibbles, bytes, words, and double words), signed and unsigned numbering systems. If one is already familiar with these terms he should at least skim this material.

### 1.1 NUMBERING SYSTEMS

Inside today's computers, data is represented as 1's and 0's. These 1's and 0's might be stored magnetically on a disk, or as a state in a transistor, core, or vacuum tube. To perform useful operations on these 1's and 0's one have to organize them together into patterns that make up codes. Modern digital systems do not represent numeric values using the decimal system. Instead, they typically use a binary or two's complement numbering system. To understand the digital system arithmetic, one must understand how digital systems represent numbers.

#### 1.1.1 A Review of the Decimal System

People have been using the decimal (base 10) numbering system for so long that they probably take it for granted. When one see a number like "123", he don't think about the value 123; rather, he generate a mental image of how many items this value represents. In reality, however, the number 123 represents:

$$1*10^2 + 2*10^1 + 3*10^0$$

OR  $100 + 20 + 3$

Each digit appearing to the left of the decimal point represents a value between zero and nine times an increasing power of ten. Digits appearing to the right of the decimal point represent a value between zero and nine times an increasing negative power of ten. For example, the value 123.456 means:

$$1*10^2 + 2*10^1 + 3*10^0 + 4*10^{-1} + 5*10^{-2} + 6*10^{-3}$$

OR  $100 + 20 + 3 + 0.4 + 0.05 + 0.006$

**Note:** Hexadecimal is often abbreviated as *hex* even though, technically speaking, *hex* means base six, not base sixteen.

#### 1.1.2 Binary Numbering System

Most modern digital systems operate using binary logic. The digital systems represents values using two voltage levels (usually 0 v and +5 v). With two such levels one can represent

exactly two different values. These could be any two different values, but by convention we use the values zero and one. These two values, coincidentally, correspond to the two digits used by the binary numbering system.

The binary numbering system works just like the decimal numbering system, with two exceptions: binary only allows the digits 0 and 1 (rather than 0–9), and binary uses powers of two rather than powers of ten. Therefore, it is very easy to convert a binary number to decimal. For each “1” in the binary string, add  $2^n$  where “n” is the bit position in the binary string (0 to  $n-1$  for  $n$  bit binary string).

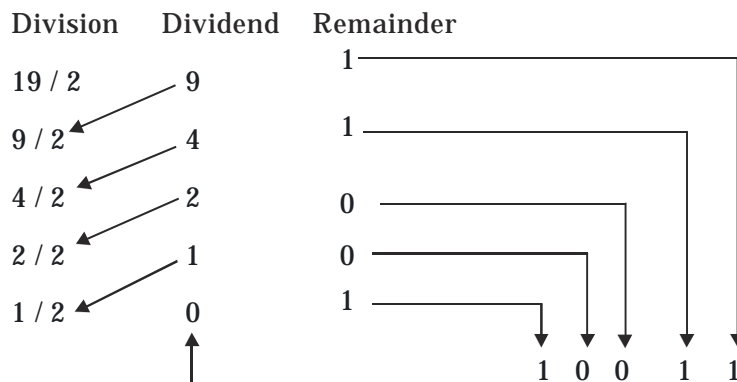
For example, the binary value  $1010_2$  represents the decimal 10 which can be obtained through the procedure shown in the below table:

Binary No.	1	0	1	0
Bit Position (n)	3rd	2nd	1st	0th
Weight Factor ( $2^n$ )	$2^3$	$2^2$	$2^1$	$2^0$
bit * $2^n$	$1*2^3$	$0*2^2$	$1*2^1$	$0*2^0$
Decimal Value	8	0	2	0
Decimal Number	$8 + 0 + 2 + 0 = 10$			

All the steps in above procedure can be summarized in short as

$$1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 = 8 + 0 + 2 + 0 = 10_{10}$$

The inverse problem would be to find out the binary equivalent of given decimal number for instance let us find out binary of  $19_{10}$  (decimal 19)



Dividend is 0, stop the procedure.

Our final number is 10011.

To convert decimal to binary is slightly more difficult. One must find those powers of two which, when added together, produce the decimal result. The easiest method is to work from the a large power of two down to  $2^0$ . For example consider the decimal value 1359:

## 14 Switching Theory

- $2^{10} = 1024$ ,  $2^{11} = 2048$ . So 1024 is the largest power of two less than 1359. Subtract 1024 from 1359 and begin the binary value on the left with a “1” digit. Binary = “1”, Decimal result is  $1359 - 1024 = 335$ .
- The next lower power of two ( $2^9 = 512$ ) is greater than the result from above, so add a “0” to the end of the binary string. Binary = “10”, Decimal result is still 335.
- The next lower power of two is 256 ( $2^8$ ). Subtract this from 335 and add a “1” digit to the end of the binary number. Binary = “101”, Decimal result is 79.
- 128 ( $2^7$ ) is greater than 79, so take a “0” to the end of the binary string. Binary = “1010”, Decimal result remains 79.
- The next lower power of two ( $2^6 = 64$ ) is less than 79, so subtract 64 and append a “1” to the end of the binary string. Binary = “10101”, Decimal result is 15.
- 15 is less than the next power of two ( $2^5 = 32$ ) so simply add a “0” to the end of the binary string. Binary = “101010”, Decimal result is still 15.
- 16 ( $2^4$ ) is greater than the remainder so far, so append a “0” to the end of the binary string. Binary = “1010100”, Decimal result is 15.
- $2^3$  (eight) is less than 15, so stick another “1” digit on the end of the binary string. Binary = “10101001”, Decimal result is 7.
- $2^2$  is less than seven, so subtract four from seven and append another one to the binary string. Binary = “101010011”, decimal result is 3.
- $2^1$  is less than three, so append a one to the end of the binary string and subtract two from the decimal value. Binary = “1010100111”, Decimal result is now 1.
- Finally, the decimal result is one, which is  $2^0$ , so add a final “1” to the end of the binary string. The final binary result is “1010100111”.

### 1.1.3 Binary Formats

In the purest sense, every binary number contains an infinite number of digits (or *bits* which is short for binary digits). Because any number of leading zero bits may precede the binary number without changing its value. For example, one can represent the number seven by:

111      00000111      ..0000000000111      000000000000111

**Note:** This book adopt the convention ignoring any leading zeros. For example,  $101_2$  represents the number five. Since the  $80 \times 86$  works with groups of eight bits, one will find it much easier to zero extend all binary numbers to some multiple of four or eight bits. Therefore, following this convention, number five is represented as  $0101_2$  or  $00000101_2$ .

Often several values are packed together into the same binary number. For convenience, a numeric value is assign to each bit position. Each bit is numbered as follows:

1. The rightmost bit in a binary number is bit position zero.
2. Each bit to the left is given the next successive bit number.

An eight-bit binary value uses bits zero through seven:

$X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0$

A 16-bit binary value uses bit positions zero through fifteen:

$$X_{15} X_{14} X_{13} X_{12} X_{11} X_{10} X_9 X_8 X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0$$

Bit zero is usually referred to as the *low order* bit. The left-most bit is typically called the *high order* bit. The intermediate bits are referred by their respective bit numbers. The low order bit which is  $X_0$  is called LEAST SIGNIFICANT BIT (LSB). The high order bit or left most bit. *i.e.*,  $X_{15}$  is called MOST SIGNIFICANT BIT (MSB).

## 1.2 DATA ORGANIZATION

In pure mathematics a value may take an arbitrary number of bits. Digital systems, on the other hand, generally work with some specific number of bits. Common collections are single bits, groups of four bits (called *nibbles*), groups of eight bits (called *bytes*), groups of 16 bits (called *words*), and more. The sizes are not arbitrary. There is a good reason for these particular values.

### 1.2.1 Bits

The smallest “unit” of data on a binary computer or digital system is a single *bit*. **Bit**, an abbreviation for Binary Digit, can hold either a 0 or a 1. A bit is the smallest unit of information a computer can understand. Since a single bit is capable of representing only two different values (typically zero or one) one may get the impression that there are a very small number of items one can represent with a single bit. That’s not true! There are an infinite number of items one can represent with a single bit.

With a single bit, one can represent any two distinct items. Examples include zero or one, true or false, on or off, male or female, and right or wrong. However, one are *not* limited to representing binary data types (that is, those objects which have only two distinct values). One could use a single bit to represent the numbers 321 and 1234. Or perhaps 6251 and 2. One could also use a single bit to represent the colors green and blue. One could even represent two unrelated objects with a single bit. For example, one could represent the color red and the number 3256 with a single bit. One can represent *any* two different values with a single bit. However, one can represent *only* two different values with a single bit.

To confuse things even more, different bits can represent different things. For example, one bit might be used to represent the values zero and one, while an adjacent bit might be used to represent the values true and false. How can one tell by looking at the bits? The answer, of course, is that one can’t. But this illustrates the whole idea behind computer data structures: *data is what one define it to be*. If one uses a bit to represent a boolean (true/false) value then that bit (by definition) represents true or false. For the bit to have any true meaning, one must be consistent. That is, if one is using a bit to represent true or false at one point in his program, he shouldn’t use the true/false value stored in that bit to represent green or blue later.

Since most items one will be trying to model require more than two different values, single bit values aren’t the most popular data type used. However, since everything else consists of groups of bits, bits will play an important role in programs. Of course, there are several data types that require two distinct values, so it would seem that bits are important by themselves. However, individual bits are difficult to manipulate, so other data types are often used to represent boolean values.

### 1.2.2 Nibbles

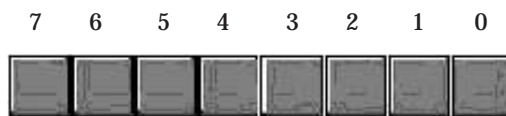
A *nibble* is a collection of four bits. It wouldn’t be a particularly interesting data structure except for two items: BCD (*binary coded decimal*) numbers and hexadecimal numbers. It

takes four bits to represent a single BCD or hexadecimal digit. With a nibble, one can represent up to 16 distinct values. In the case of hexadecimal numbers, the values 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F are represented with four bits (see “The Hexadecimal Numbering System”). BCD uses ten different digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) and requires four bits. In fact, any sixteen distinct values can be represented with a nibble, but hexadecimal and BCD digits are the primary items we can represent with a single nibble.

### 1.2.3 Bytes

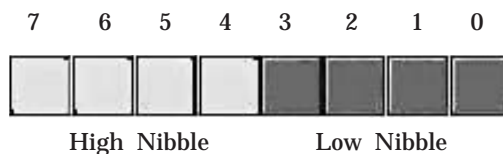
Computer memory must be able to store letters, numbers, and symbols. A single bit by itself cannot be of much use. Bits are combined to represent some meaningful data. A group of eight bits is called a byte. It can represent a character and is the smallest addressable datum (data item) on the most of the digital systems (*e.g.*  $80 \times 86$  microprocessor). The most important data type is the byte. Main memory and input/output addresses on the  $80 \times 86$  are all byte addresses. This means that the smallest item that can be individually accessed by an  $80 \times 86$  program is an eight-bit value. To access anything smaller requires that you read the byte containing the data and mask out the unwanted bits. The bits in a byte are normally numbered from zero to seven using the convention in Fig. 1.1.

Bit 0 is the *low order bit* or *least significant bit*, bit 7 is the *high order bit* or *most significant bit* of the byte. All other bits are referred by their number.



**Fig. 1.1** Bit numbering in a byte

**Note:** That a byte also contains exactly two nibbles (see Fig. 1.2).



**Fig. 1.2** The two nibbles in a byte

Bits 0–3 comprise the *low order nibble*, bits 4–7 form the *high order nibble*. Since a byte contains exactly two nibbles, byte values require two hexadecimal digits.

Since a byte contains eight bits, it can represent  $2^8$ , or 256, different values. Generally, a byte is used to represent numeric values in the range 0.255, signed numbers in the range  $-128.. + 127$  (refer “Signed and Unsigned Numbers”). Many data types have fewer than 256 items so eight bits is usually sufficient.

For a byte addressable machine, it turns out to be more efficient to manipulate a whole byte than an individual bit or nibble. For this reason, most programmers use a whole byte to represent data types that require no more than 256 items, even if fewer than eight bits would suffice. For example, we’ll often represent the boolean values true and false by  $00000001_2$  and  $00000000_2$  (respectively).

Probably the most important use for a byte is holding a character code. Characters typed at the keyboard, displayed on the screen, and printed on the printer all have numeric values.

### 1.2.4 Words

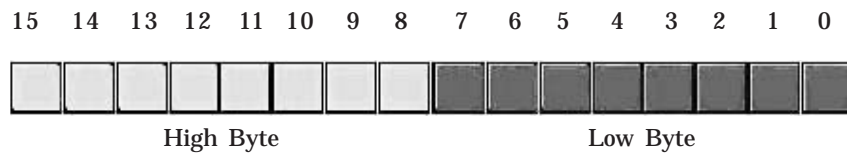
A word is a group of 16 bits. Bits in a word are numbered starting from zero on up to fifteen. The bit numbering appears in Fig. 1.3.



**Fig. 1.3** Bit numbers in a word

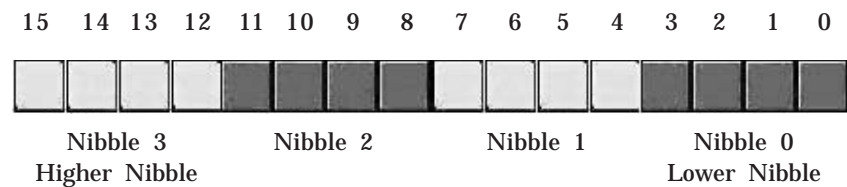
Like the byte, bit 0 is the low order bit and bit 15 is the high order bit. When referencing the other bits in a word use their bit position number.

Notice that a word contains exactly two bytes. Bits 0 through 7 form the low order byte, bits 8 through 15 form the high order byte (see Fig. 1.4).



**Fig. 1.4** The two bytes in a word

Naturally, a word may be further broken down into four nibbles as shown in Fig. 1.5.



**Fig. 1.5** Nibbles in a word

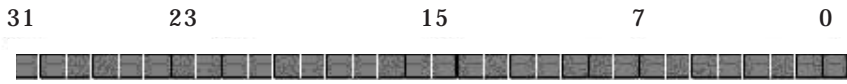
Nibble zero is the low order nibble in the word and nibble three is the high order nibble of the word. The other two nibbles are “nibble one” or “nibble two”.

With 16 bits,  $2^{16}$  (65,536) different values can be represented. These could be the values in the range 0 to 65,535 (or -32,768 to +32,767) or any other data type with no more than 65,536 values. The three major uses for words are integer values, offsets, and segment values.

Words can represent integer values in the range 0 to 65,535 or -32,768 to 32,767. Unsigned numeric values are represented by the binary value corresponding to the bits in the word. Signed numeric values use the two’s complement form for numeric values (refer “Signed and Unsigned Numbers”). Segment values, which are always 16 bits long, constitute the *paragraph address* of a code, data, extra, or stack segment in memory.

### 1.2.5 Double Words

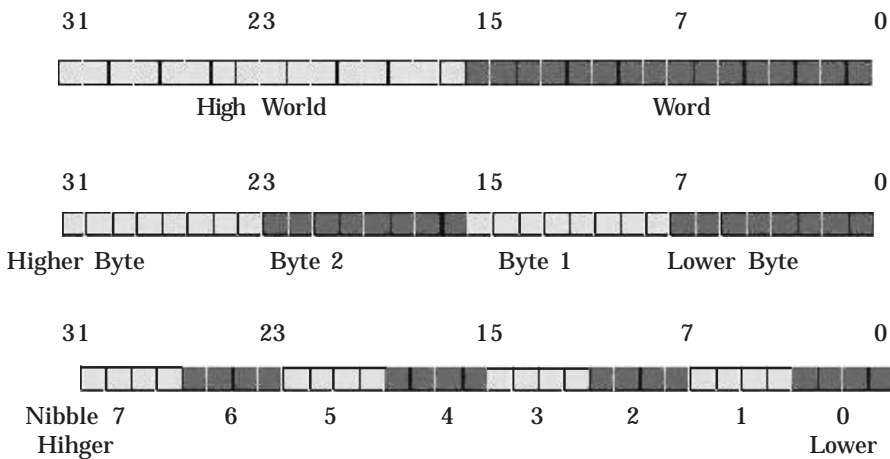
A double word is exactly what its name implies, a pair of words. Therefore, a double word quantity is 32 bits long as shown in Fig. 1.6.



**Fig. 1.6** Bit numbers in a double word

This double word can be divided into a high order word and a low order word, or four different bytes, or eight different nibbles (see Fig. 1.7).

Double words can represent all kinds of different things. First and foremost on the list is a segmented address. Another common item represented with a double word is a 32-bit integer value (which allows unsigned numbers in the range 0 to 4,294,967,295 or signed numbers in the range -2,147,483,648 to 2,147,483,647). 32-bit floating point values also fit into a double word. Most of the time, double words are used to hold segmented addresses.



**Fig. 1.7** Nibbles, bytes, and words in a double word

### 1.3 OCTAL NUMBERING SYSTEM

The octal number system uses base 8 instead of base 10 or base 2. This is sometimes convenient since many computer operations are based on bytes (8 bits). In octal, we have 8 digits at our disposal, 0-7.

Decimal	Octal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	10





To convert back and forth between octal and binary, simply substitute the proper pattern for each octal digit with the corresponding three binary digits.

For example, 372 in octal becomes 010 111 010 or 010111010 in binary.

777 in octal becomes 111 111 111 or 111111111 in binary.

The same applies in the other direction:

100111010 in binary becomes 100 111 010 or 472 in octal.

Since it is so easy to convert back and forth between octal and binary, octal is sometimes used to represent binary codes. Octal is most useful if the binary code happens to be a multiple of 3 bits long. Sometimes it is quicker to convert decimal to binary by first converting decimal to octal, and then octal to binary.

#### 1.4 HEXADECIMAL NUMBERING SYSTEM

The hexadecimal numbering system is the most common system seen today in representing raw computer data. This is because it is very convenient to represent groups of 4 bits. Consequently, one byte (8 bits) can be represented by two groups of four bits easily in hexadecimal.

Hexadecimal uses a base 16 numbering system. This means that we have 16 symbols to use for digits. Consequently, we must invent new digits beyond 9. The digits used in hex are the letters A, B, C, D, E, and F. If we start counting, we get the table below:

<b>Decimal</b>	<b>Hexadecimal</b>	<b>Binary</b>
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111
16	10	10000
17	11	10001
18 ...		



**Example.** Convert  $(1A3)_{16}$  into octal.

**Solution.**

1. Converting hex to binary

$$(1A3)_{16} = \underbrace{0001}_1 \underbrace{1010}_A \underbrace{0011}_3$$

2. Grouping of 3-bits

$$(1A3)_{16} = \begin{array}{cccc} \underline{000} & \underline{110} & \underline{100} & \underline{011} \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 0 & 6 & 4 & 3 \end{array}$$

so

$$(1A3)_{16} = (0643)_8 \equiv (643)_8$$

### Octal to Hex Conversion

1. Convert the given octal number into binary.
2. Starting from right make groups of 4-bits and designate each group as a Hexadecimal digit.

**Example.** Convert  $(76)_8$  into hexadecimal.

**Solution.** 1. Converting octal to binary

$$(76)_8 = \underbrace{111}_7 \underbrace{110}_6$$

2. Grouping of 4-bits

$$(76)_8 = \begin{array}{cc} \underline{11} & \underline{1110} \\ \downarrow & \downarrow \\ 3 & E \end{array} \equiv \begin{array}{cc} \underline{0011} & \underline{1110} \\ \downarrow & \downarrow \\ 3 & E \end{array}$$

so

$$(76)_8 = (3E)_{16}$$

### 1.5 RANGE OF NUMBER REPRESENTATION

The range of numbers that can be represented is determined by the number of digits (or bits in binary) used to represent a number. Let us consider decimal number system to understand the idea.

Highest decimal number represented by 2 digits = 99

But  $99 = 100 - 1 = 10^2 - 1$ . The power of 10 (in  $10^2 - 1$ ) indicates that it is 2 digit representation.

So highest 2-digit decimal number =  $10^2 - 1$

and lowest 2-digit decimal number = 00

Thus range of 2-digit decimal number = 00 to  $10^2 - 1$

It is evident that a total of 100 or  $10^2$  numbers (00 to 99) can be represented by 2-digits.

So we conclude that for n-digit representation

$$\text{range of decimal numbers} = 0 \text{ to } 10^n - 1$$

highest decimal number =  $10^n - 1$

total numbers that can be represented =  $10^n$

Note that highest  $n$ -digit decimal number can be represented by  $n$  9s (i.e.,  $10 - 1$ ) e.g., highest 2 digit decimal number is represented by 2 9s which is 99.

The above discussion can be generalized by taking base- $r$  number system instead of base-10 (or decimal) number system. Thus with  $n$ -digit representation–

Total distinct numbers that can be represented =  $r^n$

Highest decimal Number =  $r^n - 1$

Range of Numbers = 0 to  $r^n - 1$

where  $r$  = base or radix of Number system

$n$  = Number of digits used for representation

It is worth noting that highest decimal number can be represented by  $n(r - 1)$ s in base- $r$  system.

Let us consider the base-2 or binary number system. Thus  $2^n$  distinct quantities, in the range 0 to  $2^n - 1$ , can be represented with  $n$ -bit. If  $n = 4$ -bits, total distinct quantities (i.e., numbers) that can be represented

$$= N = 2^4 = 16$$

the range of numbers = 0 to  $2^4 - 1 = 0$  to 15

and Highest decimal number =  $2^4 - 1 = 15$

The highest decimal number 15, is represented by our 1s i.e., 1111. The range 0 to 15 corresponds to 0000 to 1111 in binary.

If we want to represent a decimal number  $M$  using  $n$ -bits, then the number  $M$  should lie in the range 0 to  $2^n - 1$  i.e.,

$$0 \leq M \leq 2^n - 1$$

or  $2^n - 1 \geq M$

or  $2^n \geq M + 1$

or  $n \geq \log_2 (M + 1)$

where  $M$  and  $n$  are integers.

In the similar way if we want to represent  $N$  distinct quantities in binary then  $N$  should not exceed  $2^n$ .

or  $2^n \geq N$

or  $n \geq \log_2 N$  Both  $n$  and  $N$  are integer

**Example.** How many bits are required to represent

(i) 16-distinct levels

(ii) 10 distinct levels

(iii) 32 distinct levels

**Solution.** (i) we have  $2^n \geq N$

or  $2^n \geq 16 \Rightarrow 2^n \geq 2^4$

or  $n \geq 4 \Rightarrow n = 4$

Thus, atleast 4-bits are required to represent 16 distinct levels, ranging from 0 to 15.

(i) We have  $n \geq \log_2 N$

or  $n \geq \log_2 10 \Rightarrow n \geq 3.32$

but  $n$  should be integer, so take next higher integer value

*i.e.*,  $n = 4$  bits

So, minimum 4-bits are required to represent 10 distinct levels, ranging from 0 to 9.

(iii)  $n \geq \log_2 N$

or  $n \geq \log_2 32 \Rightarrow n \geq \log_2 2^5$

or  $n \geq 5 \Rightarrow n = 5$

So, minimum 5-bits are required to represent 32 levels, ranging from 0 to 31.

**Example.** Calculate the minimum no. of bits required to represent decimal numbers

(i) 16

(ii) 63

**Solution.** (i) We have  $n \geq \log_2(M + 1)$  where  $M =$  given number

so  $n \geq \log_2(16 + 1) \Rightarrow n \geq \log_2(17)$

or  $n \geq 4.09$

taking next higher integer *i.e.*,  $n = 5$  bits.

Thus, atleast 5-bits are required to represent decimal number 16.

(ii)  $n \geq \log_2 (M + 1)$

$n \geq \log_2 (63 + 1) \Rightarrow n \geq \log_2 64$

or  $n \geq \log_2 2^6$  or  $n \geq 6$  bits

So, minimum 6-bits are needed to represent decimal 63.

**Example.** In a base-5 number system, 3 digit representation is used. Find out

(i) Number of distinct quantities that can be represented.

(ii) Representation of highest decimal number in base-5.

**Solution.** Given radix of number system  $r = 5$

digits of representation  $n = 3$

digits in base-5 would be  $-0, 1, 2, 3, 4$

(i) we have relation

$$\begin{aligned} \text{no of distinct quantities} &= r^n \\ &= 5^3 = 125 \end{aligned}$$

So, 125 distinct levels (quantities) can be represented.

(ii) Highest decimal Number can be represented by  $n(r - 1)$ s *i.e.*, by three 4s.

So, highest decimal Number = 444

## 1.6 BINARY ARITHMETIC

The binary arithmetic operations such as addition, subtraction, multiplication and division are similar to the decimal number system. Binary arithmetics are simpler than decimal because they involve only two digits (bits) 1 and 0.

### Binary Addition

Rules for binary addition are summarized in the table shown in Fig. 1.8.

<i>Augend</i>	<i>Addend</i>	<i>Sum</i>	<i>Carry</i>	<i>Result</i>
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	10

**Fig. 1.8** Rules for binary addition

As shown in 4th row adding 1 to 1 gives 9 carry which, is given to next binary position, similar to decimal system. This is explained in examples below:

**Example.** (i) Add 1010 and 0011 (ii) Add 0101 and 1111

**Solution.**

$$\begin{array}{r}
 \phantom{1} \leftarrow \text{Carry} \quad 1\ 1\ 1 \leftarrow \text{Carry} \\
 1\ 0\ 1\ 0 \\
 + 0\ 0\ 1\ 1 \\
 \hline
 1\ 1\ 0\ 1 \\
 \\
 \phantom{1} \leftarrow \text{Carry} \quad 1\ 1\ 1 \leftarrow \text{Carry} \\
 0\ 1\ 0\ 1 \\
 + 1\ 1\ 1\ 1 \\
 \hline
 1\ 0\ 1\ 0\ 0 \\
 \phantom{1} \uparrow \\
 \phantom{1} \text{Carry}
 \end{array}$$

### Binary Subtraction

The rules for binary subtraction is summarized in the table shown in Fig. 1.9.

<i>Minuend</i>	<i>Subtrahend</i>	<i>Difference</i>	<i>Borrow</i>
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

**Fig. 1.9** Rules for binary subtraction

The process of subtraction is very similar to decimal system in which if a borrow is needed it is taken from next higher binary position, as shown in row 2.

**Example.** Subtract 0100 from 1011

**Solution.**

$$\begin{array}{r}
 \phantom{1} \leftarrow \text{Borrow} \\
 1\ 0\ 1\ 1 \leftarrow \text{Minuend} \\
 - 0\ 1\ 0\ 0 \leftarrow \text{Subtrahend} \\
 \hline
 0\ 1\ 1\ 1 \leftarrow \text{Difference} \\
 \phantom{1} \uparrow \phantom{1} \uparrow \phantom{1} \uparrow \phantom{1} \uparrow \\
 C_3\ C_2\ C_1\ C_0
 \end{array}$$

There is no problem in column  $C_0$  and  $C_1$ . In column  $C_2$  we made  $0 - 1$ , so result = 1 and borrow = 1. Then this borrow = 1 is marked in column  $C_3$ . So result in column  $C_2$  is 1. Then in column  $C_3$  we first made  $1 - 0$  to get result = 1 and then we subtracted borrow from result, thus we get 0 in column  $C_3$ .

“Thus in subtraction, first subtract the subtrahend bit from minuend and then subtract borrow from the result.”

Watch out the next example to further clarify the concept.

**Example.** Subtract 0110 from 1001

<b>Solution.</b>	1	1			← Borrow
	1	0	0	1	← Minuend
	-0	1	1	0	← Subtrahend
	0	0	1	1	← Difference
	↑	↑	↑	↑	
	C <sub>3</sub>	C <sub>2</sub>	C <sub>1</sub>	C <sub>0</sub>	

Here, in column C<sub>1</sub> we get difference = 1 and borrow = 1. This borrow is marked in column C<sub>2</sub>, and difference = 1 is shown in the column C<sub>1</sub>. We now come to column C<sub>2</sub>. Here by 0-1 we get difference = 1 and borrow = 1. Now this borrow is marked in column C<sub>3</sub>. But in column C<sub>2</sub> already we have 9 borrow so this borrow = 1 is subtracted from difference = 1 which results in 0. Thus the difference = 0 is marked in column C<sub>2</sub>.

In the similar way we process column C<sub>3</sub> and we get difference = 0 in column C<sub>3</sub>.

### Binary Multiplication

Binary multiplication is also similar to decimal multiplication. In binary multiplication if multiplier bit is 0 then partial product is all 0 and if multiplier bit is 1 then partial product is 1. The same is illustrated in example below:

**Example.**

1 0 0 1	←	MULTIPLICAND
1 0 1	←	MULTIPLIER
1 0 0 1	←	Partial Product when multiplier bit = 1
0 0 0 0	←	Partial Product when multiplier bit = 0
1 0 0 1	←	Partial Product when multiplier bit = 0
1 0 1 1 0 1	←	FINAL PRODUCT

### Binary Division

Binary division is also similar to decimal division as illustrated in example below:

**Example.**

		1 0 1		
Divisor →	1 0 0 1	) 1 0 1 1 0 1	←	Dividend
		1 0 0 1		
		× × 1 0 0 1		
		1 0 0 1		
		× × × ×		

## 1.7 NEGATIVE NUMBERS AND THEIR ARITHMETIC

So far we have discussed straight forward number representation which are nothing but positive number. The negative numbers have got two representation

(j) complement representation.

(ii) sign magnitude representation.

We will discuss both the representation in following subsections.

### 1.7.1 1's and 2's Complement

These are the complements used for binary numbers. Their representation are very important as digital systems work on binary numbers only.

#### 1's Complement

1's complement of a binary number is obtained simply by replacing each 1 by 0 and each 0 by 1. Alternately, 1's complement of a binary can be obtained by subtracting each bit from 1.

**Example.** Find 1's complement of (i) 011001 (ii) 00100111

**Solution.** (i) Replace each 1 by 0 and each 0 by 1

$$\begin{array}{cccccc} 0 & 1 & 1 & 0 & 0 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 0 & 1 & 1 & 0 \end{array}$$

So, 1's complement of 011001 is 100110.

(ii) Subtract each binary bit from 1.

$$\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ \hline 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{array} \leftarrow \text{1's complement}$$

one can see that both the method gives same result.

#### 2's Complement

2's complement of a binary number can be obtained by adding 1 to its 1's complement.

**Example.** Find 2's complement of (i) 011001 (ii) 0101100

**Solution.** (i)

$$\begin{array}{cccccc} 0 & 1 & 1 & 0 & 0 & 1 & \leftarrow \text{Number} \\ 1 & 0 & 0 & 1 & 1 & 0 & \leftarrow \text{1's complement} \\ & & & & + & 1 & \leftarrow \text{Add 1 to 1's complement} \\ \hline 1 & 0 & 0 & 1 & 1 & 1 & \leftarrow \text{2's complement} \end{array}$$

(ii)

$$\begin{array}{cccccc} 0 & 1 & 0 & 1 & 1 & 0 & 0 & \leftarrow \text{Number} \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & \leftarrow \text{1's complement} \\ & & & & + & 1 & \leftarrow \text{Add 1 to 1's complement} \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 0 & \leftarrow \text{2's complement} \end{array}$$

There is an efficient method to find 2's complement based upon the observation made on the above 2 examples. Consider the number and its 2's complement of example (ii) as shown below:

$$\begin{array}{cccc|cccc} 0 & 1 & 0 & 1 & 1 & 0 & 0 & \leftarrow \text{Number} \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & \leftarrow \text{2's Complement} \\ \leftarrow & & & & \leftarrow & & & \\ & 1's & & & \text{Same as} & & & \\ & \text{complement} & & & \text{number} & & & \end{array}$$

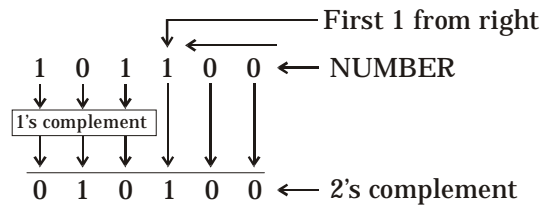
**Fig. 1.10** Number and its 2's complement



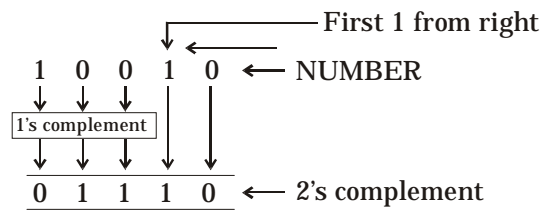
The above figure clearly shows that to find 2's complement of a binary number start from right towards left till the first 1 appears in the number. Take these bits (including first 1) as it is and take 1's complement of rest of the bits. Workout below examples to enhance your understanding.

**Example.** Find 2's complement of (i) 101100 (ii) 10010 (iii) 01001

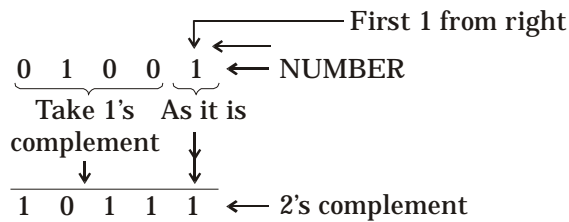
**Solution.** (i) Number = 101100



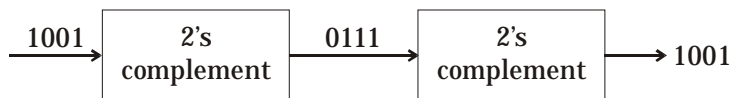
(ii) Number = 10010



(iii) Number = 01001



It is interesting to note that taking complement twice leaves the number as it is. This is illustrated in below Fig. 1.11.



**Fig. 1.11** Effect of taking complement twice

To represent a negative number using complements the process involves two steps.

- (1) obtain the binary representation of equivalent positive number for given negative number. *e.g.*, if given number is  $-2$  then obtain binary representation of  $+2$ .
- (2) Take the appropriate complement of representation obtained in step 1.

**Example.** Obtain 1's and 2's complement representation of  $-5$  and  $-7$ .

**Solution.** (i)  $-5$

1. binary of  $+5 = (0101)_2$
2. 1's complement of  $(0101)_2 = (1010)_2 \leftarrow$  Represents  $(-5)_{10}$



**Step 1.** 1's complement of 0111 = 1000

**Step 2.** Perform addition of minuend and 1's complement of subtrahend

$$\begin{array}{r} 0 \ 0 \ 1 \ 1 \ \leftarrow (3) \\ 1 \ 0 \ 0 \ 0 \ \leftarrow (-7 \text{ or } 1\text{'s complement of } +7) \\ \hline \text{Result} = \underline{1 \ 0 \ 1 \ 1} \end{array}$$

**Step 3.** No carry produced so no EAC operation.

**Step 4.** Since no carry produced in step 2, result is negative and is in complemented form. So we must take 1's complement of result to find correct magnitude of result.

1's complement of result  $(1011)_2 = (0100)_2$

so final result =  $-(0100)_2$  or  $-(4)_{10}$

Note that when (in example (ii)) the result was negative (step 2), MSB of the result was 1. When (in example (i)) the result was positive the MSB was 0. The same can be observed in 2's complement subtraction.

**2's complement Subtraction** Method of 2's complement is similar to 1's complement subtraction except the end around carry (EAC). The rules are listed below:

1. Take 2's complement of subtrahend.
2. Add 2's complement of subtrahend to minuend.
3. If a carry is produced, then discard the carry and the result is positive. If no carry is produced result is negative and is in 2's complement form.

**Example.** Perform following subtraction using 2's complement.

<i>(i)</i> 7 - 5	<i>(ii)</i> 5 - 7	}	both the numbers should have equal length
<b>Solution.</b> <i>(i)</i> 7 - 5:	binary of 7 = $(0111)_2$		
	binary of 5 = $(0101)_2$		

**Step 1.** 2's complement of subtrahend  $(=0101)_2 = (1011)_2$

**Step 2.** Perform addition of minuend and 2's complement of subtrahend

$$\begin{array}{r} \phantom{\text{Final}} \\ \phantom{\text{Carry}} \longrightarrow \phantom{1} \\ \phantom{\uparrow} \\ 0 \ 1 \ 1 \ 1 \ \leftarrow (7) \\ + \ 1 \ 0 \ 1 \ 1 \ \leftarrow (-5 \text{ or } 2\text{'s complement of } +5) \\ \hline \text{Final} \longrightarrow 1 \ 0 \ 0 \ 1 \ 0 \\ \text{Carry} \phantom{\longrightarrow} \phantom{1} \\ \phantom{\uparrow} \\ \text{Discard the final carry} \end{array}$$

**Step 3.** Since a final carry is produced in step 2 (which is discarded) the result is positive. So,

result =  $(0010)_2 = (2)_{10}$

*(ii)* 5 - 7:

binary of 5 =  $(0101)_2$   
 binary of 7 =  $(0111)_2$

**Step 1.** 2's complement of subtrahend  $(= 0111) = 1001$

**Step 2.** Addition of minuend and 2's complement of subtrahend

$$\begin{array}{r}
 0\ 1\ 0\ 1 \leftarrow 5 \\
 1\ 0\ 0\ 1 \leftarrow (-7 \text{ or } 2\text{'s complement of } +7) \\
 \hline
 1\ 1\ 1\ 0 \leftarrow \text{Result} \\
 \uparrow \\
 \text{No final carry}
 \end{array}$$

**Step 3.** Since final carry is not generated in step 2, the result is negative and is in 2's complement form. So we must take 2's complement of result obtained in step 2 to find correct magnitude of result.

$$\begin{aligned}
 2\text{'s complement of result } (1110)_2 &= (0010)_2 \\
 \text{so, final result} &= -(0010)_2 = -(2)_{10}
 \end{aligned}$$

### 1.7.3 Signed Binary Representation

Untill now we have discussed representation of unsigned (or positive) numbers, except one or two places. In computer systems sign (+ve or -ve) of a number should also be represented by binary bits.

The accepted convention is to use 1 for negative sign and 0 for positive sign. In signed representation MSB of the given binary string represents the sign of the number, in all types of representation. We have two types of signed representation:

1. Signed Magnitude Representation
2. Signed Complement Representation

In a **signed-Magnitude** representation, the MSB represent the sign and rest of the bits represent the magnitude. *e.g.*,

$$\begin{array}{ccc}
 +5 = (0\ 1\ 0\ 1)_2 & & -5 = (1\ 1\ 0\ 1)_2 \\
 \begin{array}{c} \uparrow \\ \text{+ sign} \end{array} & \begin{array}{c} \uparrow \\ \text{Magnitude} \end{array} & \begin{array}{c} \uparrow \\ \text{- sign} \end{array} & \begin{array}{c} \uparrow \\ \text{Magnitude} \end{array}
 \end{array}$$

Note that positive number is represented similar to unsigned number. From the example it is also evident that out of 4-bits, only 3-bits are used to represent the magnitude. Thus in general,  $n - 1$  bits are used to denote the magnitude. So the range of signed representation becomes  $-(2^{n-1} - 1)$  to  $(2^{n-1} - 1)$ .

In a **signed-complement** representation the positive numbers are represented in true binary form with MSB as 0. Where as the negative numbers are represented by taking appropriate complement of equivalent positive number, including the sign bit. Both 1's and 2's complements can be used for this purpose *e.g.*,

$$\begin{aligned}
 +5 &= (0101)_2 \\
 -5 &= (1010)_2 \leftarrow \text{in } 1\text{'s complement} \\
 &= (1011)_2 \leftarrow \text{in } 2\text{'s complement}
 \end{aligned}$$

Note that in signed complement representation the fact remains same that  $n - 1$  bits are used for magnitude. The range of numbers

In 1's complement	0 to $(2^{n-1} - 1)$	Positive Numbers
	- 0 to $-(2^{n-1} - 1)$	Negative Numbers
In 2's complement	0 to $(2^{n-1} - 1)$	Positive Numbers
	- 1 to $-2^{n-1}$	Negative Numbers

To illustrate the effect of these 3 representations, we consider 4-bit binary representation and draw the below table. Carefully observe the differences in three methods.

Decimal	Signed Magnitude	1's complement	2's complement
+0	0 0 0 0	0 0 0 0	0 0 0 0
+1	0 0 0 1	0 0 0 1	0 0 0 1
+2	0 0 1 0	0 0 1 0	0 0 1 0
+3	0 0 1 1	0 0 1 1	0 0 1 1
+4	0 1 0 0	0 1 0 0	0 1 0 0
+5	0 1 0 1	0 1 0 1	0 1 0 1
+6	0 1 1 0	0 1 1 0	0 1 1 0
+7	0 1 1 1	0 1 1 1	0 1 1 1
-8	—	—	1 0 0 0
-7	1 1 1 1	1 0 0 0	1 0 0 1
-6	1 1 1 0	1 0 0 1	1 0 1 0
-5	1 1 0 1	1 0 1 0	1 0 1 1
-4	1 1 0 0	1 0 1 1	1 1 0 0
-3	1 0 1 1	1 1 0 0	1 1 0 1
-2	1 0 1 0	1 1 0 1	1 1 1 0
-1	1 0 0 1	1 1 1 0	1 1 1 1
-0	1 0 0 0	1 1 1 1	—

**Fig. 1.12** Different signed representation

From the table it is evident that both signed Magnitude and 1's complement methods introduce two zeros +0 and -0 which is awkward. This is not the case with 2's complement. This is one among the reasons that why all the modern digital systems use 2's complement method for the purpose of signed representation. From the above table it is also evident that

in signed representation  $\frac{2^n}{2}$  positive numbers and  $\frac{2^n}{2}$  negative numbers can be represented

with  $n$ -bits. Out of  $2^n$  combinations of  $n$ -bits, first  $\frac{2^n}{2}$  combinations are used to denote the

positive numbers and next  $\frac{2^n}{2}$  combinations represent the negative numbers.

**Example.** In a signed representation given binary string is  $(11101)_2$ . What will be the sign and magnitude of the number represented by this string in signed magnitude, 1's complement and 2's complement representation.

**Solution.**

The number  $N = (11101)_2$

since MSB = 1 the given number is negative.

(i) In signed Magnitude MSB denotes sign and rest of the bits represent magnitude. So,

$$\begin{array}{cccc} \uparrow & 1 & 1 & 0 & 1 \\ & \text{sign} & \text{Magnitude} & & \end{array} \Big)_2 = -13$$

(ii) In 1's complement if number is negative (i.e., MSB = 1) then the magnitude is obtained by taking 1's complement of given number.

$$\begin{aligned} \text{1's complement of } (11101)_2 &= (00010)_2 \\ \text{so } (11101)_2 &= -2 \text{ in 1's complement.} \end{aligned}$$

(iii) In 2's complement if number is negative (i.e., MSB = 1) then magnitude is obtained by taking 2's complement of given number.

$$\begin{aligned} \text{2's complement of } (11101)_2 &= (00011)_2 \\ &= 3 \\ \text{so } (11101)_2 &= -3 \text{ in 2's complement.} \end{aligned}$$

**Example.** Obtain an 8-bit representation of -9 in signed Magnitude, 1's complement and 2's complement representation.

**Solution.** We first find binary of 9 i.e.,  $(9)_{10} = (1001)_2$   
 Next we represent 9 using 8-bits. So  $N = (00001001)_2$   
 $= (9)_{10}$

(i) In signed Magnitude, MSB shows sign and rest of the bits shows true magnitude. So,  
 $(-9)_{10} = (10001001)_2$

(ii) In 1's complement, negative number is represented by taking 1's complement of positive number. So,

$$\begin{aligned} (-9)_{10} &= \text{1's complement of } (00001001)_2 \\ &= (11110110)_2 \end{aligned}$$

(iii) In 2's complement

$$\begin{aligned} (-9)_{10} &= \text{2's complement of } (00001001)_2 \\ &= (11110111)_2 \end{aligned}$$

### 1.7.4 Arithmetic Overflow

When the result of an arithmetic operation requires  $n+1$  bits, upon operating on  $n$ -bits number, an overflow occurs. Alternately, if result exceeds the range 0 to  $2^n - 1$ , an overflow occurs.

Let us consider the addition of two 4-bit numbers

$$\begin{array}{r} 9 \rightarrow \quad 1 \ 0 \ 0 \ 1 \\ +8 \rightarrow \quad 1 \ 0 \ 0 \ 0 \\ \hline 17 \rightarrow 1 \ 0 \ 0 \ 0 \ 1 \end{array}$$

Thus addition of two 4-bits numbers requires 5-bits ( $n+1$  bits) to represent the sum. Alternately, the result of addition of 4-bits, falls outside the range 0 to 15 (i.e., 0 to  $2^4-1$ ). Thus, overflow has occurred.

In case of **signed arithmetic** the overflow causes the sign bit of the answer to change. In this case an overflow occurs if the result does not lie in the range  $-2^{n-1}$  to  $2^{n-1} - 1$ . In signed arithmetic overflow can occur only when two positive numbers or two negative numbers are added.

Let us consider 4-bit signed 2's complement representation.

1. Addition of two positive numbers +6 and +5

$$\begin{array}{r} +6 \rightarrow 0 \ 1 \ 1 \ 0 \\ + \quad +5 \rightarrow 0 \ 1 \ 0 \ 1 \\ \hline +11 \quad 1 \ 0 \ 1 \ 1 \end{array}$$

Since MSB of result is 1, it reflects a negative result which is incorrect. It happened because overflow has changed the sign of result.

2. Addition of two negative numbers -6 and -5

$$\begin{array}{r} -6 \rightarrow \quad 1 \ 0 \ 1 \ 0 \leftarrow 2\text{'s complement of } 6 \\ -5 \rightarrow \quad 1 \ 0 \ 1 \ 1 \leftarrow 2\text{'s complement of } 5 \\ \hline -11 \quad \quad \quad 1 \ 0 \ 1 \ 0 \ 1 \\ \quad \quad \quad \uparrow \\ \quad \quad \quad \text{Carry} \end{array}$$

In 2's complement if a carry is generated after the addition then carry is discarded and result is declared positive. Thus, result =  $(0101)_2 = +5$  which is wrong, because addition of two negative numbers should give a negative result. This happened due to overflow.

Note that overflow is a problem that occurs when result of an operation exceeds the capacity of storage device. In a computer system the programmer must check the overflow after each arithmetic operation.

### 1.7.5 9's and 10's Complement

9's and 10's complements are the methods used for the representation of decimal numbers. They are identical to the 1's and 2's complements used for binary numbers.

**9's complement:** 9's complement of a decimal number is defined as  $(10^n - 1) - N$ , where  $n$  is no. of digits and  $N$  is given decimal numbers. Alternately, 9's complement of a decimal number can be obtained by subtracting each digit from 9. **9's complement of  $N = (10^n - 1) - N$ .**

**Example.** Find out the 9's complement of following decimal numbers.

(i) 459

(ii) 36

(iii) 1697

**Solution.** (i) By using  $(10^n - 1) - N$ ; But,  $n = 3$  in this case

So,  $(10^3 - 1) - N = (10^3 - 1) - 459 = 540$

Thus 9's complement of 459 = 540

(ii) By subtracting each digit from 9

$$\begin{array}{r} 9 \ 9 \\ -3 \ 6 \\ \hline 6 \ 3 \end{array}$$

So, 9's complement of 36 is 63.

(iii) We have  $N = 1697$ , so  $n = 4$   
 Thus,  $10^n - 1 = 10^4 - 1 = 9999$   
 So,  $(10^n - 1) - N = (10^4 - 1) - 1697 = 9999 - 1697$   
 $= 8302$

Thus, 9's complement of 1697 = 8302

**10's complement:** 10's complement of a decimal number is defined as  $10^n - N$ .

$$\boxed{\text{10's complement of } N = 10^n - N}$$

but  $10^n - N = (10^n - 1) - N + 1$   
 $= 9\text{'s complement of } N + 1$

Thus, 10's complement of a decimal number can also be obtained by adding 1 to its 9's complement.

**Example.** Find out the 10's complement of following decimal numbers. (i) 459 (ii) 36.

**Solution.** (i) By using  $10^n - N$ ; We have  $N = 459$  so  $n = 3$

So,  $10^n - N = 10^3 - 459 = 541$

**So, 10's is complement of 459 = 541**

(ii) By adding 1 to 9's complement

$$\begin{aligned} 9\text{'s complement of } 36 &= 99 - 36 \\ &= 63 \end{aligned}$$

$$\begin{aligned} \text{Hence, 10's complement of } 36 &= 63 + 1 \\ &= 64 \end{aligned}$$

### 1.7.6 $r$ 's Complement and $(r - 1)$ 's Complement

The  $r$ 's and  $(r - 1)$ 's complements are generalized representation of the complements, we have studied in previous subsections.  $r$  stands for radix or base of the number system, thus  $r$ 's complement is referred as *radix complement* and  $(r - 1)$ 's complement is referred as *diminished radix complement*. Examples of  $r$ 's complements are 2's complement and 10's complement. Examples of  $(r - 1)$ 's complement are 1's complement and 9's complement.

In a base- $r$  system, the  $r$ 's and  $(r - 1)$ 's complement of the number  $N$  having  $n$  digits, can be defined as:

$$\boxed{(r - 1)\text{'s complement of } N = (r^n - 1) - N}$$

and

$$\boxed{\begin{aligned} r\text{'s complement of } N &= r^n - N \\ &= (r - 1)\text{'s complement of } N + 1 \end{aligned}}$$

The  $(r - 1)$ 's complement can also be obtained by subtracting each digit of  $N$  from  $r-1$ . Using the above methodology we can also define the 7's and 8's complement for octal system and 15's and 16's complement for hexadecimal system.

### 1.7.7 Rules for Subtraction Using $r$ 's and $(r-1)$ 's Complement

Let  $M$  (minuend) and  $S$  (subtrahend) be the two numbers to be used to evaluate the difference  $D = M - S$ , by using  $r$ 's complement and  $(r - 1)$ 's complements, and either or both the numbers may be signed or unsigned.



Untill and unless specified the given rules are equally applied to both the complements for both signed and unsigned and arithmetic. For the clarity of process let us assume that two data sets are:

Unsigned data—  $M_u = 1025$ ,  $S_u = 50$  and  $D_u = M_u - S_u$

Signed data—  $M_s = -370$ ,  $S_s = 4312$  and  $D_s = M_s - S_s$

For illustration purpose  $(r - 1)$ 's complement is used for unsigned and  $r$ 's complement for signed arithmetic.

### Step 1. Equate the Length

Find out the length of both the numbers (no. of digit) and see if both are equal. If not, then make the both the numbers equal by placing leading zeroes.

$$M_u = 1025, S_u = 50 \rightarrow \text{So } S_u = 0050$$

$$M_s = -370, S_s = 4312 \rightarrow M_s = -0370$$

### Step 2. Represent Negative Operands (for Negative Numbers only)

If either or both of operands are negative then take the appropriate complement of the number as obtained in step 1.

$$M_s = -370, \rightarrow r\text{'s of } M_s = 9999 - 0370 + 1 \rightarrow M_s = 9630 \text{ and } S_s = 4312$$

### Step 3. Complement the Subtrahend

In order to evaluate difference take the appropriate complement of the representation obtained for the SUBTRAEND  $S_U$  in step 1 and  $S_S$  in step 2.

$$S_u = 0050, (r - 1)\text{'s of } S_u = 9999 - 0050 \rightarrow S_u = 9949 \text{ and we've } M_u = 1025$$

$$S_s = 4312, r\text{'s of } S_s = 9999 - 4312 + 1 \rightarrow S_s = 5688 \text{ and we've } M_s = 9630$$

### Step 4. Addition and The Carry (CY)

Add the two numbers in the step 3 and check weather or not carry generated from MSD (Most Significant Digit) due to addition.

$$M_u = 1025, S_u = 9949 \rightarrow \text{So } D_u = M_u - S_u = 10974$$

↓  
CY

$$M_s = 9630, S_s = 5688 \rightarrow \text{So } D_s = M_s - S_s = 15318$$

↓  
CY

### Step 5. Process the Carry (CY)

In step 4, we obtained result as CY, D. The CY from MSD contains some useful information especially in some unsigned arithmetic. Processing is different for two complement.

- For  $r$ 's complement. In the case of  $r$ 's complement if there is carry from MSD in step 4 then simply discard it. We are using  $r$ 's complement to perform signed operation. In step 4 we get  $CY = 1$ ,  $D_s = 5318$  after discarding the CY.
- For  $(r - 1)$ 's complement. In this case if a carry is generated from MSD in step 4, add this carry to the LSD of the result. (We are using  $r - 1$ 's complement for

unsigned) In step 4 we got  $CY = 1$ ,  $D_u = 0974$  after adding carry to the LSD (from MSD in step 4 we get  $D_u = 0974 + 1 \rightarrow 0975$ . In this case carry is called “end-around carry”.

### Step 6. Result Manipulation

The manipulation of result is same for both the complements. The way result is manipulated is different for signed and unsigned arithmetic.

#### (a) UNSIGNED

- (1) If a carry was generated in step 4 then the result is positive(+) and the digits in the result shows the correct magnitude of result.
- (2) If there is no carry from MSD in step 4 then the result is negative (-) and the digits in result is not showing the correct magnitude. So must go for a post processing (Step 7) of result to determine the correct magnitude of the result.

#### (b) SIGNED

- (1) If the MSD of result obtained in step 5 is lesser than the half radix (*i.e.*,  $MSD < r/2$ ) then the result is +ve and representing the correct magnitude. Thus no post processing is required.
- (2) If the MSD of result obtained in step 5 is not lesser than the half radix (*i.e.*,  $MSD \geq r/2$ ) = then the result is -ve and correct magnitude of which must be obtained by post processing (Step 7).

### Step 7. Post Processing and Result Declaration

By the step 6 (a) – 1 and the step 6 (b) – 1 we know that if the result is +ve (positive) it represents the correct magnitude weather it is signed or unsigned arithmetic. However for the negative results are not showing correct magnitudes *so post processing in principle is needed for declaration of negative results.*

- (a) Declare positive results. As per the rules the result of the unsigned arithmetic is positive.

$$D_u = +0975 \text{ (Ans.)}$$

- (b) Process and declare negative results. As per the rules result of signed arithmetic is negative and is in complemented form. Take the appropriate complement to find the complement and declare the result.

$$r's \text{ of } D_s = 5318 = 9999 - 5318 + 1 = -4682 \text{ (Ans.)}$$

## 1.8 BINARY CODED DECIMAL (BCD) AND ITS ARITHMETIC

The BCD is a group of four binary bits that represent a decimal digit. In this representation each digit of a decimal number is replaced by a 4-bit binary number (*i.e.*, a nibble). Since a decimal digit is a number from 0 to 9, a nibble representing a number greater than 9 is invalid BCD. For example  $(1010)_2$  is invalid BCD as it represents a number greater than 9. The table shown in Fig. 1.13 lists the binary and BCD representation of decimal numbers 0 to 15. Carefully observe the difference between binary and BCD representation.

Decimal Number	Binary Representation	BCD Representation
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 0
3	0 0 1 1	0 0 1 1
4	0 1 0 0	0 1 0 0
5	0 1 0 1	0 1 0 1
6	0 1 1 0	0 1 1 0
7	0 1 1 1	0 1 1 1
8	1 0 0 0	1 0 0 0
9	1 0 0 1	1 0 0 1
10	1 0 1 0	0 0 0 1 0 0 0 0
11	1 0 1 1	0 0 0 1 0 0 0 1
12	1 1 0 0	0 0 0 1 0 0 1 0
13	1 1 0 1	0 0 0 1 0 0 1 1
14	1 1 1 0	0 0 0 1 0 1 0 0
15	1 1 1 1	0 0 0 1 0 1 0 1

**Fig. 1.13** Binary and BCD representation of decimal numbers

**BCD Addition:** In many application it is required to add two BCD numbers. But the adder circuits used are simple binary adders, which does not take care of peculiarity of BCD representation. Thus one must verify the result for valid BCD by using following rules:

1. If Nibble (*i.e.*, group of 4-bits) is less than or equal to 9, it is a valid BCD Number.
2. If Nibble is greater than 9, it is invalid. Add 6 (0110) to the nibble, to make it valid.

OR

If a carry was generated from the nibble during the addition, it is invalid. Add 6 (0110) to the nibble, to make it valid.

3. If a carry is generated when 6 is added, add this carry to next nibble.

**Example.** Add the following BCD numbers. (i) 1000 and 0101 (ii) 00011001 and 00011000

**Solution.** (i)

$$\begin{array}{r}
 1\ 0\ 0\ 0 \longrightarrow 8 \\
 +\ 0\ 1\ 0\ 1 \longrightarrow +\ 5 \\
 \hline
 1\ 1\ 0\ 1 \qquad \qquad 13
 \end{array}$$

Since,  $(1101)_2 > (9)_{10}$  add 6 (0110) to it

So,

$$\begin{array}{r}
 1\ 1\ 0\ 1 \\
 0\ 1\ 1\ 0 \\
 \hline
 \underbrace{1}_{1} \underbrace{0\ 0\ 1\ 1}_{3}
 \end{array}$$

So, result = 00010011

(i)

1	← Carry generated from nibble
0 0 0 1	1 0 0 1 → 19
0 0 0 1	1 0 0 0 → +18
0 0 1 1	0 0 0 1
	37

Since, a carry is generated from right most nibble we must add 6 (0110) to it.

So,

0 0 1 1	0 0 0 1	
	0 1 1 0	
0 0 1 1	0 1 1 1	→ (37) <sub>10</sub>

So, result = 00110111

**BCD Subtraction.** The best way to carry out the BCD subtraction is to use complements. The 9's and 10's complement, studied in subsection 1.7.5, are exclusively used for this purpose. Although any of the two complements can be used, we prefer 10's complement for subtraction. Following are the steps to be followed for BCD subtraction using 10's complement:

1. Add the 10's complement of subtrahend to minuend.
2. Apply the rules of BCD addition to verify that result of addition is valid BCD.
3. Apply the rules of 10's complement on the result obtained in step 2, to declare the final result *i.e.*, to declare the result of subtraction.

**Example.** Subtract 61 from 68 using BCD.

**Solution.** To illustrate the process first we perform the subtraction using 10's complement in decimal system. After that we go for BCD subtraction.

we have  $D = 68 - 61$

So, 10's complement of 61 =  $99 - 61 + 1 = 39$

So,

6 8	
+ 3 9	
1 0 7	
↑	
Carry	

In 10's complement if an end carry is produced then it is discarded and result is declared positive. So,

$D = +07$

by using BCD

1.

1	← Carry generated from nibble
BCD of 68 =	0 1 1 0 1 0 0 0
BCD of 39 = +	0 0 1 1 1 0 0 1
	1 0 1 0 0 0 0 1

2. Check for valid BCD— since a carry is generated from right most nibble, we must add 6 (0110) to it. Since the left most nibble is greater than 9, we must add 6(0110) to it.

Thus,

$$\begin{array}{r}
 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1 \\
 +\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \\
 \hline
 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1 \\
 \uparrow \\
 \text{Carry}
 \end{array}$$

3. Declaration of result – We got end carry is step 2. In 10’s complement arithmetic, end carry is discarded and result is declared positive. Hence,

$$D = (00000111)_2 = (7)_{10}$$

### 1.9 CODES

Coding and encoding is the process of assigning a group of binary digits, commonly referred to as ‘bits’, to represent, identify, or relate to a multivalued items of information. By assigning each item of information a unique combination of bits (1’s and 0’s), we transform some given information into another form. In short, a code is a symbolic representation of an information transform. The bit combination are referred to as ‘CODEWORDS’.

There are many different coding schemes, each having some particular advantages and characteristics. One of the main efforts in coding is to standardize a set of universal codes that can be used by all.

In a broad sense we can classify the codes into five groups:

- (i) Weighted Binary codes
- (ii) Non-weighted codes
- (iii) Error-detecting codes
- (iv) Error-correcting codes
- (v) Alphanumeric codes.

#### 1.9.1 Weighted Binary Codes

In weighted binary codes, each position of a number represents a specific weight. The bits are multiplied by the weights indicated; and the sum of these weighted bits gives the equivalent decimal digit. We have been familiar with the binary number system, so we shall start with straight binary codes.

(a) **Straight Binary coding** is a method of representing a decimal number by its binary equivalent. A straight binary code representing decimal 0 through 7 is given in table below:

<i>Decimal</i>	<i>Three bit straight Binary Code</i>	<i>Weights MOI</i>			<i>Sum</i>
		$2^2$	$2^1$	$2^0$	
0	000	0	0	0	0
1	001	0	0	1	1
2	010	0	2	0	2
3	011	0	2	1	3
4	100	4	0	0	4
5	101	4	0	1	5
6	110	4	2	0	6
7	111	4	2	1	7

In this particular example, we have used three bits to represent 8 distinct elements of information *i.e.*, 0 through 7 in decimal form.

Now the question arises, if  $n$  elements of information are to be coded with binary (two valued bits), then how many bits are required to assign each element of information a unique code word (bit combination). Unique is important, otherwise the code would be ambiguous.

The best approach is to evaluate how many code words can be derived from a combination of  $n$  bits.

For example: Let  $n$  = no. of bits in the codeword and  $x$  = no. of unique words

Now, if

$$n = 1, \text{ then } x = 2 \text{ (0, 1)}$$

$$n = 2, \text{ then } x = 4 \text{ (00, 01, 10, 11)}$$

$$n = 3, \text{ then } x = 8 \text{ (000, 001, ..., 111)}$$

and in general,

$$n = j, \text{ then } x = 2^j$$

that is, if we have available  $j$  no. of bits in the code word, we can uniquely encode max  $2^j$  distinct elements of information.

Inversely, if we are given  $x$  elements of information to code into binary coded format, the following condition must hold:

$$x \leq 2^j$$

or

$$j \geq \log_2 x$$

or

$$j \geq 3.32 \log_{10} x$$

where

$$j = \text{number of bits in code word.}$$

**Example.** How many bits would be required to code the 26 alphabetic characters plus the 10 decimal digits.

**Solution.** Here we have total 36 discrete elements of information.

$$\Rightarrow x = 36$$

$$\text{Now } j \geq \log_2 x$$

$$\Rightarrow j \geq \log_2 36 \text{ or } j \geq 3.32 \log_{10} 36$$

or

$$j \geq 5.16 \text{ bits}$$

Since bits are not defined in fractional parts, we know  $j \geq 6$ .

In other words, a minimum of 6 bit code is required that leaves 28 unused code words out of the 64 which are possible ( $2^6 = 64$  and  $64 - 36 = 28$ ).

This system of straight binary coding has the disadvantage that the large numbers require a great deal of hardware to handle with. For example if we have to convert decimal 2869594 to straight binary code a regrous division of this number by 2 is required untill we get remainder 0 or 1.

The above difficulty is overcome by using another coding scheme called as BCD codes.

(b) **Binary Codes Decimal Codes (BCD codes).** In BCD codes, individual decimal digits are coded in binary notation and are operated upon singly. Thus binary codes represent-

ing 0 to 9 decimal digits are allowed. Therefore all BCD codes have at least four bits ( $\therefore$  min. no. of bits required to encode to decimal digits = 4)

For example, decimal 364 in BCD

$$\begin{array}{r} 3 \rightarrow 0011 \\ 6 \rightarrow 0110 \\ \hline 4 \rightarrow 0100 \\ \hline 364 \rightarrow 0011\ 0110\ 0100 \end{array}$$

However, we should realize that with 4 bits, total 16 combinations are possible (0000, 0001, ..., 11 11) but only 10 are used (0 to 9). The remaining 6 combinations are invalid and commonly referred to as 'UNUSED CODES'.

There are many binary coded decimal codes (BCD) all of which are used to represent decimal digits. Therefore all BCD codes have atleast 4 bits and at least 6 unassigned or unused code words.

Some example of BCD codes are:

(a) 8421 BCD code, sometimes referred to as the Natural Binary Coded Decimal Code (NBCD);

(b)\* Excess-3 code (XS3);

(c)\*\* 84 -2 -1 code (+8, +4, -2, -1);

(d) 2 4 2 1 code

**Example.** Lowest  $[643]_{10}$  into XS3 code

$$\begin{array}{r} \text{Decimal} \quad 6 \quad 4 \quad 3 \\ \text{Add 3 to each} \quad \underline{3 \quad 3 \quad 3} \\ \text{Sum} \quad 9 \quad 7 \quad 6 \end{array}$$

Converting the sum into BCD code we have

$$\begin{array}{ccc} 0 & 7 & 6 \\ \downarrow & \downarrow & \downarrow \\ 1001 & 0111 & 0110 \end{array}$$

Hence, XS3 for  $[643]_{10} = 1001\ 0111\ 0110$

---

\*-XS3 is an example of nonweighted code but is a type of BCD code. It is obtained by adding 3 to a decimal number. For example to encode the decimal number 7 into an excess 3 code. We must first add 3 to obtain 10. The 10 is then encoded in its equivalent 4 bit binary code 1010. Thus as the name indicates, the XS3 represents a decimal number in binary form, as a number greater than 3.

\*\* - Dashes (-) are minus signs.

**Table : BCD codes**

<i>Decimal Digit</i>	<i>8421 (NBCD)</i>	<i>Excess-3 code (XS3)</i>	<i>84-2-1 code</i>	<i>2421 code</i>
0	0000	0011	0000	0000
1	0001	0100	0111	0001
2	0010	0101	0110	0010
3	0011	0110	0101	0011
4	0100	0111	0100	0100
5	0101	1000	1011	1011
6	0110	1001	1010	1100
7	0111	1010	1001	1101
8	1000	1011	1000	1110
9	1001	1100	1111	1111

There are many BCD codes that one can develop by assigning each column or bit position in the code, some weighting factor in such a manner that all of the decimal digits can be coded by simply adding the assigned weights of the 1 bits in the code word.

For example: 7 is coded 0111 in NBCD, which is interpreted as

$$0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = 7$$

The NBCD code is most widely used code for the representation of decimal quantities in a binary coded format.

For example: (26.98) would be represented in NBCD as

$$(26.98)_{10} = \begin{matrix} & 2 & 6 & 9 & 8 \\ & (0010 & 0110. & 1001 & 1000) \text{ NBCD} \end{matrix}$$

It should be noted that on the per digit basis the NBCD code is the binary numeral equivalent of the decimal digit it represents.

### Self complementing BCD codes

The excess 3, 8 4-2-1 and 2421 BCD codes are also known as self complementing codes.

Self complementing property- 9's complement of the decimal number is easily obtained by changing 1's to 0's and 0's to 1's in corresponding codeword or the 9's complement of self complementing code word is the same as its logical complement.

When arithmetic is to be performed, often an arithmetic "complement" of the numbers will be used in the computations. So these codes have a particular advantage in machines that use decimal arithmetic.

**Example.** The decimal digit 3 in 8 4-2-1 code is coded as 0101. The 9's complement of 3 is 6. The decimal digit 6 is coded as 1010 that is 1's complement of the code for 3. This is termed as self complementing property.

### 1.9.2 Non Weighted Codes

These codes are not positionally weighted. This means that each position within a binary number is not assigned a fixed value. Excess-3 codes and Gray codes are examples of non-weighted codes.

We have already discussed XS3 code.



**Gray code (Unit Distance code or Reflective code)**

There are applications in which it is desirable to represent numerical as well as other information with a code that changes in only one bit position from one code word to the next adjacent word. This class of code is called a unit distance code (UDC). These are sometimes also called as 'cyclic', 'reflective' or 'gray' code. These codes finds great applications in Boolean function minimization using Karnaugh map.

The gray code shown in Table below is both reflective and unit distance.

**Table : Gray codes\***

<i>Decimal Digit</i>	<i>Three bit Gray code</i>	<i>Four bit Gray code</i>
0	0 0 0	0 0 0 0
1	0 0 1	0 0 0 1
2	0 1 1	0 0 1 1
3	0 1 0	0 0 1 0
4	1 1 0	0 1 1 0
5	1 1 1	0 1 1 1
6	1 0 1	0 1 0 1
7	1 0 0	0 1 0 0
8	–	1 1 0 0
9	–	1 1 0 1
10	–	1 1 1 1
11	–	1 1 1 0
12	–	1 0 1 0
13	–	1 0 1 1
14	–	1 0 0 1
15	–	1 0 0 0

\*Gray codes are formed by reflection. The technique is as follows:

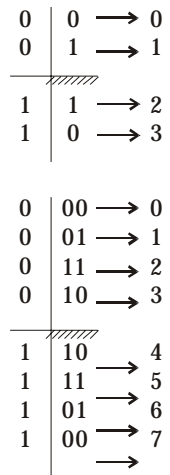
In binary we have two digits 0 and 1.

**Step I.** Write 0 and 1 and put a mirror; we first see 1 and then 0. Place 0's above mirror and 1's below mirror

We have got gray code for decimal digits 0 through 4.

**Step II.** Write these 4 codes and again put a mirror. The code will look in the order 10, 11, 01 and 00. Then place 0's above mirror and 1's below mirror.

Proceeding intactively in the same manner. We can form Gray code for any decimal digit.



### Binary to Gray conversion

- (1) Place a leading zero before the most significant bit (MSB) in the binary number.
- (2) Exclusive-OR (EXOR) adjacent bits together starting from the left of this number will result in the Gray code equivalent of the binary number.

Exclusive-OR- If the two bits EX-OR'd are identical, the result is 0; if the two bits differ, the result is 1.

**Example.** Convert binary 1010010 to Gray code word.

$$\begin{array}{cccccccc} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ \wedge & \wedge & \wedge & \wedge & \wedge & \wedge & \wedge & \wedge \\ & 1 & 1 & 1 & 1 & 0 & 1 & 1 \end{array} \Rightarrow (1010010)_2 = (1111011)_{\text{Gray}}$$

### Gray to Binary conversion

Scan the gray code word from left to right. The first 1 encountered is copied exactly as it stands. From then on, 1's will be written until the next 1 is encountered, in which case a 0 is written. Then 0's are written until the next 1 is encountered, in which case a 1 is written, and so on.

**Example 1.** Convert Gray code word 1111011 into binary.

$$\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{array} \Rightarrow (1111011)_{\text{Gray}} = (1010010)_2$$

**Example 2.** Convert Gray code word 10001011 into binary.

$$\begin{array}{cccccccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{array} \Rightarrow (10001011)_{\text{Gray}} = (11110010)_2$$

### 1.9.3 Error Detecting Codes

Binary information is transmitted from one device to another by electric wires or other communication medium. A system that can not guarantee that the data received by one device are identical to the data transmitted by another device is essentially useless. Yet anytime data are transmitted from source to destination, they can become corrupted in passage. Many factors, including external noise, may change some of the bits from 0 to 1 or viceversa. Reliable systems must have a mechanism for detecting and correcting such errors.

Binary information or data is transmitted in the form of electro magnetic signal over a channel whenever an electromagnetic signal flows from one point to another, it is subject to unpredictable interference from heat, magnetism, and other forms of electricity. This interference can change the shape or timing of signal. If the signal is carrying encoded binary data, such changes can alter the meaning of data.

In a single bit error, a 0 is changed to a 1 or a 1 is changed to a 0.

In a burst error, multiple (two or more) bits are changed.

The purpose of error detection code is to detect such bit reversal errors. Error detection uses the concept of **redundancy** which means adding extra bits for detecting errors at the destination.

For a single bit error detection, the most common way to achieve error detection is by means of a **parity bit**.

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

Table below shows a message of three bits and its corresponding odd and even parity bits.

If an odd parity is adopted, P bit is chosen such that the total no. of 1's is odd in four bit that constitute message bits and P.

If an even parity is adopted, the P bit is chosen such that the total number of 1's is even.

**Table: Parity bit generation**

Message			Odd Parity (P) bit	Even Parity bit (P)
x	y	z		
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

The message with the parity bit (either odd or even) is transmitted to its destination. The parity of the received data is checked at the receiving end. If the parity of the received data is changed (from that of transmitted parity), it means that at least one bit has changed their value during transmission. Though the parity code is meant for single error detection, it can detect any odd number of errors. However, in both the cases the original codeword can not be found.

If there is even combination of errors (means some bits are changed but parity remains same) it remains undetected.

**Checksums**—The checksum method is used to detect double errors in bits. Since the double error will not change the parity of the bits, the parity checker will not indicate any error.

In the checksums method, Initially a word A (let 11001010) is transmitted, next word B (let 00101101) is transmitted. The sum of these two words is retained in the transmitter. Then a word C is transmitted and added to the previous sum; and the new sum is retained. Similarly, each word is added to the previous sum; after transmission of all the words, the final sum called the checksum is also transmitted. The same operation is done at the receiving end and the final sum obtained here is checked against the transmitted checksum. If the two sums are equal there is no error.

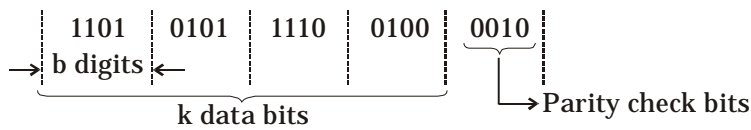
### Burst Error Detection

So far we have considered detecting or correcting errors that occur independently or randomly in digit positions. But disturbances can wipe out an entire block of digits. For example, a stroke of lightening or a human made electrical disturbance can affect several transmitted digits. Burst errors are those errors that wipe out some or all of a sequential set of digits.

A burst of length  $b$  is defined as a sequence of digits in which the first digit and  $b$  for Ball digit are in error, with the  $b-2$  digits in between either in error or received correctly. For Example  $l$  and  $m$  represent errors then a sequence  $10:l 10 m00 110l01m:l0$  has a burst length 13.

It can be shown that for detecting all burst errors of length  $b$  or less;  $b$  parity check bits are necessary and sufficient.

To construct such a code, lets group  $k$  data digits into segment of  $b$  digits in length as shown below:



Burst error detection

To this we add a last segment of  $b$  parity check digits, which are determined as follows:

“The modulo-2 sum\* of the  $i$ th digit in each segment (including the parity check segment) must be zero.”

It is easy to see that if a single sequence of length  $b$  or less is in error, parity will be violated and the error will be detected and the reciever can request retransmission of code.

### 1.9.4 Error Correcting Codes

The mechanism that we have covered upto this point detect errors but do not correct them. Error correction can be handled in two ways. In one, when an error is encountered the receiver can request the sender to retransmit entire data unit. In the other, a receiver can use an error correcting code, which automatically corrects certain errors.

In theory, it is possible to correct any binary code errors automatically using error correcting codes, however they require more reductant bits than error detecting codes. The number of bits required to correct a multiple-bit or burst error is so high that in most cases, it is inefficient to do so. For this reason, most error correction is limited to one, two, or three-bit errors. However, we shall confine our discussion to only single bit error correction.

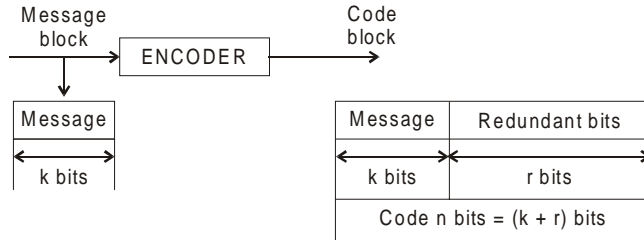
As we saw earlier, single bit errors can be detected by the addition of a redundant (parity) bit to the data (information) unit. This provides sufficient base to introduce a very popular error detection as well correction codes, known as Block codes.

**Block codes:**  $[(n, k)$  codes] In block codes, each block of  $k$  message bits is encoded into a larger block of  $n$  bits ( $n > k$ ), as shown. These are also known as  $(n, k)$  codes.

Modulo-2 sum denoted by symbol  $\oplus$  with the rules of addition as follows:

$$\left. \begin{array}{l} 0 \oplus 0 = 0 \\ 0 \oplus 1 = 1 \\ 1 \oplus 0 = 1 \\ 1 \oplus 1 = 0 \end{array} \right\}$$

The reductant\* (parity) bits 'r' are derived from message bits 'k' and are added to them. The 'n' bit block of encoder output is called a codeword.



The simplest possible block code is when the number of reductant or parity bits is one. This is known as parity check code. It is very clear that what we have studied in single bit error detection is nothing but a class of 'Block codes'.

R.W. Hamming developed a system that provides a methodical way to add one or more parity bit to data unit to detect and correct errors weight of a code.

### Hamming distance and minimum distance

The weight of a code word is defined as the number of nonzero components in it. For example,

Code word	Weight
010110	3
101000	2
000000	0

The 'Hamming distance' between two code words is defined as the number of components in which they differ.

For example, Let  $U = 1010$   
 $V = 0111$   
 $W = 1001$

Then,  $D(U, V) = \text{distance between } U \text{ and } V = 3$   
 Similarly,  $D(V, W) = 3$   
 and  $D(U, W) = 2$

The 'minimum distance' ( $D_{\min}$ ) of a block code is defined as the smallest distance between any pair of codewords in the code.

From Hamming's analysis of code distances, the following important properties have been derived. If  $D_{\min}$  is the minimum distance of a block code then

- (i) 't' number of errors can be detected if  $D_{\min} = t + 1$
- (ii) 't' number of errors can be corrected if  $D_{\min} = 2t + 1$

It means, we need a minimum distance ( $D_{\min}$ ) of at least 3 to correct single error and with this minimum. distance we can detect upto 2 errors.

---

\*The 'r' bit are not necessarily appear after 'k' bits. They may appear at the starting, end or in between 'k' data bits.

Now coming to our main objective *i.e.*, error correction, we can say that an error occurs when the receiver reads 1 bit as a 0 or a 0 bit as a 1. To correct the error, the receiver simply reverses the value of the altered bit. To do so, however, it must know which bit is in error. The secret of error correction, therefore, is to locate the invalid bit or bits.

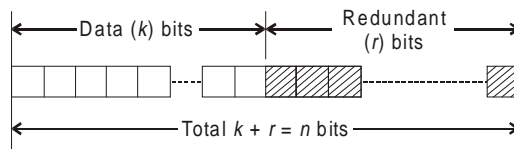
For example, to correct a single bit error in a seven bit data unit, the error correction code must determine, which of the seven data bits has changed. In the case we have to distinguish between eight different states: no error, error in position 1, error in position 2, and so on, upto error in position 7. To do so requires enough redundant bits to show all eight states.

At first glance, it appears that a 3-bit redundant code should be adequate because three bits can show eight different states (000 to 111) and can thus indicate the locations of eight different possibilities. But what if an error occurs in the redundant bits themselves. Seven bits of data plus three bits of redundancy equals 10 bits. Three bits, however, can identify only eight possibilities. Additional bits are necessary to cover all possible error locations.

### Redundant Bits

To calculate the number of redundant bits ( $r$ ) required to correct a given no. of data bits ( $k$ ), we must find a relationship between  $k$  and  $r$ . Figure shows  $k$  bits of data with  $r$  bits of redundancy added to them. The length of the resulting code is thus  $n = k + r$ .

If the total no. of bits in code is  $k + r$ , then  $r$  must be able to indicate at least  $k + r + 1$  different states. Of these, one state means no error and  $k + r$  states indicate the location of an error in each of the  $k + r$  positions.



Alternatively, we can say the  $k + r + 1$  status must be discoverable by  $r$  bits; and  $r$  bits can indicate  $2^r$  different states. Therefore,  $2^r$  must be equal to or greater than  $k + r + 1$ :

$$2^r \geq k + r + 1$$

The value of  $r$  can be determined by plugging in the value of  $k$  (the length of data unit). For example, if the value of  $k$  is 7 ( $\Rightarrow$  seven bit data), the smallest  $r$  value that can satisfy this equation is 4:

$$2^4 \geq 7 + 4 + 1$$

and

$$2^3 \not\geq 7 + 4 + 1$$

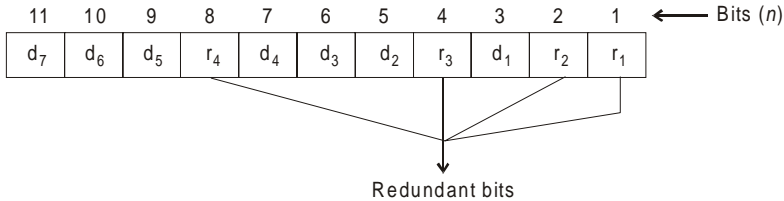
### 1.9.5 Hamming Code

So far, we have examined the number of bits required to cover all of the possible single bit error states in a transmission. But how do we manipulate those bits to discover which state has occurred? A technique developed by R.W. Hamming provides a practical solution, Hamming code is a class of block code ( $n, k$ ) and we are going to discuss (11, 7) Hamming code.

#### Positioning the Redundant bits

The 'Hamming code' can be applied to data units of any length and uses the relationship between data and redundant bits as discussed above. As we have seen, a 7 bit data unit ( $\Rightarrow k = 7$ ) requires 4 redundant bits ( $\Rightarrow r = 4$ ) that can be added to the end of data unit (or interspersed with data bits). Such that a code word of length 11 bits ( $n = 11$ ) is formed.

In figure these bits are placed in positions 1, 2, 4 and 8 (the positions in an 11-bit sequence that are powers of 2). We refer these bits as  $r_1, r_2, r_4$  and  $r_8$ .

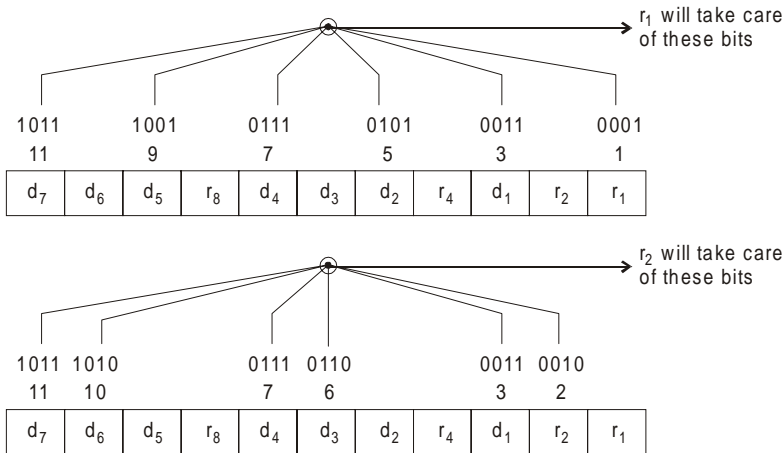


In the Hamming code, each  $r$  bit is the redundant bit for one combination\* of data bits. The combinations (modulo-2 additions) used to calculate each of four  $r$  values (viz,  $r_1, r_2, r_4$  and  $r_8$ ) for a 7 bit data sequence  $d_1$  through  $d_7$  are as follows:

- $r_1$  : bits 1, 3, 5, 7, 9, 11
- $r_2$  : bits 2, 3, 6, 7, 10, 11
- $r_4$  : bits 4, 5, 6, 7
- $r_8$  : bits 8, 9, 10, 11

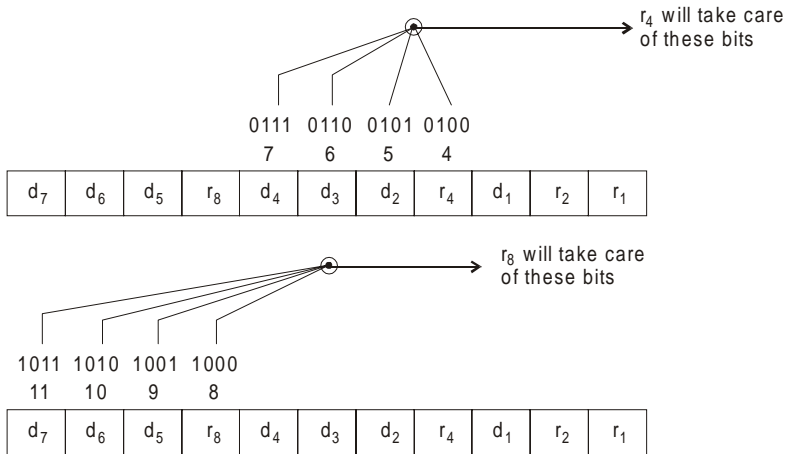
Each data bit may be included in more than one redundant bit calculation. In the sequences above, for example, each of the original data bits is included in at least two sets, while the  $r$  bits are included in only one.

To see the pattern behind this strategy, look at the binary representation of each bit position. The  $r_1$  bit is calculated using all bit positions whose binary representation includes a 1 in the right most position. The  $r_2$  bit is calculated using all bit positions with a 1 in the second position, and so on. (see Fig.)



\*In codes combination of bits means modulo 2 addition the data bits. Modulo-2 addition applies in binary field with following rules.

$$\begin{aligned}
 0 \oplus 0 &= 0 && \text{Modulo 2 } \rightarrow \oplus \\
 0 \oplus 1 &= 1 \\
 1 \oplus 0 &= 1 \\
 1 \oplus 1 &= 0
 \end{aligned}$$



**Calculating the  $r$  values**

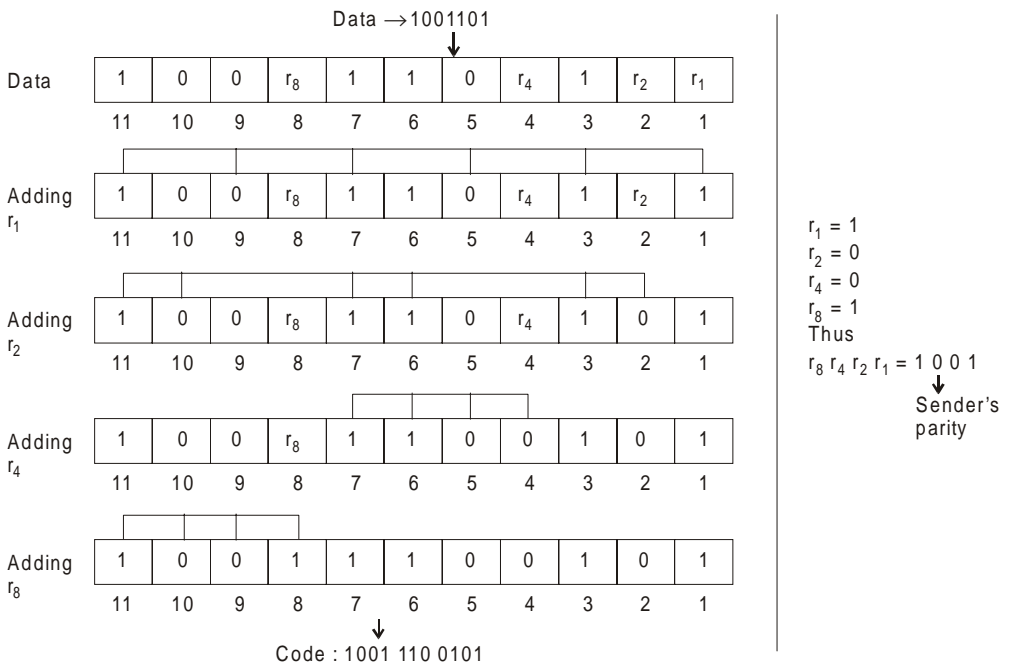
Figure shows a Hamming code implementation for a 7 bit data unit. In the first step; we place each bit of original data unit in its appropriate position in the 11-bit unit. For example, let the data unit be 1001101.

In the subsequent steps; we calculate the **EVEN** parities for the various bit combinations. The even parity value for each combination is the value of corresponding  $r$  bit. For example, the value of  $r_1$  is calculated to provide even parity for a combination of bits 3, 5, 7, 9 and 11.

$\Rightarrow$  1011 1001 0111 0101 0011 0001

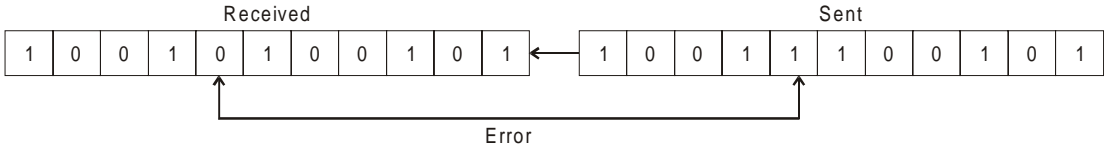
Here the total no. of 1's are 13. Thus to provide even parity  $r_1 = 1$ .

Similarly the value of  $r_2$  is calculated to provide even parity with bits 3, 6, 7, 10,  $r_4$  with bits 5, 6, 7 and  $r$  with bits 9, 10, 11. The final 11-bit code is sent.

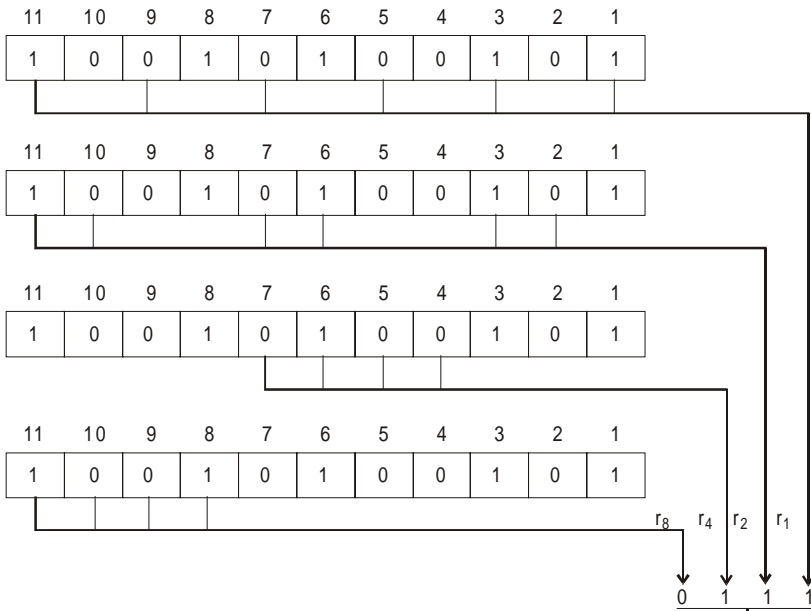




**Error detection and correction** – Suppose above generated code is received with the error at bit number 7  $\Rightarrow$  bit has changed from 1 to 0 see figure below:



The receiver receives the code and recalculates four new redundant bits ( $r_1, r_2, r_4$  and  $r_8$ ) using the same set of bits used by sender plus the relevant parity bit for each set shown below



The bit in position 7 is in Error  $\leftarrow$  7 in decimal

Then it assembles the new parity values into a binary number in order of  $r$  position ( $r_8, r_4, r_2, r_1$ ). In our example, this step gives us the binary number 0111 (7 in decimal), which is the precise location of the bit in error.

Once the bit is identified, the receiver can reverse its value and correct the error.

**Note:** If the new parity assembled is same as the parity at sender's end mean no error.

### 1.9.6 Cyclic Codes

Binary cyclic codes form a subclass of linear block codes.

An  $(n, k)$  linear block code is called the cyclic code if it satisfies the following property:

If an  $n$  tuple (a row vector of  $n$  elements),  $V = (V_0, V_1, V_2, \dots, V_{n-1})$  is a code vector or  $C$ , then the  $n$  tuple

$$V^1 = (V_{n-1}, V_0, V_1, \dots, V_{n-2})$$

obtained by shifting  $V$  cyclically one place to the right (it may be left also) is also a code vector of  $C$ . From above definition it is clear that

$$V^{(j)} = (V_{n-i}, V_{n-i+1}, \dots, V_0, V_1, \dots, V_{n-i-1}).$$

An example of cyclic code:

$$\begin{array}{cccc}
 1 & 1 & 0 & 1 \\
 \swarrow & \searrow & \searrow & \searrow \\
 1 & 1 & 1 & 0 \\
 \swarrow & \searrow & \searrow & \searrow \\
 0 & 1 & 1 & 1 \\
 \swarrow & \searrow & \searrow & \searrow \\
 1 & 0 & 1 & 1 \\
 \hline
 1 & 1 & 0 & 1
 \end{array}$$

It can be seen that 1101, 1110, 0111, 1011 is obtained by a cyclic shift of  $n$ -tuple 1101 ( $n = 4$ ). The code obtained by rearranging the four words is also a cyclic code. Thus 1110, 0111, 1011 are also cyclic codes.

This property of cyclic code allows to treat the codewords as a polynomial form. A procedure for generating an  $(n, k)$  cyclic code is as follows:

The bits of uncoded word (message) Let  $D = [d_0, d_1, d_2 \dots d_{k-1}]$  are written as the coefficients of polynomial of degree  $k - 1$ .

$$D(x) = d_0x^0 \oplus d_1x^1 \oplus d_2x^2 \oplus \dots \oplus d_{k-1}x^{k-1}$$

Similarly, the coded word, let  $V = [v_0, v_1, v_2, \dots, v_{n-1}]$  are written as the coefficients of polynomial of degree  $n - 1$ .

$$V(x) = v_0x^0 \oplus v_1x^1 \oplus v_2x^2 \oplus \dots \oplus v_{n-1}x^{n-1}$$

The coefficients of the polynomials are 0's and 1's and they belong to the binary field with the modulo-2 rules for addition as described in Hamming codes.

Now, we will state a theorem\* which is used for cyclic code generation.

**Theorem.** If  $g(x)$  is a polynomial of degree  $(n-k)$  and is a factor of  $x^{n+1}$ , then  $g(x)$  generates an  $(n, k)$  cyclic code in which the code polynomial  $V(x)$  for a data polynomial  $D(x)$  is given by

$$V(x) = D(x) \cdot g(x)$$

where

- $V(x)$  - Code word polynomial of degree  $(n - 1)$
- $D(x)$  - Data word polynomial of degree  $(k - 1)$
- $g(x)$  - Generator polynomial of degree  $(n - k)$

**Example.** Consider a  $(7, 4)$  cyclic code. The generator polynomial for this code is given as  $g(x) = 1 + x + x^3$ . Find all the code words of this code.

**Solution.** It is a  $(7, 4)$  cyclic code

$\Rightarrow$   $n =$  No. of bits in coded word  $= 7$

and  $k =$  No. of bits in data word  $= 4$ .

$(n - k) =$  No. of redundant bits in code word  $= 3$

It implies that, there are 16 different messages that are possible (10000, 0001, 0010 . . . 1110, 1111). Correspondingly, there will be 16 different codes (of 7 bits).

Now, according to above theorem, the generator polynomial  $g(x)$  must be a factor of  $(x^n + 1)**$  and of degree  $(n - k)$ .

$$x^n + 1 = x^7 + 1$$

If we factorize this polynomial we get

$$x^7+1 = (x + 1) \quad (x^3 + x + 1) \quad (x^3 + x^2 + 1)$$

I                      II                      III

\*Without giving proof that is beyond the scope of this book.

\*\*+ means modulo-2 operation  $\oplus$  in binary codes.

$$\begin{aligned} \Rightarrow \quad & \text{I Factor} \rightarrow x + 1 \\ & \text{II Factor} \rightarrow x^3 + x + 1 \\ & \text{III Factor} \rightarrow x^3 + x^2 + 1 \end{aligned}$$

The I factor does not satisfy the requirement that it must be of degree  $(n - k)$  but the II and III do satisfy.

Therefore, we can either choose II Factor or III Factor as generator polynomial  $g(x)$ . However, the set of codewords will naturally be different for these two polynomial.

In this example, we have taken  $g(x)$  as  $1 + x + x^3$ .

*i.e.*, we have to encode the 16 messages using generator polynomial.

$$g(x) = 1 + x + x^3.$$

Consider, for example, a data word 1010.

$$\Rightarrow \quad D = (d_0, d_1, d_2, d_3) = (1010)$$

Because the length is four, the data polynomial  $D(x)$

will be of the form  $d_0 + d_1x + d_2x^2 + d_3x^3$

$$\Rightarrow \quad D(x) = 1 + 0.x + 1.x^2 + 0.x^3 = 1 + x^2$$

The code polynomial

$$\begin{aligned} V(x) &= D(x) \cdot g(x) \\ &= (1 + x^2) \cdot (1 + x + x^3) \\ &= 1 + x + x^2 + \frac{x^3 + x^3}{0} + x^5 \end{aligned}$$

$$\Rightarrow \quad V(x) = 1 + x + x^2 + x^5$$

$\therefore$  if  $x = 1$  then

$$x^3 = 1$$

or if  $x = 0$  then

$$x^3 = 0$$

$$0 \oplus 0 = 1 \oplus 1 = 0$$

Because the length of codeword and  $(n)$  is 7.

So the standard polynomial will be of the form.

$$V(x) = V_0 + V_1x + V_2x^2 + V_3x^3 + V_4x^4 + V_5x^5 + V_6x^6$$

Comparing this standard polynomial with above poly. for  $V(x)$

we get

$$V = [1110010]$$

In a similar way, all code vectors can be found out.

## 1.10 SOLVED EXAMPLES

**Example. 1.** Convert each binary number to the decimal:

- |     |                    |     |              |
|-----|--------------------|-----|--------------|
| (a) | $(11)_2$           | (b) | $(.11)_2$    |
|     | $(1011)_2$         |     | $(.111)_2$   |
|     | $(10111)_2$        |     | $(.1011)_2$  |
|     | $(1111)_2$         |     | $(.10101)_2$ |
|     | $(11010111)_2$     |     | $(.0101)_2$  |
|     | $(1001)_2$         |     | $(.110)_2$   |
| (c) | $(11.11)_2$        |     |              |
|     | $(1011.1011)_2$    |     |              |
|     | $(1111.0101)_2$    |     |              |
|     | $(11010111.110)_2$ |     |              |
|     | $(1001.10101)_2$   |     |              |

**Solution.** (a)

$$\begin{aligned}
 (11)_2 &= 1 \times 2^1 + 1 \times 2^0 \\
 &= 2 + 1 = 3 \\
 (1011)_2 &= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
 &= 8 + 0 + 2 + 1 = 11 \\
 (10111)_2 &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
 &= 16 + 0 + 4 + 2 + 1 = 23 \\
 (1111)_2 &= 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
 &= 8 + 4 + 2 + 1 = 15 \\
 (11010111)_2 &= 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + \\
 &\quad 1 \times 2^1 + 1 \times 2^0 \\
 &= 128 + 64 + 0 + 16 + 0 + 4 + 2 + 1 = 215 \\
 (1001)_2 &= 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\
 &= 8 + 0 + 0 + 1 = 9
 \end{aligned}$$

(b)

$$\begin{aligned}
 (.11)_2 &= 1 \times 2^{-1} + 1 \times 2^{-2} \\
 &= .5 + .25 = (.75)_{10} \\
 (.111)_2 &= 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} \\
 &= .5 + .25 + .125 = (.875)_{10} \\
 (.1011)_2 &= 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} \\
 &= .5 + 0 + .125 + .0625 = (.6875)_{10} \\
 (.10101)_2 &= 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} \\
 &= .5 + 0 + .125 + 0 + .03125 = (.65625)_{10} \\
 (.0101)_2 &= 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} \\
 &= 0 + .25 + 0 + .0625 = (.3125)_{10} \\
 (.110)_2 &= 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} \\
 &= .5 + .25 + 0 = (.75)_{10}
 \end{aligned}$$

(c)

$$11.11 = ?$$

From part (a) and part (b), we see that

$$11 = 3$$

$$11 = .75$$

Therefore,

$$(11.11)_2 = (3.75)_{10}$$

$$1011.1011 = ?$$

$$(1011)_2 = 11$$

$$(.1011)_2 = .6875$$

Therefore,

$$(1011.1011)_2 = (11.6875)_{10}$$

$$1111.0101 = ?$$

$$(1111)_2 = 15$$

$$(.0101)_2 = .3125$$

Therefore,

$$(1111.0101)_2 = (15.3125)_{10}$$

$$11010111.110 = ?$$

$$11010111 = 215$$

$$.110 = .75$$

$$\begin{aligned} (11010111.110)_2 &= (215.75)_{10} \\ 1001.10101 &= ? \\ 1001 &= 9 \\ .10101 &= .65625 \\ (1001.10101)_2 &= (9.65625)_{10} \end{aligned}$$

**Example 2.** How many bits are required to represent the following decimal numbers, represent them in binary.

- (I)  $(17)_{10}$  (II)  $(27)_{10}$  (III)  $(81)_{10}$  (IV)  $(112)_{10}$  (V)  $(215)_{10}$

**Solution.** (I) Let  $n$  bits required

$n$  should be such that

$$2^n \geq \text{Given Number (N)}$$

Therefore,  $2^n \geq 17$

i.e.,  $n \geq 5$

Therefore minimum number of bits required = 5.

2	17	1	↑ Remainder LSB
2	8	0	
2	4	0	
2	2	0	
2	1	1	
	0		

MSB

$(17)_{10} = (10001)_2$

(II)  $(27)_{10}$

The minimum number of bits required is given by

$$2^n \geq N \text{ (given number)}$$

$$2^n \geq 27$$

i.e.,  $n \geq 5$

2	27	1	↑ LSB
2	13	1	
2	6	0	
2	3	1	
2	1	1	
	0		

MSB

$(27)_{10} = (11011)_2$

(III)  $(81)_{10}$

The minimum number of bits required is given by

$$2^n \geq N$$

$$2^n \geq 81$$

i.e.,  $n = 7$

2	81	1	↑ Remainder LSB
2	40	0	
2	20	0	
2	10	0	
2	5	1	
2	2	0	
2	1	1	
	0		

MSB

$(81)_{10} = (1010001)_2$

(IV)  $(112)_{10}$

The minimum number of required is given by

$$2^n \geq N$$

$$2^n \geq 112$$

i.e.,  $n = 7$

2	112	0	↑ LSB
2	56	0	
2	28	0	
2	14	0	
2	7	1	
2	3	1	
2	1	1	
	0		

MSB

$(112)_{10} = (1110000)_2$

(V)  $(215)_{10}$

The minimum number of bits required is given by

$$2^n \geq 215$$

i.e.,  $n = 8$

2	215	1	↑ LSB        MSB ↓
2	107	1	
2	53	1	
2	26	0	
2	13	1	
2	6	0	
2	3	1	
2	1	1	
	0		

$$(215)_{10} = (11010111)_2$$

**Example 3.** Convert the following numbers as indicated:

- (a) decimal 225.225 to binary, octal and hexadecimal.
- (b) binary 11010111.110 to decimal, octal and hexadecimal.
- (c) octal 623.77 to decimal, binary and hexadecimal.
- (d) hexadecimal 2AC5.D to decimal, octal and binary.

**Solution.** (a)  $225.225 = (?)_2$

			Remainder	
2	225	1	↑ LSB        MSB ↓	
2	112	0		
2	56	0		
2	28	0		
2	14	0		
2	7	1		
2	3	1		
2	1	1		
	0			

Integer part = 11100001

( $.225$ ) <sub>10</sub>	=	( $?$ ) <sub>2</sub>	Carry	
.225 × 2	=	0.450	0	↓
.450 × 2	=	0.900	0	
.900 × 2	=	1.800	1	
.800 × 2	=	1.600	1	
.600 × 2	=	1.200	1	
.200 × 2	=	0.400	0	
.400 × 2	=	0.800	0	
.800 × 2	=	1.600	1	
.600 × 2	=	1.200	1	

Fraction part = 001110011

Therefore,

$$(225.225)_{10} = 11100001.001110011$$

$$(225.225)_{10} = (?)_8$$

From the previous step we know the binary equivalent of decimal no. as 11100001.001110011.

For octal number, binary number is partitioned into group of three digit each starting from right to left and replacing decimal equivalent of each group.

$$\begin{array}{ccccccc} \underline{-11} & \underline{100} & \underline{001} & . & \underline{001} & \underline{110} & \underline{011} \\ \underline{011} & \underline{100} & \underline{001} & . & \underline{001} & \underline{110} & \underline{011} \\ \downarrow & \downarrow & \downarrow & & & & \\ 3 & 4 & 1 & & 1 & 6 & 3 \end{array}$$

$$(225.225)_{10} = (341.163)_8$$

$$(225.225)_{10} = (?)_{16}$$

For hexadecimal number, instead of three four digits are grouped.

$$\begin{array}{ccccccc} \underline{1110} & \underline{0001} & . & \underline{0011} & \underline{1001} & \underline{1- -} & \\ \underline{1110} & \underline{0001} & . & \underline{0011} & \underline{1001} & \underline{1000} & \\ 14 \equiv \mathbf{E} & 1 & & 3 & 9 & 8 & \\ (225.225)_{10} & = & \mathbf{E1.398} & & & & \end{array}$$

$$(b) \quad (11010111.110)_2 = (?)_{10}$$

From example 1.6.1

$$(11010111.110)_2 = (215.75)_{10}$$

$$(11010111.110)_2 = (?)_8$$

$$\begin{array}{cccc} \underline{-11} & \underline{010} & \underline{111} & \\ \underline{011} & \underline{010} & \underline{111} & = 327 \\ \downarrow & \downarrow & \downarrow & \\ 3 & 2 & 7 & \end{array}$$

$$\begin{array}{ccc} \underline{.110} & & \\ \downarrow & = & 6 \\ \mathbf{6} & & \\ (11010111.110)_2 & = & (327.6)_8 \\ (11010111.110)_2 & = & (?)_{16} \end{array}$$

$$\begin{array}{ccc} \underline{1101} & \underline{0111} & \\ 13 \equiv \mathbf{D} & 7 & = \mathbf{D7} \end{array}$$

$$\begin{array}{ccc} \underline{.110-} & & \\ \underline{.1100} & = & \mathbf{C} \\ 12 = \mathbf{C} & & \end{array}$$

$$(11010111.110)_2 = (\mathbf{D7.C})_{16}$$

$$(c) \quad \begin{array}{ccc} (623.77)_8 & = & (?)_2 \\ 623 & = & 110010011 \\ .77 & = & 11111 \end{array}$$

$$(623.77)_8 = (110010011.111111)_2$$

$$(623.77)_8 = (?)_{16}$$

$$\begin{array}{r} \underline{-} \underline{-} \underline{-} \underline{1} \underline{1001} \underline{0011} \\ \underline{0001} \underline{1001} \underline{0011} \\ 1 \quad 9 \quad 3 \end{array} = 193$$

$$\begin{array}{r} \underline{1111} \quad \underline{11} \underline{-} \underline{-} \\ \underline{1111} \quad \underline{1100} \\ 15 = F \quad 12 = C \end{array} = FC$$

$$(623.77)_8 = (193.FC)_{16}$$

$$(623.77)_8 = (?)_{10}$$

$$\begin{aligned} 623 &= 6 \times 8^2 + 2 \times 8^1 + 3 \times 8^0 \\ &= 384 + 16 + 3 \\ &= 403 \end{aligned}$$

$$\begin{aligned} .77 &= 7 \times 8^{-1} + 7 \times 8^{-2} \\ &= 7 \times .125 + 7 \times .015625 \\ &= .875 + .109375 \\ &= 0.9843 \end{aligned}$$

$$(623.77)_8 = (403.9843)_{10}$$

(d)  $(2AC5.D)_{16} = (?)_2$

$$2AC5 = 0010101011000101$$

$$D = 1101$$

$$(2AC5.D)_{16} = (10101011000101.1101)_2$$

$$(2AC5.D)_{16} = (?)_8$$

$$\begin{array}{r} \underline{-10} \underline{101} \underline{011} \underline{000} \underline{101} \cdot \underline{110} \underline{1} \underline{-} \underline{-} \\ \underline{010} \underline{101} \underline{011} \underline{000} \underline{101} \cdot \underline{110} \underline{100} \\ 2 \quad 5 \quad 3 \quad 0 \quad 5 \quad 6 \quad 4 \end{array}$$

$$(2AC5.D)_{16} = (25305.64)_8$$

$$(2AC5.D)_{16} = (?)_{10}$$

$$\begin{aligned} 2AC5 &= 2 \times 16^3 + 10 \times 16^2 + 12 \times 16^1 + 5 \times 16^0 \\ &= 2 \times 4096 + 10 \times 256 + 12 \times 16 + 5 \times 1 \\ &= 8192 + 2560 + 192 + 5 \\ &= 10949 \end{aligned}$$

$$D = 13 \times 16^{-1}$$

$$= 13 \times .0625$$

$$= .8125$$

$$(2AC5.D)_{16} = (10949.8125)_{10}$$







## 1.11 EXERCISES

1. Write 9's and 10's complement of the following numbers:  
 $+9090$   
 $-3578$   
 $+136.8$   
 $-136.28$
2. (a) Convert the decimal integer's  $+21$  and  $-21$  into 10's complement and 9's complement.  
 (b) Convert the above two numbers in binary and express them in six bit (total) signed magnitude and 2's complement.
3. (a) Find the decimal equivalent of the following binary numbers assuming signed magnitude representation of the binary number:  
 (I) 001000      (II) 1111  
 (b) Write the procedure for the subtraction of two numbers with  $(r - 1)$ 's complement.  
 (c) Perform the subtraction with the following binary numbers using 2's complement and 1's complement respectively.  
 (I)  $11010 - 1101$       (II)  $10010 - 10011$   
 (d) Perform the subtraction with following decimal numbers using 10's complement and 9's complement respectively.  
 (I)  $5294 - 749$       (II)  $27 - 289$
4. Convert:  
 (I)  $(225.225)_{12}$  to Hexadecimal number.  
 (II)  $(2AC5.15)_{16}$  to Octal number.
5. Perform the following using 6's complement:  
 (I)  $(126)_7 + (42)_7$   
 (II)  $(126)_7 - (42)_7$
6. Represent the following decimal numbers in two's complement format:  
 (I)  $+5$       (II)  $+25$       (III)  $-5$       (IV)  $-25$       (V)  $-9$
7. Represent the decimal numbers of question 6 in ones complement format.
8. Find the decimal equivalent of each of the following numbers assuming them to be in two's complement format.  
 (a) 1000      (b) 0110      (c) 10010      (d) 00110111
9. Convert the following octal numbers into equivalent decimal numbers:  
 (a) 237      (b) 0.75      (c) 237.75
10. Represent the following decimal numbers in sign-magnitude format:  
 (a)  $-11$       (b)  $-7$       (c)  $+12$       (d)  $+25$

# 2 CHAPTER

## DIGITAL DESIGN FUNDAMENTALS— BOOLEAN ALGEBRA AND LOGIC GATES

---

### 2.0 INTRODUCTORY CONCEPTS OF DIGITAL DESIGN

George Boole, in his work entitled ‘An Investigation of the Laws of Thought’, on which are founded the Mathematical Theories of Logic and Probability (1854), introduced the fundamental concepts of a two-values (binary) system called Boolean Algebra. This work was later organized and systemized by Claude Shannon in ‘Symbolic Analysis of Relay and Switching Circuits (1938)’. Digital design since that time has been pretty much standard and advanced, following Boole’s and Shannon’s fundamentals, with added refinements here and there as new knowledge has been unearthed and more exotic logic devices have been developed.

Digital design is the field of study relating the adoption of **Logic** concepts to the design of recognizable, realizable, and reliable digital hardware.

When we begin study of logic, digital logic, binary systems, switching circuits, or any other field of study that can be classified as being related to digital design, we must concern ourselves with learning some philosophical premises from which we must launch our studies. In order to reach a desirable theoretical, as well as conceptual, understanding of digital design, you must grasp some fundamental definitions and insight giving concepts.

Generally speaking, being involved in digital design is dealing in “LOGIC” a term that certainly needs some definition. LOGIC, by definition, *is a process of classifying information. Information is intelligence related to ideas, meanings, and actions which can be processed or transformed into other forms.* For example, NEWS is information by virtue of the fact that it is intelligence related to ACTIONS, be it good news or bad news. News can be heard, read, seen or even felt or any combination of all four, indicating the possibility of its transformation into different forms.

“BINARY LOGIC,” or two-valued logic, *is a process of classifying information into two classes.* Traditionally, binary arguments, or that information which can be definitely classified as two valued, has been delivered either TRUE or FALSE. Thus, the Boolean variable is unlike the algebraic variables of the field of real numbers in that any Boolean variable can take on only two values, the TRUE or the FALSE. Traditionally, (High or Low-Asserted or Not Asserted) it is standard to use the shorthand symbols 1 for TRUE and 0 for the FALSE.

### 2.1 TRUTH TABLE

A Boolean variable can take on only two values, not an infinite number as, the variable of the real number system, can. This basic difference allows us to illustrate all possible logic conditions of a Boolean variable or a collection of Boolean variables using a finite tabular

format called a ‘truth-table’. Further, the nontrivial decisions in digital design are based on more than one-two valued variable. Thus, if an output is to be completely specified as a function of two inputs, there are four input combinations that must be considered. If there are three inputs, then eight combinations must be considered and from this we see that  $n$  inputs will require  $2^n$  combinations to be considered.

A TRUTH-TABLE as suggested is a tabular or graphical technique for listing all possible combinations of input variables, arguments, or whatever they may be called, in a vertical order, listing each input combination one row at a time (Table 2.1). When every possible combination is recorded, each combination can be studied to determine whether the ‘output’ or ‘combined interaction’ of that combination should be ASSERTED or NOT-ASSERTED. Of course the information used to determine the combined interaction or output must come from studying arguments of the logic problem. For example

- (i) Let we have a TV that operates with a switch. The TV, becomes on or off with the switch on or off respectively.

**Table 2.1(a)**

True ↑	I/P Switch	O/P TV
High ← ASSERTED ←	Off 0	Off 0
Low ← NOT ASSERTED ←	On 1	On 1
↓ False		

- (ii) Let we have a TV that operates with two switches. When both the switches are ‘ON’ then only TV becomes ‘ON’ and in all other cases TV is ‘Off’.

**Table 2.1(b)**

<i>S.1</i>	<i>S.2</i>	<i>TV</i>
0	0	0 OFF
0	1	0 OFF
1	0	0 OFF
1	1	1 ON

- (iii) Let the TV operate with three switches. The condition now is that when at least two switches are ‘ON’ the TV becomes ‘ON’ and in all other conditions ‘TV’ is ‘OFF’.

**Table 2.1(c)**

<i>S.1</i>	<i>S.2</i>	<i>S.3</i>	<i>TV</i>
0	0	0	0 OFF
0	1	0	0 OFF
0	1	0	0 OFF
0	1	1	1 ON
1	0	0	0 OFF
1	0	1	1 ON
1	1	0	1 ON
1	1	1	0 OFF

Table 2.1(a) illustrates the use of a one variable T.T. and how the output or combined interaction is manually listed to the right of each possible combination. Table 2.1(b) and Table 2.1(c) show the standard form for two and three variable truth-tables. In review, what is

suggested here is that after all the input variables have been identified, and all the possible combinations of these variables have been listed in the truth-table on the left, then each row should be studied to determine what output or combined interaction is desired for that input combination. Further, note that the input combinations are listed in ascending order, starting with the binary equivalent of zero. The TRUTH-TABLE also allows us to establish or prove Boolean identities without detailed mathematical proofs, as will be shown later.

## 2.2 AXIOMATIC SYSTEMS AND BOOLEAN ALGEBRA

In chapter 1 we have discussed the AND, OR, and INVERTER functions and stated that it can be proven that these functions make up a sufficient set to define a two-valued Boolean Algebra. Now we introduce some formal treatment to this two-valued Boolean algebra.

### Axiomatic Systems

Axiomatic systems are founded on some fundamental statements referred to as ‘axioms’ and ‘postulates.’ *As you delve deeper into the origin of axioms and postulates, you find these to be predicted on a set of undefined objects that are accepted on faith.*

Axioms or postulates are statements that make up the framework from which new systems can be developed. They are the basis from which theorems and the proofs of these theorems are derived. For example, *proofs are justified on the basis of a more primitive proof.* Thus, we use the statement—*‘From this we justify this.’* Again, we find a process that is based on some point for which there exist no further primitive proofs. Hence, we need a starting point and that starting point is a set of axioms or postulates.

Axioms are formulated by combining intelligence and empirical evidence and should have some basic properties. These are:

1. They are statements about a set of undefined objects.
2. They must be consistent, that is, they must not be self-contradictory.
3. They should be simple but useful, that is, not lengthy or complex.
4. They should be independent, that is, these statements should not be interdependent.

The study of axiomatic systems related to logic motivated the creation of the set of postulates known as the ‘HUNTINGTON POSTULATES’. E.V. Huntington (1904) formulated this set of postulates that have the basic properties described desirable, consistent, simple and independent. These postulates as set forth can be used to evaluate proposed systems and those systems that meet the criteria set forth by these postulates become known as Huntington System. Further, once a proposed system meets the criteria set forth by the Huntington Postulates, automatically all theorems and properties related to other Huntington systems become immediately applicable to the new system.

Thus, we propose a Boolean algebra and test it with the Huntington postulates to determine its structure. We do this so that we can utilize the theorems and properties of other Huntington system for a new system that is defined over a set of voltage levels and hardware operators. Boolean algebra, like other axiomatic systems, is based on several operators defined over a set of undefined elements. A SET is any collection of elements having some common property; and these elements need not be defined. The set of elements we will be dealing with is {0, 1}. The 0 and 1, as far as we are concerned, are some special symbols and have no numerical annotation whatsoever. They are simply some objects we are going to make some statements about. An operation ( $\cdot$ ,  $+$ ) is defined as a rule defining the results of an operation on two elements of the set. Because these operators operate on two elements, they are commonly referred to as “binary operators”.



4(b) Can be shown by a similar table.

5. From the INVERTER function table

(COMPLEMENT)

$$1 \cdot \bar{1} = 1 \cdot 0 = 0, \quad 0 \cdot \bar{0} = 0 \cdot 1 = 0$$

$$1 + \bar{1} = 1 + 0 = 1, \quad 0 + \bar{0} = 0 + 1 = 1$$

6. It is obvious that the set  $S = \{0, 1\}$  fulfills the minimum requirements of having at least two elements where  $0 \neq 1$ .

From this study the following postulates can be listed below:

**Table 2.2.1**

Postulate 2	(a) $A + 0 = A$	(b) $A \cdot 1 = A$	Intersection Law
Postulate 3	(a) $A + B = B + A$	(b) $A \cdot B = B \cdot A$	Commutating Law
Postulate 4	(a) $A(B + C) = AB + AC$	(b) $A + BC = (A + B)(A + C)$	Distributive Law
Postulate 5	(a) $A + \bar{A} = 1$	(b) $A \cdot A' = 0$	Complements Law

We have just established a two valued Boolean algebra having a set of two elements, 1 and 0, two binary operators with operation rules equivalent to the AND or OR operations, and a complement operator equivalent to the NOT operator. Thus, Boolean algebra has been defined in a formal mathematical manner and has been shown to be equivalent to the binary logic presented in Chapter 1. The presentation is helpful in understanding the application of Boolean algebra in gate type circuits. The formal presentation is necessary for developing the theorems and properties of the algebraic system.

### 2.2.2 Basic Theorems and Properties of Boolean Algebra

*Duality.* The Huntington postulates have been listed in pairs and designated by part (a) and (b) in Table 2.2.2. One part may be obtained from other if the binary operators (+ and  $\cdot$ ) and identity elements (0 and 1) are interchanged. This important property of Boolean algebra is called the duality principle. It states that every algebraic expression deducible from the postulates of Boolean algebra remain valid if the operators and identity elements are interchanged. In a two valued Boolean algebra, the identity elements and the elements of the set are same: 1 and 0.

*Basic Theorems.* Table 2.2.2 lists six theorems of Boolean algebra. The theorems, like the postulates, are listed in pairs; each relation is the dual of the one paired with it. The postulates are basic axioms of the algebraic structure and need no proof. The theorems must be proven from the postulates.

**Table 2.2.2 Theorems of Boolean Algebra**

Theorem		
1. (a) $A + A = A$	(b) $A \cdot A = A$	Tautology Law
2. (a) $A + 1 = 1$	(b) $A \cdot 0 = 0$	Union Law
3. (a) $(A')' = A$		Involution Law
4. (a) $A + (B + C) = (A + B) + C$	(b) $A \cdot (B \cdot C) = (A \cdot B) \cdot C$	Associative Law
5. (a) $(A + B)' = A'B'$	(b) $(A \cdot B)' = A' + B'$	De Morgan's Law



- 6. (a)  $A + AB = A$  (b)  $A(A + B) = A$  Absorption Law
- 7. (a)  $A + A'B = A + B$  (b)  $A(A' + B) = AB$
- 8. (a)  $AB + AB' = A$  (b)  $(A + B)(A + B') = A$  Logical adjancy
- 9. (a)  $AB + A'C + BC = AB + A'C$  (b)  $(A + B)(A' + C)(B + C) = (A + B)$  Consensus Law

The proofs of the theorem are presented below. At the right is listed the number of postulate which justifies each step of proof.

Theorem 1(a)  $A + A = A$

$$\begin{aligned}
 A + A &= (A + A).1 && \text{by postulate 2(b)} \\
 &= (A + A)(A + A') && 5(a) \\
 &= A + AA' && 4(b) \\
 &= A + 0 && 5(b) \\
 &= A && 2(a)
 \end{aligned}$$

Theorem 1(b)  $A.A = A.$

$$\begin{aligned}
 A.A &= A.A + 0 && \text{by postulate 2(a)} \\
 &= A.A + A.A' && 5(b) \\
 &= A(A + A') && 4(a) \\
 &= A.1 && 5(a) \\
 &= A && 2(b)
 \end{aligned}$$

Note that theorem 1(b) is the dual of theorem 1(a) and that each step the proof in part (b) is the dual of part (a). Any dual theorem can be similarly derived from the proof of its corresponding pair.

Theorem 2(a)  $A + A = 1$

$$\begin{aligned}
 A + 1 &= 1.(A + 1) && \text{by postulate 2(b)} \\
 &= (A + A')(A + 1) && 5(a) \\
 &= A + A'.1 && 4(b) \\
 &= A + A' && 2(b) \\
 &= 1 && 5(a)
 \end{aligned}$$

Theorem 2(b)  $A.0 = 0$  by duality.

Theorem 3.  $(A')' = A$  From postulate 5, we have

$A + A' = 1$  and  $A.A' = 0$ , which defines the complement of A. The complement of A' is A and is also (A')'. Therefore, since the complement is unique, we have that (A')' = A.

Theorem 4(a)  $A + (B + C) = (A + B) + C$

We can prove this by perfect induction method shown in table below:

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

(Contd.)...

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

We can observe that  $A + (B + C) = (A + B) + C$

\*Theorem 4(b)—can be proved in similar fashion.

\*Theorem 5(a) and 5(b)—can also be proved by perfect induction method.

Theorem 6(a)  $A + AB = A$   
 $A + AB = A(1 + B)$   
 $= A(1)$   
 $= A$

6(b)  $A.(A + B) = A$  By duality.

Theorem 7(a)  $A + A'B = A + B$   
 $A + A'B = A.1 + A'B$   
 $= A(B + B') + A'B$   
 $= AB + AB' + A'B$   
 $= AB + AB + AB' + A'B$   
 $= A(B + B') + B(A + A')$   
 $= A + B.$

7(b)  $A.(A' + B) = A.B$  By duality.

Theorem 8(a)  $AB + AB' = A$   
 $AB + AB' = A(B + B')$   
 $= A$

8(b)  $(A + B) . (A + B') = A$  By duality.

Theorem 9(a)  $AB + A'C + BC = AB + A'C$   
 $AB + A'C + BC = AB + A'C + BC(A + A')$   
 $= AB + A'C + ABC + A'BC$   
 $= AB(1 + C) + A'C(1 + B)$   
 $= AB + A'C$

9(b)  $(A + B) (A' + C) (B + C) = (A + B) (A' + C)$  By duality.

### 2.3 BOOLEAN FUNCTIONS

A binary variable can take the value of 0 or 1. A Boolean function is an expression formed with binary variable, the two binary operators OR and AND, the unary operator NOT, parantheses 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

$$F_1 = xy'z$$

---

\*The proof of 4(b), 5(a) and 5(b) is left as an exercise for the reader.

The function  $F$  is equal to 1 when  $x = 1$ ,  $y = 0$  and  $z = 1$ ; otherwise  $F = 0$ . This is an example of a Boolean function represented as an algebraic expression. A Boolean function may 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  $n$  binary variables, and a column showing the combinations for which the function is equal to 1 or 0 as discussed previously. As shown in Table 2.3, there are eight possible distinct combination for assigning bits to three variables. The table shows that the function  $F$  is equal to 1 only when  $x = 1$ ,  $y = 0$  and  $z = 1$  and equal to 0 otherwise.

Consider now the function

$$F_2 = x'y'z + x'yz + xy'$$

$F_2 = 1$  if  $x = 0$ ,  $y = 0$ ,  $z = 1$  or

$x = 0$ ,  $y = 1$ ,  $z = 1$  or

$x = 1$ ,  $y = 0$ ,  $z = 0$  or

$x = 1$ ,  $y = 0$ ,  $z = 1$

$F_2 = 0$ , otherwise.

**Table 2.3**

$x$	$y$	$z$	$F_1$	$F_2$	$F_3$
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	1	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	1	0	0	0

The number of rows in the table is  $2^n$ , where  $n$  is the number of binary variables in the function.

The question now arises, Is an algebraic expression of a given Boolean function unique? Or, is it possible to find two algebraic expressions that specify the same function? The answer is yes. Consider for example a third function.

$$F_3 = xy' + x'z$$

$F_3 = 1$  if  $x = 1$ ,  $y = 0$ ,  $z = 0$  or

$x = 1$ ,  $y = 0$ ,  $z = 1$  or

$x = 0$ ,  $y = 0$ ,  $z = 1$  or

$x = 0$ ,  $y = 1$ ,  $z = 1$

$F_3 = 0$ , otherwise.

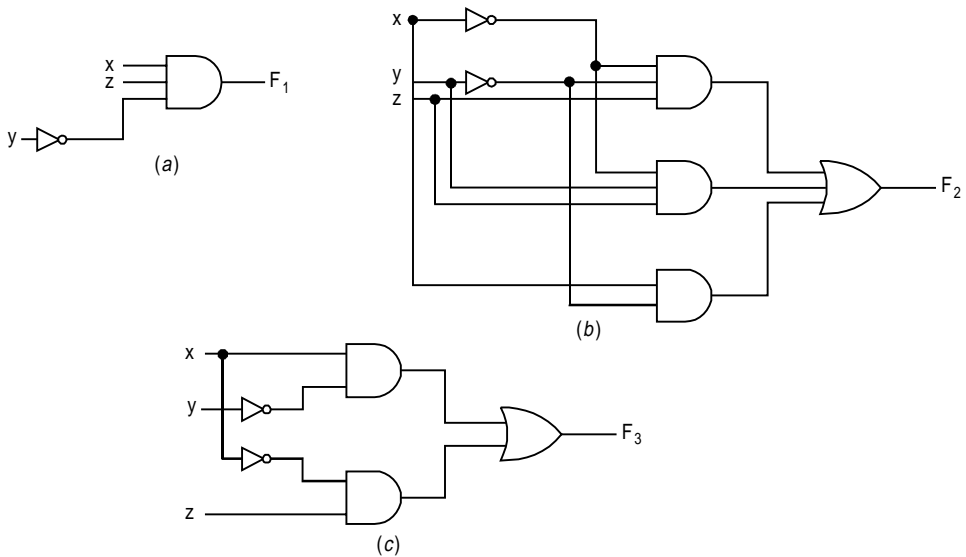
From table, we find that  $F_3$  is same as  $F_2$  since both have identical 1's and 0's for each combination of values of the three binary variables. In general, two functions of  $n$  binary variables are said to be equal if they have the same value for all possible  $2^n$  combinations of the  $n$  variables.

As a matter of fact, the manipulation of Boolean algebra is applied mostly to the problem of finding simpler expressions for the same function.

### 2.3.1 Transformation of Boolean Function into Logic Diagram

A Boolean function may be transformed from an algebraic expression into a logic diagram composed of AND, OR and NOT gates. Now we shall implement the three functions discussed above as shown in Fig. 2.3.1.

- Here we are using inverters (NOT gates) for complementing a single variable. In general however, it is assumed that we have both the normal and complement forms available.
- There is an AND gate for each product term in the expression.
- An OR gate is used to combine (seen) two or more terms.



**Fig. 2.3.1** (a, b, c)

From the diagrams, it is obvious that the implementation of  $F_3$  requires fewer gates and fewer inputs than  $F_2$ . Since  $F_3$  and  $F_2$  are equal Boolean functions, it is more economical to implement  $F_3$  form than the  $F_2$  form. To find simpler circuits, we must know how to manipulate Boolean functions to obtain equal and simpler expression. These simplification (or minimization) techniques will be related in detail in next chapter.

### 2.3.2 Complement of a Function

The complement of a function  $F$  is  $F'$  and is obtained from an interchange of 0's for 1's ... and 1's for 0's in the value of  $F$ . The complement of a function may be derived algebraically through De Morgan's theorem. De Morgan's theorem can be extended to three or more variables. The three-variable form of the De Morgan's theorem is derived below:

$$\begin{aligned}
 (A + B + C)' &= (A + X)' && \text{Let } B + C = X \\
 &= A'X' && \text{by theorem 5(a)} \\
 &= A' \cdot (B + C)' && \text{substituting } B + C = X \\
 &= A' \cdot (B'C') && \text{theorem 5(a)} \\
 &= A'B'C' && \text{theorem 4(a)}
 \end{aligned}$$

This theorem can be generalized as

$$(A + B + C + D + \dots + F)' = A'B'C'D'\dots F'$$

and its DUAL

$$(ABCD \dots F)' = A' + B' + C' + D' + \dots + F'$$

The generalized form of De Morgan's theorem states that the complement of a function is obtained by interchanging AND and OR operators and complementing each literal.

**Example.** Determine the complements of the following function:

$$F_1 = AB' + C'D$$

**Solution.**

$$F_1 = AB' + C'D$$

$$\begin{aligned} F_1' &= (AB' + C'D)' \\ &= (AB')' \cdot (C'D)' \\ &= (A' + B) \cdot (C + D') \end{aligned}$$

## 2.4 REPRESENTATION OF BOOLEAN FUNCTIONS

Boolean functions (logical functions) are generally expressed in terms of logical variables. Values taken on by the logical functions and logical variables are in the binary form. Any logical variable can take on only one of the two values 0 and 1 or any logical variable (binary variable) may appear either in its normal form (A) or in its complemented form (A'). As we will see shortly later that an arbitrary logic function can be expressed in the following forms:

- (i) Sum of Products (SOP)
- (ii) Product of Sums (POS)

### Product Term

The AND function is referred to as product. The logical product of several variables on which a function depends is considered to be a product term. The variables in a product term can appear either in complemented or uncomplemented (normal) form. For example  $AB'C$  is a product form.

### Sum Term

The OR function is generally used to refer a sum. The logical sum of several variables on which a function depends is considered to be a sum term. Variables in a sum term also can appear either in normal or complemented form. For example  $A + B + C'$ , is a sum term.

### Sum of Products (SOP)

The logic sum of two or more product terms is called a 'sum of product' expression. It is basically on OR operation of AND operated variables such as  $F = A'B + B'C + A'BC$ .

### Product of Sums (POS)

The logical product of two or more sum terms is called a 'product of sum' expression. It is basically an AND operation of OR operated variables such as  $F = (A' + B) \cdot (B' + C) \cdot (A' + B + C)$ .

### 2.4.1 Minterm and Maxterm Realization

Consider two binary variables A and B combined with an AND operation. Since each variable may appear in either form (normal or complemented), there are four combinations, that are possible—AB, A'B, AB', A'B'.

Each of these four AND terms represent one of the four distinct combinations and is called a minterm, or a standard product or fundamental product.

Now consider three variable—A, B and C. For a three variable function there are 8 minterms as shown in Table 2.4.1. (Since there are 8 combinations possible). The binary numbers from 0 to 7 are listed under three variables. Each minterm is obtained from an AND term of the three variables, with each variable being primed (complemented form) if the corresponding bit of the binary number is a 0 and unprimed (normal form) if a 1. The symbol is  $m_j$ , where  $j$  denotes the decimal equivalent of the binary number of the minterm designated.

In a similar manner,  $n$  variables can be combined to form  $2^n$  minterms. The  $2^n$  different minterms may be determined by a method similar to the one shown in table for three variables.

Similarly  $n$  variables forming an OR term, with each variable being primed or unprimed, provide  $2^n$  possible combinations, called maxterms or standard sums.

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.

It is interesting to note that each maxterm is the complement of its corresponding minterm and vice versa.

Now we have reached to a level where we are able to understand two very important properties of Boolean algebra through an example. The example is same as we have already discussed in Section (2.1) Truth Table.

**Table 2.4.1 Minterm and Maxterm for three binary variables**

Decimal Eqn.				MINTERMS		MAXTERMS	
	A	B	C	Term	Designation	Term	Designation
0	0	0	0	A'B'C'	$m_0$	A + B + C	M <sub>0</sub>
1	0	0	1	A'B'C	$m_1$	A + B + C'	M <sub>1</sub>
2	0	1	0	A'BC'	$m_2$	A + B' + C	M <sub>2</sub>
3	0	1	1	A'BC	$m_3$	A + B' + C'	M <sub>3</sub>
4	1	0	0	AB'C'	$m_4$	A' + B + C	M <sub>4</sub>
5	1	0	1	AB'C	$m_5$	A' + B + C'	M <sub>5</sub>
6	1	1	0	ABC'	$m_6$	A' + B' + C	M <sub>6</sub>
7	1	1	1	ABC	$m_7$	A' + B' + C	M <sub>7</sub>

Let we have a TV that is connected with three switches. TV becomes 'ON' only when atleast two of the three switches are 'ON' (or high) and in all other conditions TV is 'OFF' (or low).

Let the three switches are represented by three variable A, B and C. The output of TV is represented by F. Since there are three switches (three variables), there are 8 distinct combinations possible that is shown in TT.

SWITCHES			TV (o/p)
A	B	C	F
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

HIGH (ON) → 1  
 LOW (OFF) → 0.

The TV becomes ‘ON’ at four combinations. These are 011, 101, 110 and 111. We can say that F is determined by expressing the combinations A’BC, AB’C and ABC. Since each of these minterms result in F = 1, we should have

$$F = A'BC + AB'C + ABC' + ABC$$

$$= m_3 + m_5 + m_6 + m_7.$$

This demonstrates an important property of Boolean algebra that ‘Any Boolean function can be expressed as sum of minterms or as ‘Sum of product’. However, there is no guarantee that this SOP expression will be a minimal expression. In other words, SOP expressions are likely to have redundancies that lead to systems which requires more hardware that is necessary. This is where the role of theorems and other reduction techniques come into play as will be shown in next chapter.

As mentioned, any TRUTH-TABLE INPUT/OUTPUT specifications can be expressed in a SOP expression. To facilitate this a shorthand symbology has been developed to specify such expressions. This is done by giving each row (MINTERM) in the TRUTH-TABLE a decimal number that is equivalent to the binary code of that row, and specifying the expression thus:

$$F = \Sigma(m_3, m_5, m_6, m_7)$$

which reads: F = the sum-of-products of MINTERMS 3, 5, 6 and 7. This shorthand notation can be further shortend by the following acceptable symbology:

$$F = \Sigma(3, 5, 6, 7)$$

Expression such as these serve as great aids to the simplification process, as shown in next chapter.

Now, continuing with the same example, consider the complement of Boolean function that can be read from Truth-table by forming a minterm for each combination that produces 0 in the function and by ORing

$$F' = A'B'C' + A'B'C + A'BC' + AB'C$$

Now, if we take the complement of F’, we get F.

$$\Rightarrow F = (A + B + C) \cdot (A + B + C') \cdot (A + B' + C) \cdot (A' + B + C)$$

$$= M_0 \quad M_1 \quad M_2 \quad M_4$$

This demonstrates a second important property of the Boolean algebra that ‘Any Boolean function can be expressed as product-of-maxterms or as product of sums’. The procedure for

obtaining the product of maxterms directly from Truth-table is as; Form a maxterm for each combination of the variables that produces a 0 in the function, and then form the AND of all those functions. Output will be equal to F because in case of maxterms 0 is unprimed.

The shorthand symbology for POS expressions is as follows—

$$F = \Pi(M_0, M_1, M_2, M_4)$$

or

$$F = \Pi(0, 1, 2, 4)$$

Boolean functions expressed as sum of minterms (sum of product terms) SOP or product of maxterms, (Product of sum terms) POS are said to be in CANONICAL form or STANDARD form.

### 2.4.2 Standard Forms

We have seen that for  $n$  binary variables, we can obtain  $2^n$  distinct minterms, and that any Boolean function can be expressed as a sum of minterms or product of maxterms. It is sometimes convenient to express the Boolean function in one of its standard form (SOP or POS). If not in this form, it can be made as follows:

**1. Sum of Product.** First we expand the expression into a sum of AND terms. Each term is then inspected to see if it contains all the variable. If it misses one or more variables, it is ANDed with an expression such as  $A + A'$ , where  $A$  is one of the missing variables.

**Example.** Express the Boolean function  $F = x + y'z$  in a sum of product (sum of minterms) form.

**Solution.** The function has three variables  $x, y$  and  $z$ . The first term  $x$  is missing two variables; therefore

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

This is still missing one variable:

$$\begin{aligned} x &= xy(z + z') + xy'(z + z') \\ &= xyz + xy'z' + xy'z + xy'z' \end{aligned}$$

The second term  $y'z$  is missing one variable:

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

Combining all terms, we have

$$F = x + y'z = xyz + xy'z' + xy'z + xy'z' + xy'z + x'y'z$$

But  $xy'z$  appears twice, and according to theorem 1 ( $A + A = A$ ), it is possible to remove one of them. Rearranging the min terms in ascending order, we have:

$$\begin{aligned} F &= x'y'z + xy'z' + xy'z + xy'z' + xyz \\ &= m_1 + m_4 + m_5 + m_6 + m_7. \end{aligned}$$

⇒

$$F(x, y, z) = \Sigma(1, 4, 5, 6, 7)$$

An alternative method for deriving product terms (minterms) is to make a T.T. directly from function.  $F = x + y'z$  from T.T., we can see directly five minterms where the value of function is equal to 1. Thus,



$$F(x, y, z) = \Sigma(1, 4, 5, 6, 7)$$

x	y	z	$F = x + y'z$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

**2. Product of Sums.** To express the Boolean function as product of sums, it must first be brought into a form of OR terms. This may be done by using the distributive law  $A + BC = (A + B) \cdot (A + C)$ . Then any missing variable A in each OR term is 0Red with  $AA'$ .

**Example.** Express the Boolean function  $F = AB + A'C$  in a product of sum (product of mixterm) form.

**Solution.** First convert the function into OR terms using distributive law.

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

The function has three variables A, B and C. Each OR term is missing one variable therefore:

$$\begin{aligned} A' + B &= A' + B + CC' = (A' + B + C)(A' + B + C') \\ A + C &= A + C + BB' = (A + B + C)(A + B' + C) \\ B + C &= B + C + AA' = (A + B + C)(A' + B + C) \end{aligned}$$

Combining all these terms and removing those that appear more than once.

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

$M_0 \qquad M_2 \qquad M_4 \qquad M_5$

$$\Rightarrow F(x, y, z) = \Pi(0, 2, 4, 5)$$

An alternative method for deriving sum terms (maxterms) again is to make a TT directly from function.

$$F = AB + A'C$$

From TT, we can see directly four maxterms where the value of function is equal to 0.

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

A	B	C	$F = AB + A'C$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1

(Contd.)...

A	B	C	$F = AB + A'C$
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

### 2.4.3 Conversion between Standard Forms

Consider the table shown in section 2.4.1(b) to help you establish the relationship between the MAXTERM and MINTERM numbers.

From table we see that  $m_j = \overline{M_j}$

or  $m_j = \overline{m_j}$

An interesting point can be made in relationship between MAXTERM lists and MINTERMS lists. The subscript number of the terms in the MAXTERM list correspond to the same subscript numbers for MINTERMS that are not included in the MINTERM list. From this we can say the following:

Give : II (Set of MAXTERM numbers)

We know that the function derived from this list will yield precisely the same result as the following:

$\Sigma$ (set of MINTERMS numbers that are not included in the MAXTERM list)

For example,

Given,  $F(A, B, C) = \text{II}(0, 1, 4, 6)$

We know immediately that

$F(A, B, C) = \Sigma(2, 3, 5, 7)$

## 2.5 LOGIC GATES

### Introduction

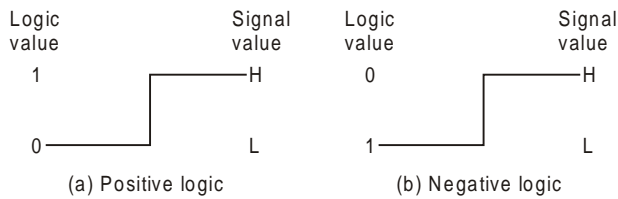
We have seen that the foundation of logic design is seated in a well defined axiomatic system called Boolean algebra, which was shown to be what is known as a “Huntington system”. In this axiomatic system the definition of AND and OR operators or functions was set forth and these were found to be well defined operators having certain properties that allow us to extend their definition to Hardware applications. These AND and OR operators, sometimes referred to as connectives, actually suggest a function that can be emulated by some H/w logic device. The logic Hardware devices just mentioned are commonly referred to as “gates”.

Keep in mind that the usage of “gate” refers to an actual piece of Hardware where “function” or “operation” refers to a logic operator AND. On the other hand, when we refer to a “gate” we are referring directly to a piece of hardware called a gate. The main point to remember is ‘Don’t confuse gates with logic operators’.

#### 2.5.1 Positive and Negative Logic Designation

The binary signals at the inputs or outputs of any gate can have one of the two values except during transition. One signal levels represents logic 1 and the other logic 0. Since two signal values are assigned two to logic values, there exist two different assignments of signals to logic.

Logics 1 and 0 are generally represented by different voltage levels. Consider the two values of a binary signal as shown in Fig. 2.5.1. One value must be higher than the other since the two values must be different in order to distinguish between them. We designate the higher voltage level by H and lower voltage level by L. There are two choices for logic values assignment. Choosing the high-level (H) to represent logic 1 as shown in (a) defines a positive logic system. Choosing the low level L to represent logic-1 as shown in (b), defines a negative logic system.



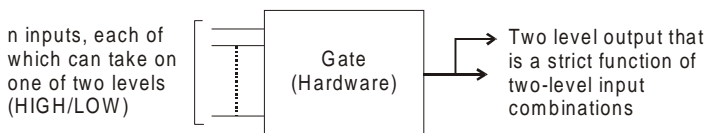
**Fig. 2.5.1**

The terms positive and negative are somewhat misleading since both signal values may be positive or both may be negative. Therefore, it is not signal polarity that determines the type of logic, but rather the assignment of logic values according to the relative amplitudes of the signals.

The effect of changing from one logic designation to the other equivalent to complementing the logic function because of the principle of duality of Boolean algebra.

**2.5.2 Gate Definition**

A ‘gate’ is defined as a multi-input ( $\geq 2$ ) hardware device that has a two-level output. The output level (1-H/0-L) of the gate is a strict and repeatable function of the two-level (1-H/0-L) combinations applied to its inputs. Fig. 2.5.2 shows a general model of a gate.



**Fig. 2.5.2** The general model of a gate.

The term “logic” is usually used to refer to a decision making process. A logic gate, then, is a circuit that can decide to say yes or no at the output based upon inputs.

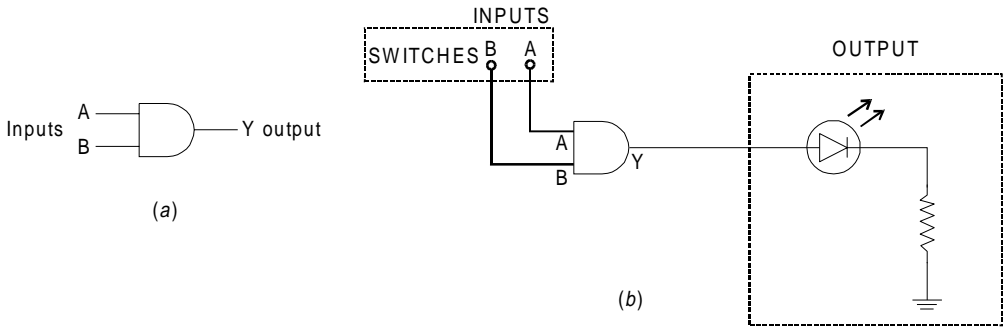
We apply voltage as the input to any gate, therefore the Boolean (logic) 0 and 1 do not represent actual number but instead represent the state of a voltage variable or what is called its logic level. Sometimes logic 0 and logic 1 may be called as shown in table below:

**Table 2.5.2**

<i>Logic 0</i>	<i>Logic 1</i>
False	True
Off	On
Low	High
No	Yes
Open switch	Close switch

### 2.5.3 The AND Gate

The AND gate is sometimes called the “all or nothing gate”. To show the AND gate we use the logic symbol in Fig. 2.5.3(a). This is the standard symbol to memorize and use from now on for AND gates.



**Fig. 2.5.3** (a) AND Gate logic symbol. (b) Practical AND gate circuit.

Now, let us consider Fig. 2.5.3(b). The AND gate in this figure is connected to input switches A and B. The output indicator is an LED. If a low voltage (Ground, GND) appears at inputs, A and B, then the output LED is not bit. This situation is illustrated in table (). Line 1 indicates that if the inputs are binary 0 and 0, then the output will be binary 0. Notice that only binary 1s at both A and B will produce a binary 1 at the output.

**Table 2.5.3 AND Truth Table**

INPUTS				OUTPUTS	
A		B		Y	
Switch Voltage	Binary	Switch Voltage	Binary	Light	Binary
Low	0	Low	0	No	0
Low	0	High	1	No	0
High	1	Low	0	No	0
High	1	High	1	Yes	1

It is a +5V compared to GND appearing at A, B, or Y that is called a binary 1 or a HIGH voltage. A binary 0, or Low voltage, is defined as a GND voltage (near 0V compared to GND) appearing at A, B or Y. We are using positive logic because it takes a positive +5V to produce what we call a binary 1.

The truth table is said to describe the AND function. The unique output from the AND gate is a HIGH only when all inputs are HIGH.

Fig. 2.5.3 (c) shows the ways to express that input A is ANDed with input B to produce output Y.

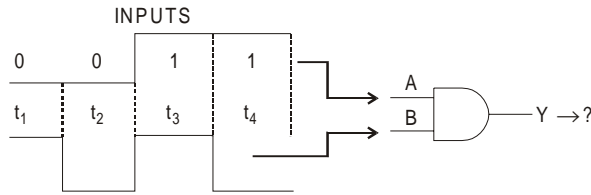
Boolean Expression	AND Symbol ↑ $A \cdot B = Y$															
Logic Symbol																
Truth Table	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1
A	B	Y														
0	0	0														
0	1	0														
1	0	0														
1	1	1														

**Fig. 2.5.3** (c)

**Pulsed Operation**

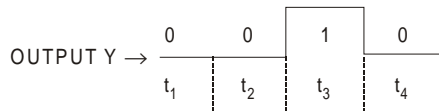
In many applications, the inputs to a gate may be voltage that change with time between the two logic levels and are called as pulsed waveforms. In studying the pulsed operation of an AND gate, we consider the inputs with respect to each other in order to determine the output level at any given time. Following example illustrates this operation:

**Example.** Determine the output  $Y$  from the AND gate for the given input waveform shown in Fig. 2.5.3(d).



**Fig. 2.5.3 (d)**

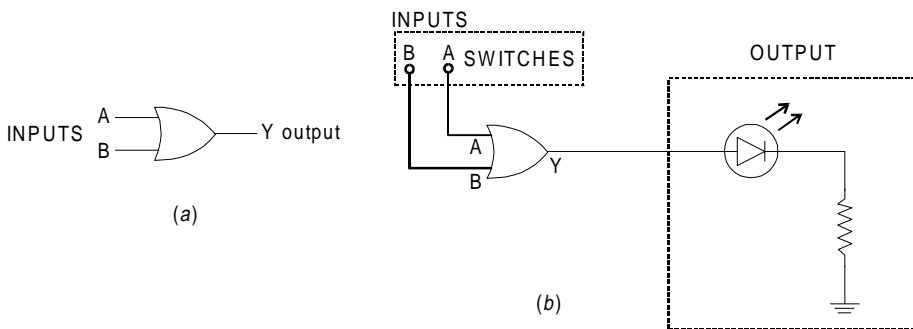
**Solution.** The output of an AND gate is determined by realizing that it will be high only when both inputs are high at the same time. For the inputs the outputs is high only during  $t_3$  period. In remaining times, the outputs is 0 as shown in Fig. 2.5.3(e).



**Fig. 2.5.3 (e)**

**2.5.4 The OR Gate**

The OR gate is sometimes called the “any or all gate”. To show the OR gate we use the logical symbol in Fig. 2.5.4(a).



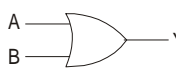
**Fig. 2.5.4 (a)** OR gate logic symbol. **(b)** Practical OR gate circuit.

A truth-table for the ‘OR’ gate is shown below according to Fig. 2.5.4(b). The truth-table lists the switch and light conditions for the OR gate. The unique output from the OR gate is a LOW only when all inputs are low. The output column in Table (2.5.4) shows that only the first line generates a 0 while all others are 1.

**Table 2.5.4**

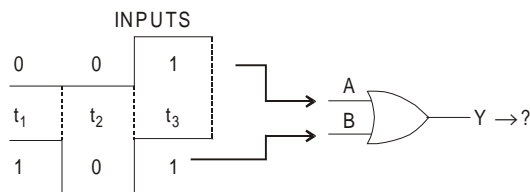
INPUTS				OUTPUTS	
A		B		Y	
Switch	Binary	Switch	Binary	Light	Binary
Low	0	Low	0	No	0
Low	0	High	1	No	0
High	1	Low	0	No	0
High	1	High	1	Yes	1

Fig. 2.5.4(c) shows the ways to express that input A is ORed with input B to produce output Y.

Boolean Expression	OR Symbol ↑ $A + B = Y$															
Logic Symbol																
Truth Table	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	1
A	B	Y														
0	0	0														
0	1	1														
1	0	1														
1	1	1														

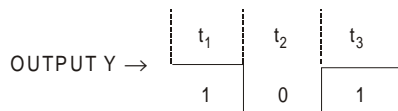
**Fig. 2.5.4 (c)**

**Example.** Determine the output Y from the OR gate for the given input waveform shown in Fig. 2.5.4(d).



**Fig. 2.5.4 (d)**

**Solution.** The output of an OR gate is determined by realizing that it will be low only when both inputs are low at the same time. For the inputs the outputs is low only during period  $t_2$ . In remaining time output is 1 as shown in Fig. 2.5.4(e).

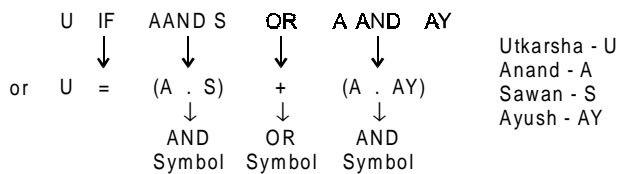


**Fig. 2.5.4 (e)**

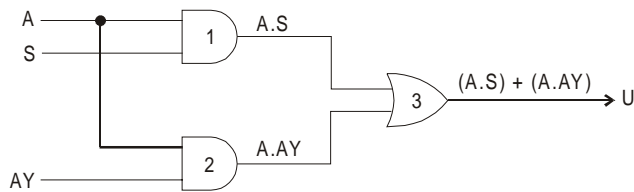
We are now familiar with AND and OR gates. At this stage, to illustrate at least in part how a word statement can be formulated into a mathematical statement (Boolean expression) and then to hardware network, consider the following example:

**Example.** *Utkarsha will go to school if Anand and Sawan go to school, or Anand and Ayush go to school.*

**Solution.** → This statement can be symbolized as a Boolean expression as follows:



The next step is to transform this Boolean expression into a Hardware network and this is where AND and OR gates are used.



The output of gate 1 is high only if both the inputs A and S are high (mean both Anand and Sawan go to school). This is the first condition for Utkarsha to go to school.

The output of gate 2 is high only if both the inputs A and A.Y are high (means both Anand and Ayush go to school). This is the second condition for Utkarsha to go to school.

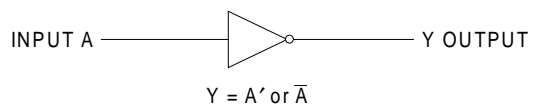
According to example atleast one condition must be fulfilled in order that Utkarsha goes to school. The output of gate 3 is high when any of the input to gate 3 is high means at least one condition is fulfilled or both the inputs to gate 3 are high means both the conditions are fulfilled.

The example also demonstrates that Anand has to go to school in any condition otherwise Utkarsha will not go to school.

### 2.5.5 The Inverter and Buffer

Since an Inverter is a single input device, it performs no logic interaction function between two variables, and to say that merely changing a voltage level constitute a logic operation would be misleading. Therefore we define it as an Inverter function rather than a logic operator like the AND and OR operators. The NOT circuit performs the basic logical function called inversion or complementation. That is why, it is also known as Inverter. The NOT circuit has only input and one output. The purpose of this gate is to give an output that is not the same as the input. When a HIGH level is applied to an inverter, a LOW level appears at its output and vice versa. The logic symbol for the inverter is shown in Fig. 2.5.5(a).

If we were to put in a logic at 1 and input A in Fig. 2.5.5(a), we would get out the opposite, or a logical 0, at output Y.



**Fig. 2.5.5 (a)** A logic symbol and Boolean expression for an inverter.

The truth-table for the inverter is shown in Fig. 2.5.5(b). If the voltage at the input of the inverter is LOW, then the output voltage is HIGH, if the input voltage is HIGH, then the output is LOW. We can say that output is always negated. The terms “negated”, “complemented” and “inverted”, then mean the same things.

INPUT		OUTPUT	
A		B	
Voltages	Binary	Voltages	Binary
LOW	0	HIGH	1
HIGH	1	LOW	0

Fig. 2.5.6 (b) Truth-table for an inverter.

Now consider the logic diagram as shown in Fig. 2.5.5(c), that shows an arrangement where input A is run through two inverters. Input A is first inverted to produce a “not A” ( $\bar{A}$ ) and then inverted a second time for “double not A” ( $\bar{\bar{A}}$ ). In terms of binary digits, we find that when the input 1 is inverted twice, we end up with original digit. Therefore, we find  $\bar{\bar{A}} = A$ .

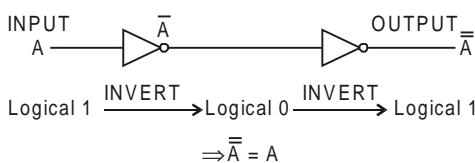


Fig. 2.5.5 (c) Effect of double inverting.

The symbol shown in figure is that of a non-inverting buffer/driver. A buffer produces the transfer function but does not produce any logical operation, since the binary value of the output is equal to the binary value of the input. The circuit is used merely for power amplification of the signal and is equivalent to two inverters connected in cascade. Fig. 2.5.5(e) shows the T.T. for the buffer.

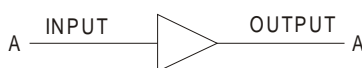


Fig. 2.5.5 (d) Non-inverting buffer/driver logic symbol.

INPUT		OUTPUT	
A		B	
Voltages	Binary	Voltage	Binary
HIGH	1	HIGH	1
LOW	0	LOW	0

Fig. 2.5.5 (e) Truth table for buffer.

**Example.** Determine the output Y from the inverter for the given input waveform shown in Fig. (2.5.5 f).

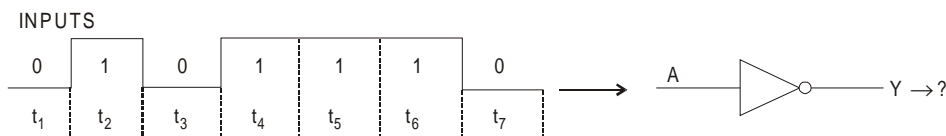
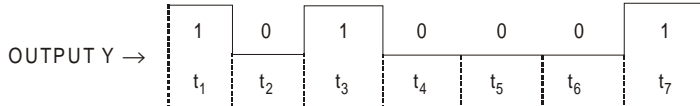


Fig. 2.5.5 (f)



**Solution.** The output of an Inverter is determined by realizing that it will be high when input is low and it will be low when input is high.



### 2.5.6 Other Gates and Their Functions

The AND, OR, and the inverter are the three basic circuits that make up all digital circuits. Now, it should prove interesting to examine the other 14 possible output specification (except AND and OR) for an arbitrary two-input gate.

Consider Table (2.5.6.)

**Table 2.5.6: Input/Output specifications for the 16 possible outputs derived from two-input gates**

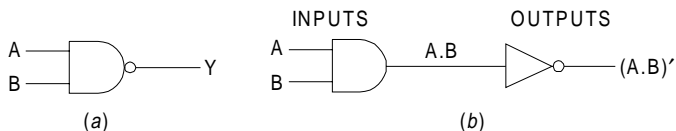
A	B	$\overline{\text{GND}}$	F <sub>1</sub>	F <sub>2</sub>	F <sub>3</sub>	F <sub>4</sub>	F <sub>5</sub>	F <sub>6</sub>	F <sub>7</sub>	F <sub>8</sub>	F <sub>9</sub>	F <sub>10</sub>	F <sub>11</sub>	F <sub>12</sub>	F <sub>13</sub>	F <sub>14</sub>	+V
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
			↓					↓	↓	↓	↓					↓	
			N					E	N	A	E					N	
			O					X	A	N	X					O	
			R					O	N	D	N					R	
								R	D		O						
											R						

Scanning the table for gates that exhibit the Boolean AND and OR operator, we see that F<sub>1</sub> (NOR), F<sub>7</sub> (NAND), F<sub>8</sub> (AND) and F<sub>14</sub> (OR) are the only outputs from gates which manifest the AND and OR operators. *We shall see very shortly that both NAND and NOR gates can be used as AND and OR gates.* Because of this, they are found in integrated circuit form. All the rest are more complex and deemed unuseful for AND/OR implementation and are not normally found in gate form, with two exceptions. They are F<sub>6</sub> and F<sub>9</sub>. F<sub>6</sub> is the Input/output specification for a gate called the EXCLUSIVE OR gate and F<sub>9</sub> is the specification for the COINCIDENCE, EQUIVALENCE, or EXNOR gate, also referred to as an EXCLUSIVE NOR.

### 2.5.7 Universal Gates

**NAND and NOR gates.** The NAND and NOR gates are widely used and are readily available from most integrated circuit manufacturers. A major reason for the widespread use of these gates is that they are both *UNIVERSAL gates*, universal in the sense that both can be used for AND operators, OR operators, as well as Inverter. Thus, we see that a complex digital system can be completely synthesized using only NAND gates or NOR gates.

**The NAND Gate.** The NAND gate is a NOT AND, or an inverted AND function. The standard logic symbol for the NAND gate is shown in Fig. (2.5.7a). The little invert bubble (small circle) on the right end of the symbol means to invert the output of AND.



**Fig. 2.5.7** (a) NAND gate logic symbol (b) A Boolean expression for the output of a NAND gate.

Figure 2.5.7(b) shows a separate AND gate and inverter being used to produce the NAND logic function. Also notice that the Boolean expression for the AND gate,  $(A.B)$  and the NAND  $(A.B)'$  are shown on the logic diagram of Fig. 2.5.7(b).

The truth-table for the NAND gate is shown in Fig. 2.5.7(c). The truth-table for the NAND gate is developed by inverting the output of the AND gate. The unique output from the NAND gate is a LOW only when all inputs are HIGH.

INPUT		OUTPUT	
A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

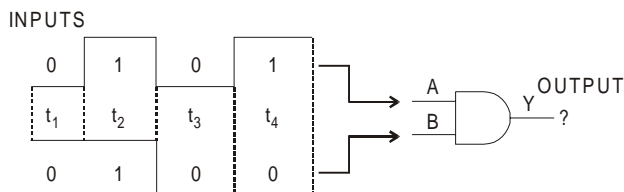
**Fig. 2.5.7** (c) Truth-table for AND and NAND gates.

Fig. 2.5.7 (d) shows the ways to express that input A is NANDed with input B yielding output Y.

Boolean Expression	$\overline{A \cdot B} = Y$ or $\overline{AB} = Y$ (NOT Symbol) $A \cdot B = Y$ (AND Symbol)															
Logic Symbol																
Truth Table	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	Y	0	0	1	0	1	1	1	0	1	1	1	0
A	B	Y														
0	0	1														
0	1	1														
1	0	1														
1	1	0														

**Fig. 2.5.7** (d)

**Example.** Determine the output Y from the NAND gate from the given input waveform shown in Fig. 2.6.7 (e).



**Fig. 2.5.7** (e)

**Solution.** The output of NAND gate is determined by realizing that it will be low only when both the inputs are high and in all other conditions it will be high. The output Y is shown in Fig. 2.5.7(f).

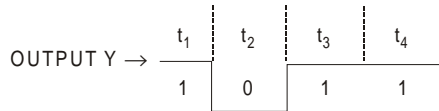


Fig. 2.5.7 (f)

**The NAND gate as a UNIVERSAL Gate**

The chart in Fig. 2.5.7(g) shows how you would wire NAND gates to create any of the other basic logic functions. The logic function to be performed is listed in the left column of the table; the customary symbol for that function is listed in the center column. In the right column, is a symbol diagram of how NAND gates would be wired to perform the logic function.

Logic Function	Symbol	Circuit using NAND gates only
Inverter		
AND		
OR		

Fig. 2.5.7 (g)

**The NOR gate.** The NOR gate is actually a NOT OR gate or an inverted OR function. The standard logic symbol for the NOR gate is shown in Fig. 2.5.7(h)

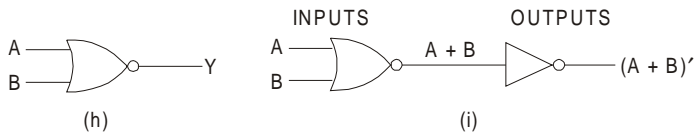


Fig. 2.5.7 (h) NOR gate logic symbol (i) Boolean expression for the output of NOR gate.

Note that the NOR symbol is an OR symbol with a small invert bubble on the right side. The NOR function is being performed by an OR gate and an inverter in Fig. 2.5.7(i). The Boolean function for the OR function is shown  $(A + B)$ , the Boolean expression for the final NOR function is  $(A + B)$ .

The truth-table for the NOR gate is shown in Fig. 2.5.7(j). Notice that the NOR gate truth table is just the complement of the output of the OR gate. The unique output from the NOR gate is a HIGH only when all inputs are LOW.

INPUTS		OUTPUTS	
A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Fig. 2.5.7 (j) Truth-table for OR and NOR gates.

Figure 2.5.7(k) shows the ways to express that input A is ORed with input B yielding output Y.

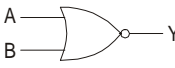
Boolean Expression	$\overline{A + B} = Y \quad \text{or} \quad (A + B)' = Y$ <p style="text-align: center;"> <small>↓ NOT Symbol</small>  <small>↑ OR Symbol</small> </p>															
Logic Symbol																
Truth Table	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	0
A	B	Y														
0	0	1														
0	1	0														
1	0	0														
1	1	0														

Fig. 2.5.7 (k)

**Example.** Determine the output Y from the NOR gate from the given input waveform shown in Fig. 2.5.7(l).

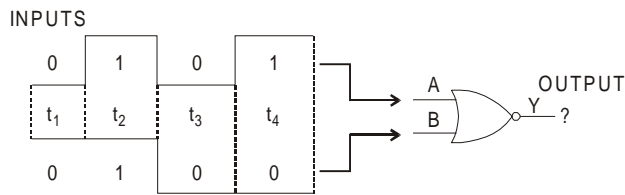


Fig. 2.5.7 (l)

**Solution.** The output of NOR gate is determined by realizing that it will be HIGH only when both the inputs are LOW and in all other conditions it will be high. The output Y is shown in Fig. 2.5.7(m).

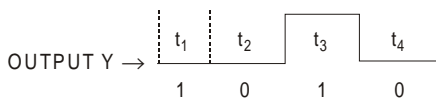


Fig. 2.5.7 (m)

**The NOR gate as a UNIVERSAL gate.**

The chart in Fig. 2.5.7(n) shows how would your wire NOR gates to create any of the other basic logic functions.

Logic Function	Symbol	Circuit using NOR gates only
Inverter		
AND		
OR		

Fig. 2.5.7 (n)

**2.5.8 The Exclusive OR Gate**

The exclusive OR gate is sometimes referred to as the “Odd but not the even gate”. It is often shortend as “XOR gate”. The logic diagram is shown in Fig. 2.5.8 (a) with its Boolean expression. The symbol  $\oplus$  means the terms are XORed together.

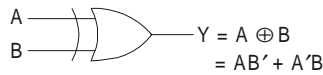


Fig. 2.5.8 (a)

The truth table for XOR gate is shown in Fig. 2.5.8 (b). Notice that if any but not all the inputs are 1, then the output will be 1. ‘The unique characteristic of the XOR gates that it produces a HIGH output only when the odd no. of HIGH inputs are present.’

INPUTS		OUTPUTS
A	B	$A \oplus B = AB' + A'B$
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 2.5.8 (b)

To demonstrate this, Fig. 2.5.8 (c) shows a three input XOR gate logic symbol and the truth table Fig. 2.5.8 (d). The Boolean expression for a three input XOR gate can be written as

$$\begin{aligned}
 Y &= (A \oplus B) \oplus C \\
 &= (AB' + A'B) \oplus C \\
 X &= AB' + A'B
 \end{aligned}$$

Now Let

⇒ We have  $X \oplus C$

$$= XC' + X'C$$

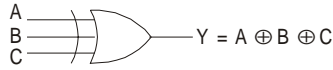


Fig. 2.5.8 (c)

INPUTS			OUTPUTS
A	B	C	Y
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

Fig. 2.5.8 (d)

Putting the value of X, we get

$$Y = (AB' + A'B)C + (AB' + AB).C$$

$$Y = AB'C' + A'BC' + A'B'C + ABC$$

The HIGH outputs are generated only when odd number of HIGH inputs are present (see T.T.)

*'This is why XOR function is also known as odd function'.*

Fig. 2.5.8 (e) shows the ways to express that input A is XORed with input B yielding output Y.

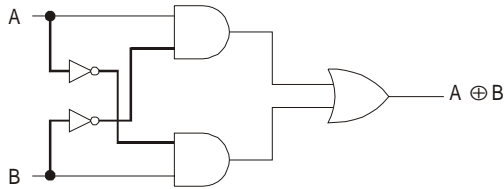
Boolean Expression	$A \oplus B = Y$ 															
Logic Symbol																
Truth Table	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	0
A	B	Y														
0	0	0														
0	1	1														
1	0	1														
1	1	0														

Fig. 2.5.8 (e)

The XOR gate using AND OR-NOT gates.

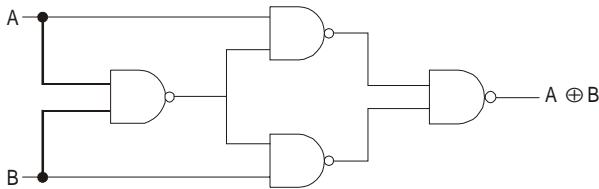
we know

$$A \oplus B = AB' + A'B$$

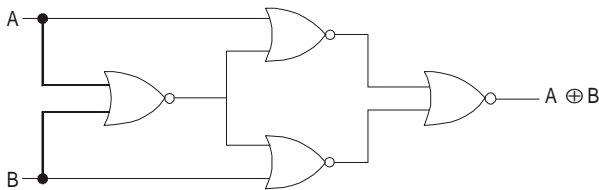


As we know NAND and NOR are universal gates means any logic diagram can be made using only NAND gates or using only NOR gates.

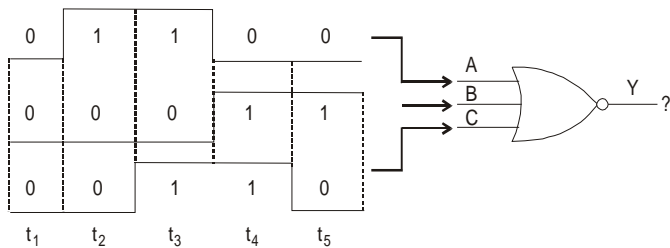
**XOR gate using NAND gates only.**



XOR using NOR gates only. The procedure for implementing any logic function using only universal gate (only NAND gates or only NOR gates) will be treated in detail in section 2.6.

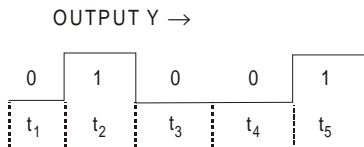


**Example.** Determine the output  $Y$  from the XOR gate from the given input waveform shown in Fig. 2.5.8 (f).



**Fig. 2.5.8 (f)**

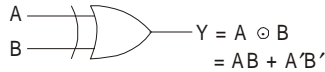
**Solution.** The output XOR gate is determined by realizing that it will be HIGH only when the odd number of high inputs are present therefore the output  $Y$  is high for time period  $t_2$  and  $t_5$  as shown in Fig. 2.5.8 (g).



**Fig. 2.5.8 (g)**

### 2.5.9 The Exclusive NOR gate

The Exclusive NOR gate is sometimes referred to as the ‘COINCIDENCE’ or ‘EQUIVALENCE’ gate. This is often shortened as ‘XNOR’ gate. The logic diagram is shown in Fig. 2.5.9 (a).



**Fig. 2.5.9 (a)**

Observe that it is the XOR symbol with the added invert bubble on the output side. The Boolean expression for XNOR is therefore, the invert of XOR function denoted by symbol  $\odot$ .

$$\begin{aligned}
 A \odot B &= (A \oplus B)' \\
 &= (AB' + A'B)' \\
 &= (A' + B) \cdot (A + B') \\
 &= AA' + A'B' + AB + BB' \\
 &= AB + A'B'.
 \end{aligned}$$

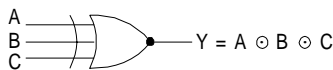
The truth table for XNOR gate is shown in Fig. 2.5.9 (b).

INPUTS		OUTPUTS
A	B	$A \odot B = AB + A'B'$
0	0	1
0	1	0
1	0	0
1	1	1

**Fig. 2.5.9 (b)**

Notice that the output of XNOR gate is the complement of XOR truth table.

‘The unique output of the XNOR gate is a LOW only when an odd number of input are HIGH’.



**Fig. 2.5.9 (c)**

INPUTS			OUTPUTS
A	B	C	Y
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

**Fig. 2.5.9 (d)**



To demonstrate this, Fig. 2.5.9 (c) shows a three input XNOR gate logic symbol and the truth-table 2.5.9 (d).

Figure 2.5.9 (e) shows the ways to express that input A is XNORed with input B yielding output Y.

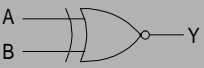
Boolean Expression	$A \odot B = Y$															
Logic Symbol																
Truth Table	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	1
A	B	Y														
0	0	1														
0	1	0														
1	0	0														
1	1	1														

Fig. 2.5.9 (e)

Now at this point, it is left as an exercise for the reader to make XNOR gate using AND-OR-NOT gates, using NAND gates only and using NOR gates only.

**Example.** Determine the output Y from the XNOR gate from the given input waveform shown in Fig. 2.5.9 (f).

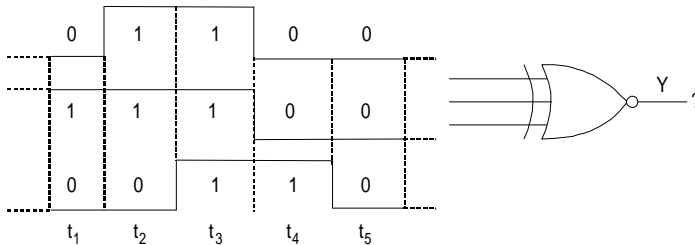


Fig. 2.5.9 (f)

**Solution.** The output of XNOR gate is determined by realizing that it will be HIGH only when the even-number of high inputs are present, therefore the output Y is high for time period  $t_2$  and  $t_5$  as shown in Fig. 2.5.9 (g).

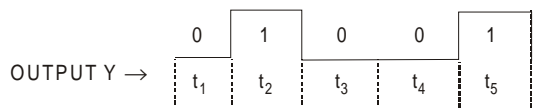


Fig. 2.5.9 (g)

### 2.5.10 Extension to Multiple Inputs in Logic Gates

The gates we have studied, except for the inverter and buffer can be extended to have more than two inputs.

A gate can be extended to have multiple inputs the binary operation it represents is commutative and associative.

The AND and OR operations defined in Boolean algebra, possesses these two properties. For the NAD function, we have

$$x.y = y.x \text{ Commutative}$$

and

$$x.(yz) = (x.y).z \text{ Associative}$$

Similarly for the OR function,

$$x + y = y + x \text{ Commutative}$$

and

$$(x + y) + z = x + (y + z) \text{ Associative}$$

It means that the gate inputs can be interchanged and that the AND and OR function can be extended to the or more variables.

The NAND and NOR operations are commutative but not associative. For example

$$*(x \uparrow y) = (y \uparrow x) \Rightarrow x' + y' = y' + x' = \text{commutative}$$

But

$$x \uparrow (y \uparrow z) \neq (x \uparrow y) \uparrow z$$

⇒

$$[x'.(y.z)'] \neq [(x.y)'.z]'$$

⇒

$$x' + yz \neq xy + z'$$

Similarly

$$**(x \downarrow y) = (y \downarrow x) \Rightarrow x'y' = y'x' \rightarrow \text{commutative}$$

But

$$x (y \downarrow z) \neq (x \downarrow y) \downarrow z$$

⇒

$$[x + (y + z)'] \neq [(x + y)' + z]$$

$$x'y + x'z \neq xz' + yz'$$

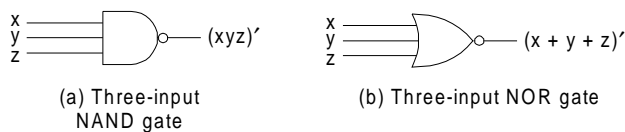
This difficulty can be overcome by slightly modifying the definition of operation. We define the multiple NAND (or NOR) gate as complemented AND (or OR) gate.

Thus by this new definition, we have

$$x \uparrow y \uparrow z = (xyz)'$$

$$x \downarrow y \downarrow z = (x + y + z)'$$

The graphic symbol for 3-input NAND and NOR gates is shown.



The exclusive–OR and equivalence gates are both commutative and associative and can be extended to more than two inputs.

The reader is suggested to verify that, both X-OR and X-NOR possess commutative and associative property.

---

\* NAND symbol

\*\* NOR symbol

**SOLVED EXAMPLES**

**Example 1.** Give the concluding statement of all the logic gates, we have studied in this chapter:

**Solution. AND:** The output is HIGH only when all the inputs are HIGH, otherwise output is LOW.

**OR:** The output is LOW only when all the inputs are LOW, otherwise output is HIGH.

**NOT:** The output is complement of input.

**NAND:** The output is LOW only when all the inputs are HIGH, otherwise the output is HIGH.

**NOR:** The output is HIGH only when all the inputs are LOW, otherwise output is LOW.

**EX-OR:** The output is HIGH only when both the inputs are not same, otherwise output is Low.

**OR**

The output is HIGH only when odd number of inputs are HIGH, otherwise output is LOW.

**EX-NOR:** The output is HIGH only when both the inputs are same, otherwise output is LOW.

**OR**

The output is HIGH only when even number of inputs are HIGH, otherwise output is LOW.

**Example 2.** Show how an EX-OR gate can be used as NOT gate or inverter:

**Solution.** The expression for NOT gate is

$$y = \bar{A} \text{ where } y = \text{output and } A = \text{input}$$

The expression for EX-OR gate is

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

where A and B are inputs.

In the expression of EX-OR we see that the first term  $\bar{A}B$  can give the complement of input A, if B = 1 and second term  $AB = 0$ . So we connect the B input to logic HIGH (i.e., logic 1) to full fill the requirements above stated. i.e.,

from figure

$$y = \bar{A}.1 + A.0$$

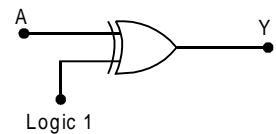
or

$$y = \bar{A} \text{ i.e., complement}$$

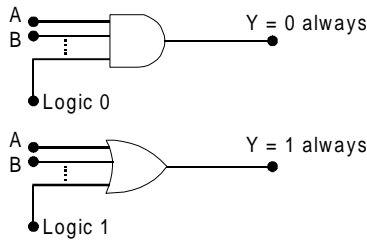
Thus above connection acts as inverter or NOT gate.

**Example 3.** Show, how an AND gate and an OR gate can be masked.

**Solution.** Masking of gates means disabling the gate. It is the process of connecting a gate in such a way that the output is not affected by any change in inputs i.e., the output remains fixed to a logic state irrespective of inputs.



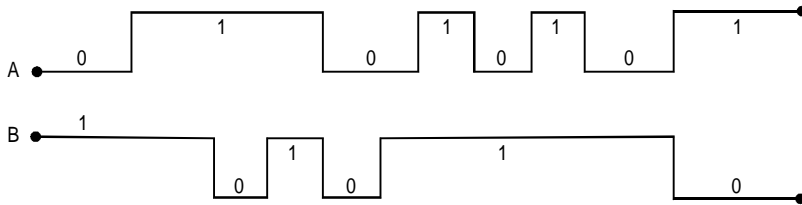
**Fig. Ex. 2:** EX-OR Gate connected as NOT Gate



**Fig. Ex. 3:** Masking of AND gate and OR gate

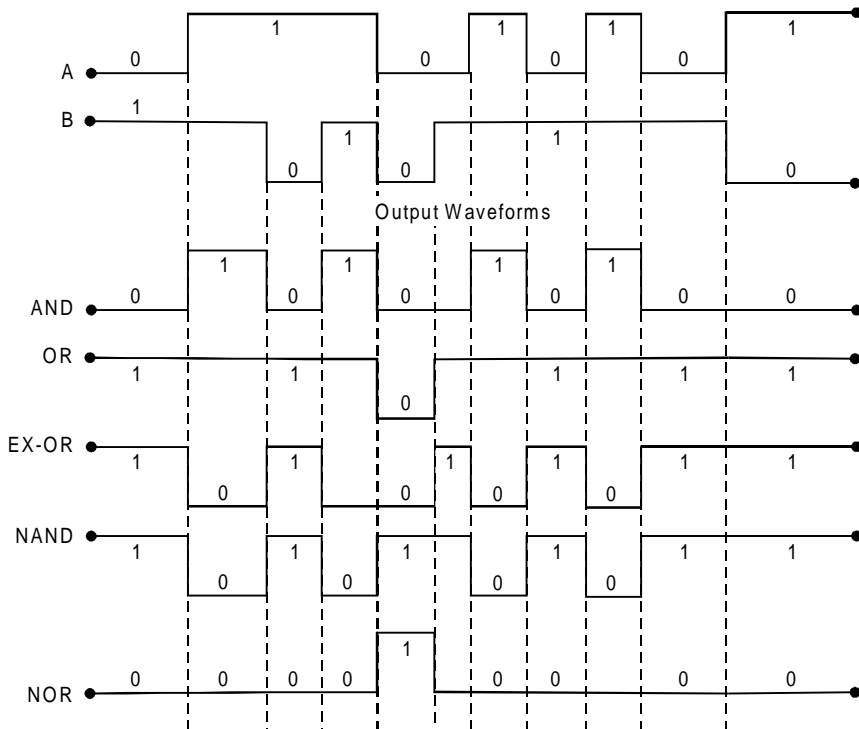
AND gate can be masked by connecting one of its input to logic LOW (i.e. logic 0) and Or gate can be marked by connecting one of its input to logic HIGH (i.e. logic 1)

**Example 4.** Below shown waveforms are applied at the input of 2-input logic gate.



Draw the output waveform if the gate is (a) AND gate (b) OR gate (c) EX-OR gate (d) NAND gate (e) NOR gate.

**Solution.** The waveforms can be drawn by recalling the concluding statements of logic gates given in earlier examples.



**Example 5.** Show how a 2 input AND gate, OR gate, EX-OR gate and EX-NOR gate can be made transparent.

**Solution.** Transparent gate means a gate that passes the input as it is to the output i.e. the output logic level will be same as the logic level applied at input. Let the two inputs are A and B and output is  $y$ . We will connect the gates in such a way that it gives  $y = A$ .

For **AND gate** we have expression

$$y = A.B$$

if  $B = 1$

$$y = A$$

So connect input B to logic 1 for **OR gate** we have

$$y = A + B$$

if  $B = 0$

$$y = A$$

So connect input B to logic 0 for **EX-OR gate** we have

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

if  $B = 0,$

then  $\bar{A}B = A,$  and  $\bar{A}\bar{B} = 0$

and  $y = A$

Hence connect input B to logic 0 for **EX-NOR gate** we have

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

if  $B = 1,$

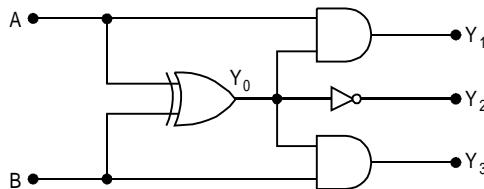
then  $\bar{\bar{A}B} = 0$  and  $\bar{A}\bar{B} = A$

so  $y = A$

Hence connect input B to logic 1

If we take multiple input logic gates then connecting them as above is called *enabling* a gate then connecting them as above is called *enabling* a gate e.g. if we take three input (A, B, C) AND gate and connect the input (A, B, C) AND gate and connect the input C to logic 1, then output  $y = A.B$ . Thus the gate is enabled.

**Example 6.** Determine the purpose of below shown digital circuit.



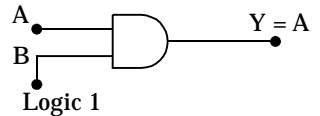
**Solution.** From the figure we see that

$$y_0 = A \oplus B = \bar{A}B + A\bar{B}$$

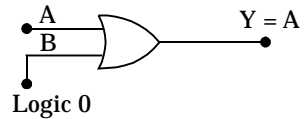
$$y_1 = A.y_0$$

$$y_2 = \bar{y}_0$$

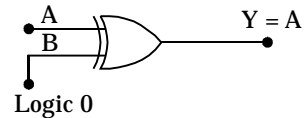
$$y_3 = B.y_0$$



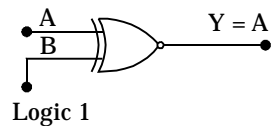
**Fig. Ex. 5 (a):** Transparent AND gate



**Fig. Ex. 5 (b):** Transparent OR gate



**Fig. Ex. 5 (c):** Transparent EX-OR gate



**Fig. Ex. 5 (d):** Transparent EX-NOR gate

Now we draw three truth tables, one for each of the outputs  $y_1$ ,  $y_2$ , and  $y_3$  to determine the purpose of the circuit.

(i) From the table (i), it is evident that  $y_1 = 1$ , only when  $A = 1$  and  $B = 0$ . It means that  $y_1$  is HIGH only when  $A > B$ , as shown in third row of Table (i)

**Table (i)**

A	B	$Y_0$	$Y_1$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	0

(ii) It is evident from Table (ii) that  $y_2 = 1$  if  $A = B = 0$  and  $A = B = 1$ . Thus  $y_2$  goes HIGH only when  $A = B$ , as shown by first and last row of Table (ii).

**Table (ii)**

A	B	$Y_0$	$Y_1$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

(iii) It is evident from Table (iii) that  $y_3 = 1$  if  $A = 0$  and  $B = 1$ . Thus  $y_3$  goes HIGH only when  $A < B$  (or  $B > A$ ), as shown by the second row of table (iii).

**Table (iii)**

A	B	$Y_0$	$Y_1$
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Thus from the above discussion it can be concluded that the given circuit is 1-bit data comparator. In this circuit when HIGH,  $y_1$  indicates  $A > B$ ,  $y_2$  indicate the equality two datas, and  $y_3$  indicates  $A < B$ .

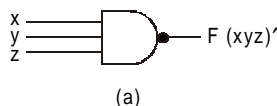
## 2.6 NAND AND NOR IMPLEMENTATION

In Section 2.5.7 we have seen that NAND and NOR are universal gates. Any basic logic function (AND, OR and NOT) can be implemented using these gates only. Therefore digital circuits are more frequently constructed with NAND or NOR gates that with AND, OR and NOT gates. Moreover NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families. Because of this prominence, rules and procedures have been developed for implementation of Boolean functions by using either NAND gates only or NOR gates only.

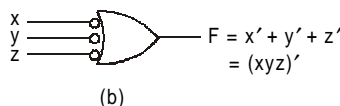
### 2.6.1 Implementation of a Multistage (or Multilevel) Digital Circuit using NAND Gates Only

To facilitate the implementation using NAND gates only, we first define two graphic symbols for these gates as follows-shown in Fig. 2.6.1(a) and (b).

(a) The AND-invert symbol



(b) The invert-OR symbol



**Fig. 2.6.1** (a) (b)

This symbol we have been defined in Section (2.5). It consists of an AND graphic symbol followed by a small circle.

This is an OR graphic symbol preceded by small circles in all the inputs.

To obtain a multilevel NAND circuit from a given Boolean function, the procedure is as follows:

1. From the given Boolean function, draw the logic diagram using basic gates (AND, OR and NOT). In implementing digital circuit, it is assumed that both normal and inverted inputs are available. (e.g., If  $x$  and  $x'$  both are given in function, we can apply them directly that is there is no need to use an inverter or NOT gate to obtain  $x'$  from  $x$ ).
2. Convert all AND gates to NAND using AND-invert symbol.
3. Convert all OR gates to NAND using Invert-OR symbol.
4. Since each symbol used for NAND gives inverted output, therefore it is necessary to check that if we are getting the same value as it was at input. [For example if the input is in its normal form say  $x$ , the output must also be  $x$ , not  $x'$  (inverted or complemented value). Similarly if input is in its complemented form say  $x'$ , the output must also be  $x'$ , not  $x$  (normal value)].

If it is not so, then for every small circle that is not compensated by another small circle in the same line, insert an inverter (i.e., one input NAND gate) or complement the input variable.

Now consider a Boolean function to demonstrate the procedure:

$$Y = A + (B + C')(D'E + F)$$

**Step 1.** We first draw the logic diagram using basic gates shown in Fig. 2.6.1 (c). (It is assumed that both normal or complemented forms are available i.e., no need of inverter).

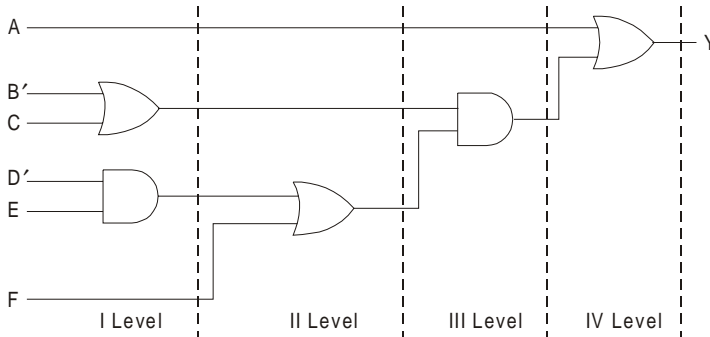


Fig. 2.6.1 (c)

There are four levels of gating in the circuits.

**Step 2 and 3**

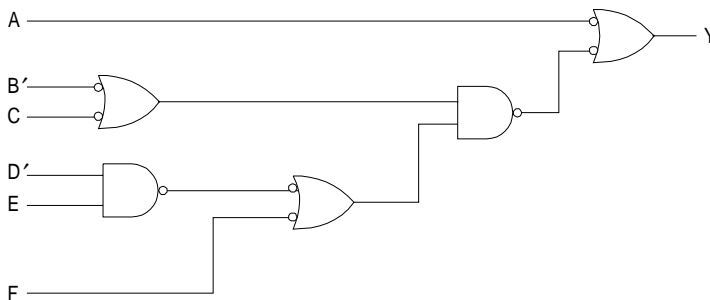


Fig. 2.6.1 (d)

Here, we have converted all AND gates to NAND using AND-invert and all OR gates to NAND using invert OR symbol shown in Fig. 2.6.1 (d).

**Step 4.** From the above figures obtained from step 2 and 3, it is very clear that only two inputs D' and E are emerging in the original forms at the output. Rest *i.e.*, A, B', C and F are emerging as the complement of their original form. So we need an inverter after inputs A, B', C and F or alternatively we can complement these variables as shown in Fig. 2.6.1 (e).

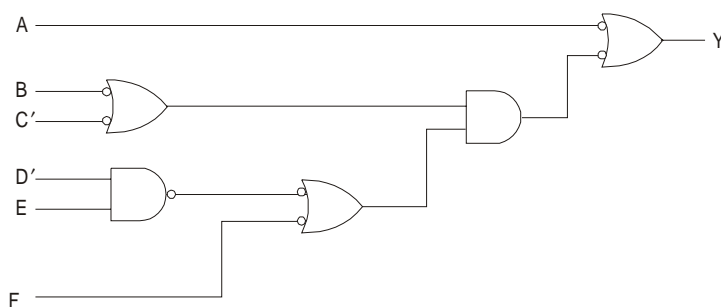


Fig. 2.6.1 (e)

Now because both the symbols AND-invert and invert-OR represent a NAND gate, Fig. 2.6.1 (e) can be converted into one symbol diagram shown in Fig. 2.6.1 (f). The two symbols were taken just for simplicity of implementation.

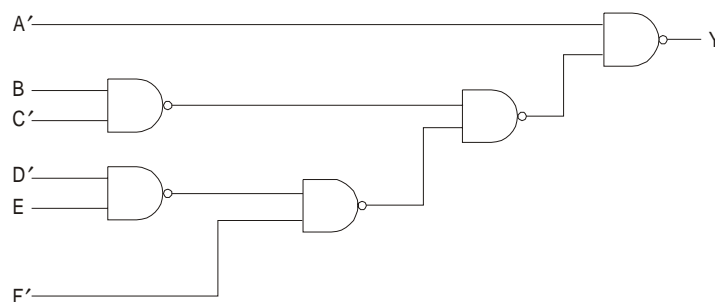


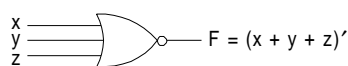
Fig. 2.6.1 (f)

After a sufficient practice, you can directly implement any Boolean function as shown in Fig. 2.6.1 (f).

### 2.6.2 Implementation of a Multilevel digital circuit using NOR gates only

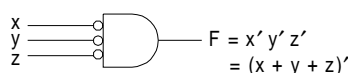
We first define two basic graphic symbols for NOR gates as shown in Fig. 2.6.2 (a) and (b).

(a) The OR-invert symbol



(a) This is an OR graphic symbol followed by a small circle.

(b) The invert-AND symbol



(b) This is an AND graphic symbol preceded by small circles in all the inputs.

Fig. 2.6.2 (a) (b)



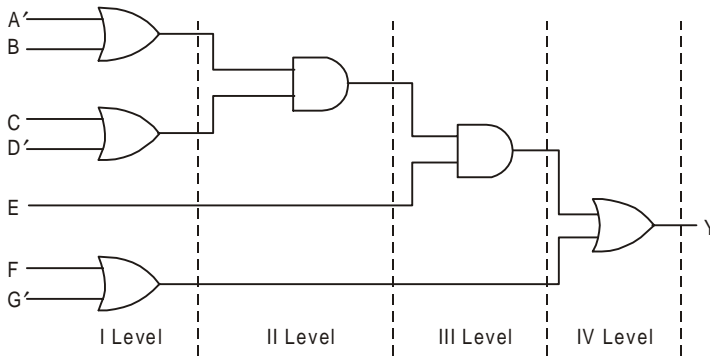
Procedure to obtain a multilevel NOR circuit from a given Boolean function is as follows:

1. Draw the AND-OR logic diagram from the given expression. Assume that both normal and complemented forms are available.
2. Convert all OR gates to NOR gates using OR-invert symbol.
3. Convert all AND gates to NOR gates using invert-AND symbol.
4. Any small circle that is not complement by another small circle along the same line needs an inverter (one input NOR gate) or the complementation of input variable.

Consider a Boolean expression to demonstrate the procedure:

$$Y = [(A' + B).(C + D')]E + (F + G')$$

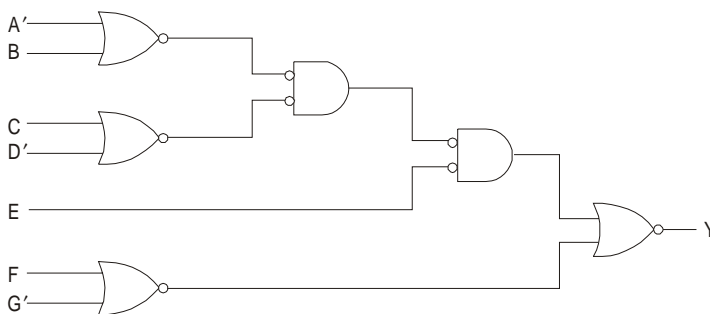
**Step 1.** We first draw the AND-OR logic diagram shown in Fig. 2.6.2 (c).



**Fig. 2.6.2 (c)**

There are four levels of gating in the circuit.

**Step 2 and 3.** Here we have to convert all OR gates to NOR using OR-invert and all AND gates to NOR using invert AND symbol. This is shown in Fig. 2.6.2(d).



**Fig. 2.6.2 (d)**

**Step 4.** From Fig. 2.6.2 (d), it is clear that all the input variables are emerging in the same form at the output Y as they were at input. Therefore there is no need of inverter at inputs or complementing the input variables.

Here again, both symbols OR-invert and invert-AND represent a NOR gate, so the diagram of Fig. 2.6.2 (d) can be converted into one symbol shown in Fig. 2.6.2 (e).

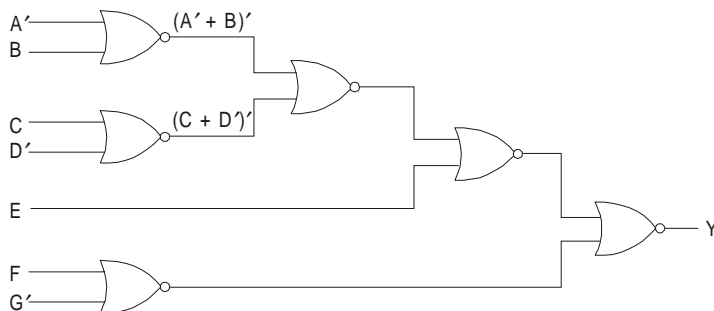


Fig. 2.6.2 (e)

2.7 EXERCISES

1. Write Boolean equations for  $F_1$  and  $F_2$ .

A	B	C	$F_1$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

A	B	C	$F_2$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

- Consider 2-bit binary numbers, say A, B and C, D. A function X is true only when the two numbers are different construct a truth table for X.
- Draw truth table and write Boolean expression for the following:
  - X is a 1 only if A is a  $\perp$  and B is a  $\perp$  or if A is 0 and B is a 0.
  - X is a 0 if any of the three variables A, B and C are 1's. X is a  $\perp$  for all other conditions.
- Prove the following identities by writing the truth tables for both sides:
  - $A.(B + C) = (A.B) + (A.C)$
  - $(A.B.C)' = A' + B' + C'$
  - $A.(A + B) = A$
  - $A + A'B = A + B$
- Prove the following:
  - $(X + Y)(X + Y') = X$
  - $XY + X'Z + YZ = XY + X'Z$
  - $(X + Y') = X.Y$
  - $(X + Y)(X + Z) = X + Y + Z$
  - $(X + Y + Z)(X + Y + Z') = X + Y$

6. Without formally deriving can logic expression, deduct the value of each function W, X, Y, Z.

A	B	C	W	X	Y	Z
0	0	0	0	1	0	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	0	0
1	0	0	0	1	1	1
1	0	1	0	1	1	0
1	1	0	0	1	1	0
1	1	1	0	1	1	0

7. A large room has three doors, A, B and C, each with a light switch that can turn the room light ON or OFF. Flipping any switch will change the condition of the light. Assuming that the light switch is off when the switch variables have the values 0, 0, 0 write a truth table for a function LIGHT that can be used to direct the behaviour of the light. Derive a logic equation for light.
8. Use DeMorgan's theorem to complement the following Boolean expression.
- $Z = Z = x.(y + w.v)$
  - $Z = x.y.w + y(w' + v')$
  - $Z = x' + yY$
  - $F = (AB + CD)''$
  - $F = ((A + B') (C' + D))'$
  - $F = ((A + B') (C + D))'$
  - $F = [((ABC)' (EFG)')' + ((HIJ)' (KLM)')']'$
  - $F = [((A + B)' (C + D)' (E + F)' (G + H)')']'$
9. A certain proposition is true when it is not true that the conditions A and B both hold. It is also true when conditions A and B both hold but condition C does not. Is the proposition true when it is not true that conditions B and C both hold? Use Boolean algebra to justify your answer.
10. Define the following terms:
- Connical
  - Minterm
  - Mexterm
  - Sum-of-sums form
  - Product-of-sum form
  - Connical sum-of-product
  - Connical product-of-sums

11. Write the standard SOP and the standard POS form for the truth tables:

(a)

x	y	z	F
0	0	0	1
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

(b)

x	y	z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

12. Convert the following expressions into their respective connical forms:

- (a)  $AC + A'BD + ACD'$
- (b)  $(A + B + C')(A + D)$

13. Which of the following expressions is in sum of product form? Which is in product of sums form?

- (a)  $A + (B.D)'$
- (b)  $C.D'.E + F' + D$
- (c)  $(A + B).C$

14. Find the connical s-of-p form for the following logic expressions:

- (a)  $W = ABC + BC'D$
- (b)  $F = VXW'Y + W'XYZ$

15. Write down the connical s-of-p form and the p-of-s form for the logic expression whose TT is each of the following.

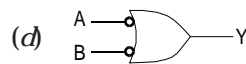
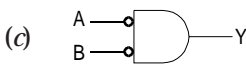
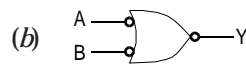
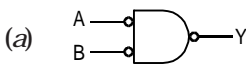
(a)

$x_1$	$y_2$	$z_3$	Z
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

(b)

W	X	Y	Z	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

- 16.** Convert the following expressions to sum-of-product forms:
- $AB + CD (AB' + CD)$
  - $AB (B'C' + BC)$
  - $A + B [AC + (B + C)' D]$
- 17.** Given a Truth table
- Express  $W_1$  and  $W_2$  as a sum of minterms
  - Express  $W_1$  and  $W_2$  as product of minterms
  - Simplify the sum of minterm expressions for  $W_1$  and  $W_2$  using Boolean algebra.
- 18.** Draw logic circuits using AND, OR and NOT gates to represent the following:
- $AB' + A'B$
  - $AB + A'B' + A'BC$
  - $A + B [C + D (B + C)']$
  - $A + BC' + D (E' + F')$
  - $[(AB)' (CD)']'$
- 19.** Produce a graphical realization of Inverter, AND, OR and XOR using:
- NAND gates
  - NOR gates
- 20.** Implement the Boolean function  
 $F = AB'CD' + A'BCD' + AB'C'D + A'BC'D$  with exclusive-OR and AND gates.
- 21.** Draw the logic circuits for the following functions.
- 22.** A warning busser is to sound when the following conditions apply:
- Switches A, B, C are on.
  - Switches A and B are on but switch C is off.
  - Switches A and C are on but switch B is off.
  - Switches B and C are on but switch A is off.
- Draw a truth table and obtain a Boolean expression for all conditions. Also draw the logic diagram for it using (i) NAND gates (ii) NOR gates. If the delay of a NAND gate is 15 ns and that of a NOR gate is 12 ns, which implementation is better?
- 23.** Obtain the following operations using only NOR gates.
- NOT
  - OR
  - AND
- 24.** Obtain the following operations using only NAND gates.
- NOT
  - AND
  - OR
- 25.** Find the operation performed for each of the gates shown in figure below, with the help the truth table.



- 26.** Write the expression for EX-OR operation and obtain
- The truth table
  - Realize this operation using AND, OR, NOT gates.
  - Realize this operation using only NOR gates.
- 27.** Verify that the (i) NAND (ii) NOR operations are commutative but not associate.
- 28.** Verify that the (i) AND (ii) OR (iii) EX-OR operations are commutative as well as associative.
- 29.** Prove that
- A positive logic AND operation is equivalent to a negative logic OR operation and vice versa.
  - A positive logic NAND operation is equivalent to a negative logic NOR operation and vice versa.
- 30.** Prove the logic equations using the Boolean algebraic (switching algebraic) theorems.
- $A + \overline{A}B + \overline{A}B = A + B$
  - $AB + \overline{A}B + \overline{A}B = \overline{A}B$
- Verify these equations with truth table approach.
- 31.** Prove De Morgan's theorems.
- 32.** Using NAND gates produce a graphical realization of
- Inverter
  - AND
  - OR
  - XOR
- 33.** Using NOR gates also produce a graphical realization of
- Inverter
  - AND
  - OR
- 34.** Prove  $(X + Y)(X + Y') = X$
- 35.**  $XY + X'Z + YZ = XY + X'Z$
- 36.** Prove the following:
- 37.** (a)  $(A + B)' = A.B$                       (b)  $(A + B)(A + C) = A + BC$
- 38.** Prove the identities:
- $A = (A + B)(A + B)'$
  - $A + B = (A + B + C)(A + B + C)'$
- 39.** Obtain the simplified expressions in s-of-p for the following Boolean functions:
- $xy + x'y'z' + x'y'z'$
  - $ABD + A'C'D' + A'B + A'CD' + AB'D'$

- (c)  $x'z + w'xy' + w(x'y + xy')$
- (d)  $F(x, y, z) = \Sigma(2, 3, 6, 7)$
- (e)  $F(A, B, C') (A + D)$

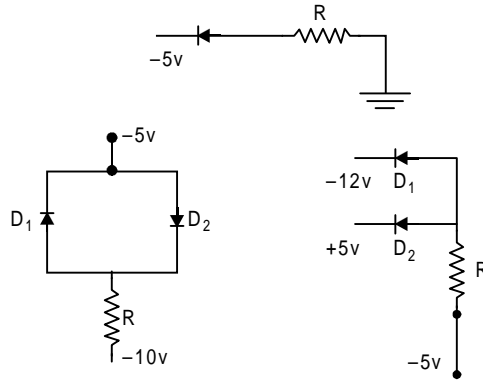
**40.** Convert the following expressions into their respective Canonical forms

- (a)  $AC + A'BD + ACD'$
- (b)  $(A + B + C) (A + D)$

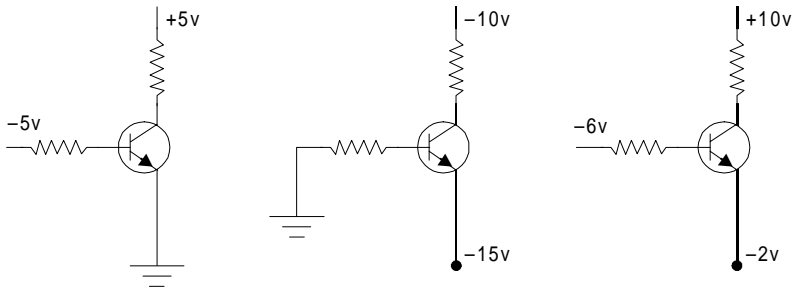
**41.** Write the standard SOP and the standard POS form for the truth tables

(a)				(b)			
<i>x</i>	<i>y</i>	<i>z</i>	$F_{(x, y, z)}$	<i>x</i>	<i>y</i>	<i>z</i>	$F_{(x, y, z)}$
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	1	1
1	0	0	1	1	0	0	1
1	0	1	0	1	0	1	1
1	1	0	0	1	1	0	0
1	1	1	1	1	1	1	1

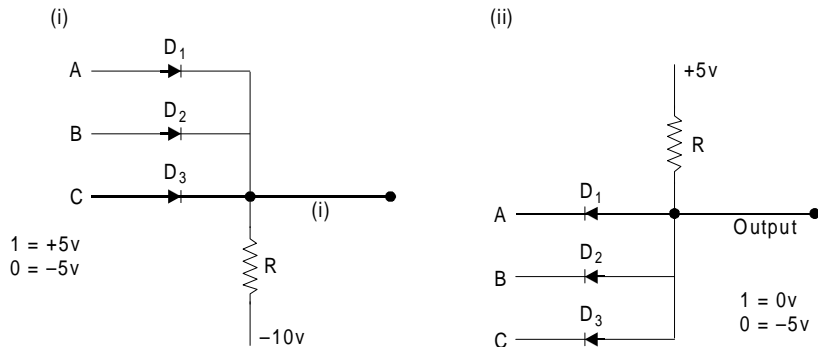
- 42.** Consider 2-bit binary numbers, say A, B and C, D. A function X is true only when the two numbers are different.
  - (a) Construct a truth table for X
  - (b) Construct a four-variable K-Map for X, directly from the word definition of X
  - (c) Derive a simplified logic expression for X.
- 43.** Show an implementation of the half-adder using NAND gates only.
- 44.** A three-bit message is to be transmitted with an even-parity bit.
  - (i) Design the logic circuit for the parity generator.
  - (ii) Design the logic circuit for the parity checker.
- 45.** Implement the Boolean function:  $F = AB'CD' + A'BCD' + AB'C'D + A'BC'D'$  with exclusive-OR and AND gates.
- 46.** Construct a 3-to-8 line decoder.
- 47.** Construct a  $3 \times 8$  multiplexer.
- 48.** Construct a  $8 \times 3$  ROM using a decoder and OR gates.
- 49.** Construct a  $16 \times 4$  ROM using a decoder and OR gates.
- 50.** Determine which of the following diodes below are conducting and which are non conducting.



51. Determine which transistors are ON are which are OFF.



52. Construct voltage and logic truth tables for the circuits shown below. Determine



the logic operation performed by each. Assume ideal diodes i.e., -neglect the voltage drop across each diode when it is forward biased.

53. Draw the logic circuits for the following functions:

- (a)  $B.(A.C') + D + E'$
- (b)  $(A + B)'.C + D$
- (c)  $(A + B).(C' + D)$

54. Prove the following identities by writing the truth tables for both sides.

- (a)  $A.(B + C) == (A.B) + (A.C)$
- (b)  $(A.B.C)' == A' + B' + C'$



- (c)  $A.(A + B) == A$
- (d)  $A + A'.B == A + B$

55. A warningbuzzer is to sound when the following conditions apply:

- Switches A, B, C are on.
- Switches A and B are on but switch c is off.
- Switches A and C are on but switch B is off.
- Switches C and B are on but switch A is off.

Draw a truth table for this situation and obtain a Boolean expression for it. Minimize this expression and draw a logic diagram for it using only (a) NAND (b) NOR gates. If the delay of a NAND gate is 15ns and that of a NOR gate is 12ns, which implementation is faster.

56. Which of the following expressions is in sum of products form? Which is in product of sums form ?

- (a)  $A.(B.D)'$
- (b)  $C.D'.E + F' + D$
- (c)  $(A + B).C$

57. Without formally deriving an logic expressions, deduce the value of each function W, X, Y and Z.

A	B	C	W	X	Y	Z
0	0	0	0	1	0	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	0	0
1	0	0	0	1	1	1
1	0	1	0	1	1	0
1	1	0	0	1	1	1
1	1	1	0	1	1	0

58. Define the following terms:

- (a) Canonical
- (b) Minterm
- (c) Maxterm
- (d) Sum-of-products form
- (e) Product-of-sum form
- (f) Canonical sum-of-products
- (g) Canonical product-of-sums

59. An audio (beeper) is to be activated when the key has been removed from the ignition of a car and the headlights are left *on*. The signal is also to be activated if the key is in the ignition lock and the driver's door is opened. A **1** level is produced when the headlight switch is *on*. A **1** level is also produced from the

ignition lock when the key is in the lock, and a **1** level is available from the driver's door when it is open. Write the Boolean equation and truth table for this problem.

- 60.** A car has two seat switches to detect the presence of a passenger and two seat belt switches to detect fastened seat belts. Each switch produces a *1* output when activated. A signal light is to flash when the ignition when the ignition is switched on any passenger present without his or her seat belt fastened. Design a suitable logic circuit.
- 61.** Draw logic diagrams for the simplified expressions found in Question 37 using.
- NAND gates only.
  - NOR gates only.
- Assume both  $A$  and  $A'$ ,  $B$  and  $B'$  .. etc. are available as inputs.
- 62.** You are installing an alarm bell to help protect a room at a museum form unauthorized entry. Sensor devices provide the following logic signals:
- ARMED = the control system is active  
 DOOR = the room door is closed  
 OPEN = the museum is open to the public  
 MOTION = There is motion in the room
- Devise a sensible logic expression for ringing the alarm bell.
- 63.** A large room has three doors, A, B and C, each with a light switch that can turn the room light ON or OFF. Flipping any switch will change the condition of the light. Assuming that the light switch is off when the switch variables have the values 0, 0, 0 write a truth table for a function LIGHT that can be used to direct the behaviour of the light. Derive a logic equation for LIGHT. Can you simplify this equation ?
- 64.** Design a combinational circuit that accepts a three-bit number and generates an output binary number equal to the square of the input number.
- 65.** Design a combinational circuit whose input is a four-bit number and whose output is the 2's compliment of the input number.
- 66.** A combinational circuit has four inputs and one output. The output is equal to 1 when (I) all the inputs are equal to 1 or (II) none of the inputs are equal to 1 or (III) an odd number of inputs are equal to 1.
- (i) Obtain the truth table.
  - (ii) Find the simplified output function in SOP
  - (iii) Find the simplified output function in POS
  - (iv) Draw the two logic diagrams.
- 67.** Find the canonical s-of-p form the following logic expressions:
- (a)  $W = ABC + BC'D$
  - (b)  $F = VXW'Y + W'XYZ$
- 68.** A certain proposition is true when it is not true that the conditions A and B both hold. It is also true when conditions A and B both hold but condition C does not. Is the proposition true when it is true that conditions B and C both hold ? Use Boolean algebra to justify your answer.

69. Write down the canonical s-of-p form and the p-of-s form for the logic expression whose truth table is each of the following.

(I)

$X_1$	$X_2$	$X_3$	$Z$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

(II)

$A$	$B$	$C$	$W$
0	0	0	1
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

(III)

$W$	$X$	$Y$	$Z$	$F$
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	1	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

70. Use Question 5.
- (a) Using K-maps, obtain the simplified logic expressions for both s-of-p and p-of-s for each truth table in the question 5 above.
  - (b) From the p-of-s expressions, work backwards using a K-map to obtain the s-of-p canonical expression for each truth table.
71. Apply De-Morgan's theorems to each expression
- (a)  $\overline{A + \overline{B}}$
  - (b)  $\overline{\overline{AB}}$
  - (c)  $\overline{\overline{A + B + C}}$
  - (d)  $\overline{\overline{A B C}}$
  - (e)  $\overline{A (B + C)}$
  - (f)  $\overline{\overline{AB} + \overline{CD}}$
  - (g)  $\overline{\overline{AB} + \overline{CD}}$
  - (h)  $\overline{(A + \overline{B}) + (\overline{C} + D)}$

(j)  $\overline{\overline{\overline{ABC}} \overline{\overline{\overline{EFG}}}} + \overline{\overline{\overline{HIJ}} \overline{\overline{\overline{KLM}}}}$       (j)  $\overline{A + \overline{BC} + CD + \overline{BC}}$

(k)  $\overline{\overline{\overline{(A + B)} \overline{\overline{\overline{(C + D)} \overline{\overline{\overline{(E + F)} \overline{\overline{\overline{(G + H)}}}}}}}}}}$

72. Convert the following expressions to sum-of-products forms:

(a)  $AB + CD(\overline{AB} + CD)$       (b)  $AB(\overline{BC} + BC)$       (c)  $A + B[AC + (B + \overline{C})D]$

73. Write a Boolean expression for the following

- (a) X is a 1 only if a is a 1 and B is a 1 or if A is a 0 and B is a 0
- (b) X is a 0 if any of the three variables A, B and C are 1's. X is a 1 for all other conditions.

74. Draw logic circuits using AND, OR and NOT elements to represent the following

(a)  $A\overline{B} + \overline{A}B$       (b)  $AB + \overline{A}\overline{B} + \overline{A}BC$       (c)  $A + B[C + D(B + \overline{C})]$

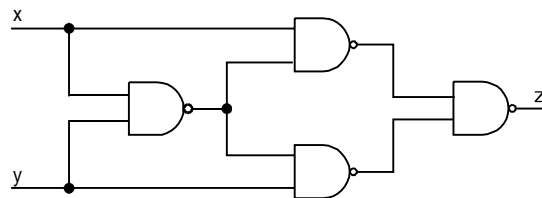
(d)  $A + \overline{BC} + D(\overline{E} + \overline{F})$       (e)  $\overline{\overline{AB}} \overline{\overline{CD}}$       (f)  $[(A + B)(C + D)]E + FG$

75. Use Duality to derive new Boolean identities from the ones you obtained by simplification in the previous question.

76. Use De Morgan's Theorem to complement the following Boolean expressions

(a)  $z = x.(y + w.v)$

(b)  $z = x.y.w + y.(\overline{w} + \overline{v})$



Prove this implements XOR.

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

(d)  $z = x + y.\overline{w}$

(e)  $z = (x + y).w$

(f)  $z = x + \overline{y.w}$

77. Using Boolean Algebra verify that the circuit in figure 1 implements an exclusive OR (XOR) function

- (a) Express  $z_1$  and  $z_2$  as a sum of minterms
- (b) Express  $z_1$  and  $z_2$  as a product of maxterms
- (c) Simplify the sum of the minterm for  $z_1$  and  $z_2$  using Boolean algebra.

# BOOLEAN FUNCTION MINIMIZATION TECHNIQUES

---

## 3.0 INTRODUCTION

The minimization of combinational expression is considered to be one of the major steps in the digital design process. This emphasis on minimization stems from the time when logic gates were very expensive and required a considerable amount of physical space and power. However with the advent of Integrated circuits (SSI, MSI, LSI and VLSI) the traditional minimization process has lessened somewhat, there is still a reasonable degree of correlation between minimizing gate count and reduced package count.

It is very easy to understand that the complexity of logical implementation of a Boolean function is directly related to the complexity of algebraic expression from which it is implemented.

Although the truth table representation of a Boolean function is unique, but when expressed algebraically, it can appear in many different forms.

Because of the reason mentioned above, the objective of this chapter is to develop an understanding of how modern reduction techniques have evolved from the time consuming mathematical approach (Theorem reduction) to quick graphical techniques called 'mapping' and 'tabular method' for large number of variable in the combinational expression.

## 3.1 MINIMIZATION USING POSTULATES AND THEOREM OF BOOLEAN ALGEBRA

The keys to Boolean minimization lie in the theorems introduced in Chapter 2. Section 2.3.2. The ones of major interest are theorem member 6, 7 and 8.

Then, 6 (a) $A + AB = A$	(b) $A(A + B) = A$	Absorption
7 (a) $A + A'B = A + B$	(b) $A(A' + B) = AB$	
8 (a) $AB + AB' = A$	(b) $(A + B)(A + B') = A$	Logic Adjacency

Theorem 6 and 7 have special significance when applied to expression in standard form, whereas theorem 8 is of particular importance in simplifying canonical form expression.

— Theorem 6 has a wood statements—*If a smaller term or expression is formed entirely in a larger term, then the larger term is superfluous.*

Theorem 6 can be applied only when an expression is in a 'standard form', that is, one that has at least one term which is not a MIN or MAX term.

**Example 3.1**

$$\begin{aligned}
 F &= CD + AB'C + ABC' + BCD \\
 &\quad \uparrow \hspace{10em} \uparrow \\
 &\quad \text{Thm 6} \hspace{10em} [\because A + AB = A] \\
 &= CD + AB'C + ABC'
 \end{aligned}$$

⇒ Select one of the smaller terms and examine the larger terms which contain this smaller term.

— for application of Theorem 7, the larger terms are scanned looking for of application the smaller in its complemented form.

**Example 3.2**

$$\begin{aligned}
 F &= AB + BEF + A'CD + B'CD \\
 &= AB + BEF + CD (A' + B') \\
 &= AB + BEF + CD (AB)' \\
 &\quad \uparrow \hspace{10em} \uparrow \\
 &\quad \text{Thm 7} \hspace{10em} \rightarrow \text{Using Demorgans's Theorem} \\
 &= AB + BEF + CD \hspace{10em} (\because A + A'B = AB)
 \end{aligned}$$

— Theorem 8 is the basis of our next minimization technique *i.e.*, Karnaugh map method. It has a word statement—‘If any two terms in a canonical or standard form expression vary in only one variable, and that variable in one term is the complement of the variable in the other term then of the variable is superfluous to both terms.

**Example 3.3**

$$\begin{aligned}
 F &= A'B'C' + A'B'C + ABC' + AB'C \\
 &\quad \uparrow \text{Thm 8} \uparrow \hspace{2em} \uparrow \text{Thm 8} \uparrow \\
 &\quad \swarrow \hspace{10em} \searrow \\
 F &= A'B' + AC
 \end{aligned}$$

**Example 3.4**

$$\begin{aligned}
 F &= A'B'C' + AB'C' + ABC' + A'B'C \\
 &\quad \boxed{\hspace{10em} \text{Thm 8} \hspace{10em} \text{Thm 8} \hspace{10em}} \\
 &\quad \swarrow \\
 &= A'B' + AC'
 \end{aligned}$$

By now it should become obvious that another techniques is needed because this techniques backs specific rules to predict each succeeding step in manipulative process.

Therefore, if we could develop some graphical technique whereby the application of ‘Theorem 8’ the logical adjacency theorem is made obvious and where the desired grouping could be plainly displayed, we would be in mind better position to visualize the proper application of theorem.

**3.2 MINIMIZATION USING KARNAUGH MAP (K-MAP) METHOD**

In 1953 Maurice Karnaugh developed K-map in his paper titled ‘The map method for synthesis of combinational logic circuits.

The map method provides simple straight forward procedure for minimizing Boolean functions that may be regarded as pictorial form of truth table. K-map orders and displays the

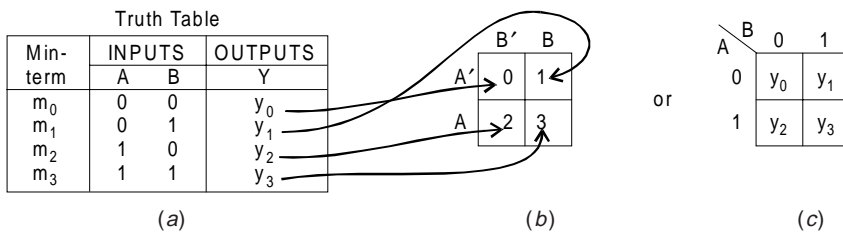
minterms in a geometrical pattern such that the application of the logic adjacency theorem becomes obvious.

- The K-map is a diagram made up of squares. Each square represents one minterm.
- Since any function can be expressed as a sum of minterms, it follows that a Boolean function can be recognized from a map by the area enclosed by those squares. Whose minterms are included in the operation.
- By various patterns, we can derive alternative algebraic expression for the same operation, from which we can select the simplest one. (One that has minimum member of literals).

Now let us start with a two variable K map.

### 3.2.1 Two and Three Variable K Map

If we examine a two variable truth table, Fig. 3.2.1(a) we can make some general observations that support the geometric layout of K Map shown in Fig. 3.2.1(b).



**Fig. 3.2.1** (a) (b) and (c)

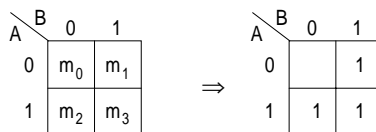
The four squares (0, 1, 2, 3) represent the four possibly combinations of A and B in a two variable truth table. Square 1 in the K-map; then, stands for  $A'B'$ , square 2 for  $A'B$ , and so forth. The map is redrawn in Fig. 3.2.1(c) to show the relation between the squares and the two variables. The 0's and 1's marked for each row and each column designate the values of variables A and B respectively. A appears primed in row 0 and unprimed in row 1. Similarly B appears primed in column 0 and unprimed in column 1.

Now let us map a Boolean function  $Y = A + B$ .

Method I—(i) Draw the truth table of given function.

Min-term	A	B	Y
$m_0$	0	0	0
$m_1$	0	1	1
$m_2$	1	0	1
$m_3$	1	1	1

(ii) Draw a two variable K map an fill those squares with a 1 for which the value of minterm in the function is equal to 1.



The empty square in the map represent the value of minterms [ $m_0$  (or  $A'B'$ )] that is equal to zero in the function. Thus, actually this empty square represents zero.

Method II-(i) Find all the minterms of the function  $Y = A + B$ .

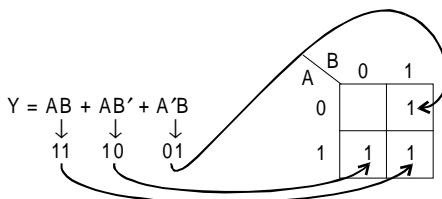
$$Y = A + B = A(B + B') + B(A + A')$$

$$= AB + AB' + AB + A'B$$

⇒

$$Y = AB + AB' + A'B$$

(ii) Draw a two variable K map using this sum of minterms expression.



— These a K map, is nothing more than an interesting looking Truth-Table, and it simply provide a graphical display of ‘implicants’ (minterms) involved in any SOP canonical or standard form expression.

Now examine a three variable truth table shown in 3.2.1 (d).

Truth Table

Min term	INPUTS			OUTPUT
	A	B	C	
$m_0$	0	0	0	$y_0$
$m_1$	0	0	1	$y_1$
$m_2$	0	1	0	$y_2$
$m_3$	0	1	1	$y_3$
$m_4$	1	0	0	$y_4$
$m_5$	1	0	1	$y_5$
$m_6$	1	1	0	$y_6$
$m_7$	1	1	1	$y_7$

Fig. 3.2.1 (d)

Here we need a K-map with 8 squares represent all the combination (Minterms) of input variables A, B and C distinctly. A three-variable map is shown in Fig. 3.2.1 (e).

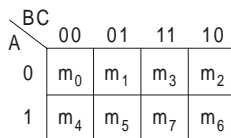


Fig. 3.2.1 (e)

It is very important to realize that in the three variable K-map of Fig. 3.2.1 (e), the minterms are not arranged in a binary sequence, but similar to ‘Gray Code’ sequence.

The gray code sequence is a unit distance sequence that means only one bit changes in listing sequence.



Our basic objective in using K-map is the simplify the Boolean function to minimum number of literals. The gray code sequencing greatly helps in applying. 'Logic Adjacency theorem' to adjacent squares that reduces number of literals.

The map in Fig. 3.2.1 (e) is redrawn in Fig. 3.2.1 (f) that will be helpful to make the pictures clear.

		BC			
		00	01	11	10
A	0	A'B'C'	A'B'C	A'BC	A'BC'
	1	AB'C'	AB'C	ABC	ABC'

Fig. 3.2.1 (f)

We can see that any two adjacent squares in the map differ by only one variable, which is primed in one square and unprimed in other.

For example  $m_3$  (A'BC) and  $m_7$  (ABC) are two adjacent squares. Variable A is primed in  $m_3$  and unprimed in  $m_7$ , whereas the other two variables are same in both squares. Now applying 'logic adjacency' theorem, it is simplified to a single AND term of only two literals. To clarify this, consider the sum of  $m_3$  and  $m_7 \rightarrow m_3 + m_7 = A'BC + ABC = BC(A + A) = BC$ .

### 3.2.2 Boolean Expression Minimization Using K-Map

1. Construct the K-map as discussed. Enter 1 in those squares corresponding to the minterms for which function value is 1. Leave empty the remaining squares. *Now in following steps the square means the square with a value 1.*
2. Examine the map for squares that can not be combined with any other squares and form group of such signal squares.
3. Now, look for squares which are adjacent to only one other square and form groups containing only two squares and which are not part of any group of 4 or 8 squares. A group of two squares is called a *pair*.
4. Next, Group the squares which result in groups of 4 squares but are not part of an 8-squares group. A group of 4 squares is called a *quad*.
5. Group the squares which result in groups of 8 squares. A group of 8 squares is called *octet*.
6. Form more pairs, quads and outlets to include those squares that have not yet been grouped, and use only a minimum no. of groups. There can be overlapping of groups if they include common squares.
7. Omit any redundant group.
8. Form the logical sum of all the terms generated by each group.

Using Logic Adjacency Theorem we can conclude that,

- a group of two squares eliminates one variable,
- a group of four sqs. eliminates two variable and a group of eight squares eliminates three variables.

Now let us do some examples to learn the procedure.

**Example 3.2.** Simplify the boolean for  $F = AB + AB' + A'B$ . Using two variable K-map. This function can also be written as

$$F(A, B) = \Sigma(1, 2, 3)$$

**Solution. Step 1.** Make a two variable K-map and enter 1 in squares corresponding to minterms present in the expression and leave empty the remaining squares.

		B	
		0	1
A	0	$m_0$	$m_1$
	1	$m_2$	$m_3$

**Step 2.** There are no 1's which are not adjacent to other 1's. So this step is discarded.

		B	
		0	1
A	0		1
	1	1	1

**Step 3.**  $m_1$  is adjacent to  $m_3 \Rightarrow$  forms a group of two squares and is not part of any group of 4 squares. [A group of 8 squares is not possible in this case].

		B	
		0	1
A	0		1
	1	1	1

Similarly  $m_2$  is also adjacent to  $m_3 \Rightarrow$  forms another group of two squares and is not a part of any group of 4 squares.

**Step 4 and 5.** Discarded because these in no quad or outlet.

**Step 6.** All the 1's have already been grouped.

There is an overlapping of groups because they include common minterm  $m_3$ .

**Step 7.** There is no redundant group.

**Step 8.** The terms generated by the two groups are 'OR' operated together to obtain the expression for F as follows:

F = A	+	B
↓		↓
From group $m_2 m_3$		From group $m_1 m_3$
↓		↓
This row is corresponding to the value of A is equal to 1.		This column is corresponding to the value of B is equal to 1.

**Example 3.3.** Simplify the Boolean function  $F(A, B, C) = \Sigma(3, 4, 6, 7)$ .

**Solution. Step 1.** Construct K-map.

There are cases where two squares in the map are considered to be adjacent even through they do not touch each other. In a three var K-map,  $m_0$  is adjacent to  $m_2$  and  $m_4$  is adjacent to  $m_6$ .

Algebraically  
and

$$m_0 + m_2 = A'B'C' + A'BC' = A'C'$$

$$m_4 + m_6 = AB'C' + ABC' = AC'$$

	BC			
A	00	01	11	10
0	$m_0$	$m_2$	$m_3$	$m_2$
1	$m_4$	$m_5$	$m_7$	$m_6$

Consequently, we must modify the definition of adjacent squares to include this and other similar cases. This we can understand by considering that the map is drawn on a surface where the right and left edges touch each other to form adjacent squares.

**Step 2.** There are no 1's which are not adjacent to other 1's so this step is discarded.

	BC			
A	00	01	11	10
0	$m_0$	$m_2$	$m_3$	$m_2$
1	$m_4$	$m_5$	$m_7$	$m_6$

**Step 3.**  $m_3$  is adjacent to  $m_7$ . It forms a group of two squares and is not a part of any group of 4 or 8 squares.

Similarly  $m_6$  is adjacent to  $m_7$ . So this is second group (pair) that is not a part of any group of 4 or 8 squares.

Now according to new definition of adjacency  $m_4$  and  $m_6$  are also adjacent and form a pair. Moreover, this pair (group) is not a part of any group of 4 or 8 sqs.

**Step 4 and 5.** Discarded, because no quad or octet is possible.

**Step 6.** All the 1's have already been grouped. These is an overlapping of groups because they include a common minterm  $m_7$ .

**Step 7.** The pair formed b  $m_6$   $m_7$  is redundant because  $m_6$  is already covered in pair  $m_4$   $m_6$  and  $m_7$  in pair  $m_3$   $m_7$ . Therefore, the pair  $m_6$   $m_7$  is discarded.

**Step 8.** The terms generated by the remaining two groups are 'OR' operated together to obtain the expression for F as follows:

$$F = AC' \qquad + \qquad BC$$

↓

From group  $m_4$   $m_6$   
The row is corresponding to the value of  $A = 1$  and in the two columns (00 →  $B'C'$  and the 10 →  $BC'$ ), the value  $C = 0 \Rightarrow C'$  is common  
=  $AC'$

↓

From group  $m_3$   $m_7$ .  
Correspond to both rows ( $\Rightarrow A = 0$  and  $A = 1$ ) so A is omitted, and single column ( $\Rightarrow B = 1$  and  $C = 1$ ),  
 $\Rightarrow BC$ .

With a little more practice, you can make yourself comfortable in minimization using K-map technique. Then you have no used to write all the steps. You can directly minimize the given function by drawing simply the map.

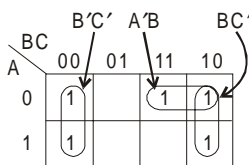
Now let us do one more example of a three variable K-map.

**Example 3.4.** *Simply the following Boolean function by first expressing it in sum of minterms.*

$$F = A'B + BC' + B'C'$$

**Solution.** The given Boolean expression is a function of three variables A, B and C. The three product terms in the expression have two literals and are represented in a three variable map by two squares each.

The two squares corresponding to the first terms  $A'B$  are formed in map from the coincidence of  $A'$  ( $\Rightarrow A = 0$ , first row) and B (two last columns) to gives squares 011 and 010.



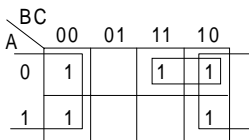
Note that when marking 1's in the squares, it is possible to find a 1 already placed there from a preceding term. This happens with the second term  $BC'$ , has 1's in the sqs. which 010 and 110, the sq. 010 is common with the first term  $A'B$ , so only one square (corresponding to 110) is marked 1.

Similarly, the third term  $B'C'$  corresponds to column 00 that is squares 000 and 100.

The function has a total of five minterms, as indicated by five 1's in the map. These are 0, 2, 3, 4 and 6. So the function can be expressed in sum of minterms term:

$$F(A, B, C) = \Sigma(0, 2, 3, 4, 6)$$

Now, for the simplification purpose, let us redraw the map drawn above:



First we combine the four adjacent squares in the first and last columns to given the single literal term  $C'$ .

The remaining single square representing minterm 3 is combined with an adjacent square that has already been used once. This is not only permissible but rather desirable since the two adjacent squares give the two literal term  $A'B$  and the single sq. represent the three literal minterm  $A'BC$ . The simplified function is therefore,

$$F = A'B + C'$$

### 3.2.3 Minimization in Products of Sums Form

So far, we have seen in all previous examples that the minimized function were expressed in sum of products form. With a minor modification, product of sums (POS) form can be obtained. The process is as follows:

1. Draw map as for SOP; mark the 0 entries. The 1's places in the squares represents minterms of the function. The minterms not included in the function denote the complement of the function. Thus the complement of a function is represented on the map by the squares marked by 0's.
2. Group 0 entries as you group 1 entries for a SOP reading, to determine the simplified SOP expression for  $F'$ .

3. Use De Morgan's theorem on  $F'$  to produce the simplified expression in POS form.

**Example 3.5.** Given the following Boolean function:

$$F = A'C + A'B + AB'C + BC$$

Find the simplified products of sum (POS) expression.

**Solution. Step 1.** We draw a three variable K-map. From the given function we observe that minterms 1, 2, 3, 5, and 7 are having the value 1 and remaining minterms *i.e.*, 0, 4 and 6 are 0. So we mark the 0 entries.

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

**Step 2.** Minterms 0 and 4 forms a pair, giving value =  $B'C'$ . Similarly minterms 4 and 6 forms a second pair giving value =  $AC'$ . Therefore we get  $F' = AC' + B'C'$ .

**Step 3.** Applying De Morgan's theorem automatically converts SOP expression into POS expression, giving the value of  $F$

$$\begin{aligned} \Rightarrow & (F)' = [AC' + B'C']' \\ \Rightarrow & F = [(AC)'. (B'C)'] \\ \Rightarrow & \boxed{F = (A + C) . (B + C)} \end{aligned}$$

### 3.2.4 Four Variable K-Map

Let us examine a four variable truth table shown is Fig. We used a K-map with 16 squares to represent all the minterms of input variables A, B, C and D distinctly. A four-variable K-map is shown in fig.

	BC			
	00	01	11	10
AB				
00	$m_0$	$m_1$	$m_3$	$m_2$
01	$m_4$	$m_5$	$m_7$	$m_6$
11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$
10	$m_8$	$m_9$	$m_{11}$	$m_{10}$

**Truth Table**

Minterm	Inputs				Output Y
	A	B	C	D	
$m_0$	0	0	0	0	$y_0$
$m_1$	0	0	0	1	$y_1$
$m_2$	0	0	1	0	$y_2$
$m_3$	0	0	1	1	$y_3$
$m_4$	0	1	0	0	$y_4$
$m_5$	0	1	0	1	$y_5$
$m_6$	0	1	1	0	$y_6$

Minterm	Inputs				Output Y
	A	B	C	D	
$m_7$	0	1	1	1	$y_7$
$m_8$	1	0	0	0	$y_8$
$m_9$	1	0	0	1	$y_9$
$m_{10}$	1	0	1	0	$y_{10}$
$m_{11}$	1	0	1	1	$y_{11}$
$m_{12}$	1	1	0	0	$y_{12}$
$m_{13}$	1	1	0	1	$y_{13}$
$m_{14}$	1	1	1	0	$y_{14}$
$m_{15}$	1	1	1	1	$y_{15}$

The rows and column are numbered in a reflected-code sequence, with only one digit changing value between two adjacent rows or columns.

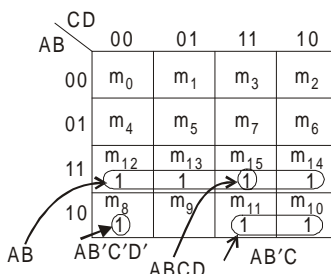
The minimization process is similar as well have done in a three variable K-Map. However the definition of adjacency can further be extended. Considering the map to be on a surface with the top and bottom edges, as well as right and left edges, touching each other of form adjacent squares.

For example  $m_0$  is adjacent to  $m_2, m_4$  as well as to  $m_8$ , similarly  $m_3$  is adjacent to  $m_1, m_7, m_11$  as will as to  $m_{15}$  and so on.

**Example 3.6.** Simplify the given fraction.

$$F = ABCD + AB'C'D' + AB'C + AB$$

**Solution. Step 1.** The given function is consisting of four variables A, B, C and D. We draw a four variable K-map. The first two terms in the function have fours literals and are repeated in a four variable map by one square each. The square corresponding to first term ABCD is equivalent to minterm 1111. ( $m_{15}$ ). Similarly the square for second term  $AB'C'D'$  is equivalent to minterm 1000 ( $m_8$ ) the third term in the function has three literals and is represented in a four var map by two adjacent squares.  $AB'$  corresponds to 4th row (i.e. 10) in the map and C corresponds to last two columns (i.e. 11 and 10) in the map. The last term AB has two (A term with only one literal is represented by 8 adjacent square in map. Finally a 1 in all 16 squares give  $F = 1$ . It means all the minterms are having the value equal to 1 in the function). Literals and is represented by 4 adjacent squares. Here AB simply corresponds to 3<sup>rd</sup> row (i.e.,  $AB = 11$ ).



Now let us redraw the map first for simplification purpose.

**Step 2.** Is discarded. (No 1's which are not adjacent to other 1's)

**Step 3.** Discard (No pairs which are not part of any larger group)

**Step 4.** There are three quads.

Minterms 8, 10, 12, 14 from first quad.

Minterms 12, 13, 14, 15 form second quad

and Minterms 10, 11, 14, 15 form third quad.

**Step 5.** Discarded. (No octages)

**Step 6.** Discarded (All 1's have been grouped.)

**Step 7.** Discard (No redundant term)

**Step 8.** The terms generated by three groups one 'OR' operated as follow

$$\begin{array}{ccccccc}
 F & = & AD' & + & AB & + & AC. \\
 & & \downarrow & & \downarrow & & \downarrow \\
 & & \text{From} & & \text{Second} & & \text{Third} \\
 & & \text{First} & & \text{group} & & \text{group.} \\
 & & \text{group} & & & & 
 \end{array}$$

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

**Example 3.7.** Obtain (a) minimal sum of product (b) minimal product of sum expression for the function given below:

$$F(w, x, y, z) = \sum (0, 2, 3, 6, 7, 8, 10, 11, 12, 15).$$

**Solution.** The given function can also be written in product of minterm form as

$$F(w, x, y, z) = \prod (1, 4, 5, 9, 13, 14).$$

Squares with 1's are grouped to obtain minimal sum of product; square with 0's are grouped to obtain minimal product of sum, as shown.

		YZ			
WX \		00	01	11	10
00		1		1	1
01				1	1
11		1		1	
10		1		1	1

(a) We draw a four variable map using minterms whose values in the function are equal to 1.

- Minterm 8 and 12. Form a pair.
- Minterms 0, 2, 8 and 10 form I quad.
- Minterms 3, 7, 11, 15 form II quad.
- Minterms 2, 3, 6, 7 form III quad.

Therefore,

$$\begin{array}{ccccccc}
 F & = & x'z & + & yz & + & w'y & + & wy'z' \\
 & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
 & & \text{Due to} & & \text{II quad} & & \text{III quad} & & \text{Due to pair} \\
 & & \text{I quad.} & & & & & & 
 \end{array}$$

(b) We draw a four variable map using minterms whose values in the function are equal to zero. These minterms are 1, 4, 5, 9, 13 and 14.

	YZ			
WX \	00	01	11	10
00		0		
01	0	0		
11		0		0
10		0		

- Minterm 14 can not be combined with any other square in the map 4 fours a group with single squares.
- Minterms 4 and 5 form a pair.
- Minterms 1, 5, 9 and 13 form a quad.

Therefore,

$$F' = wxyz' + w'xy' + y'z'$$

Applying De Morgan's theorem

$$(F')' = [wxyz' + w'xy' + y'z']'$$

⇒

$$F = (wxyz)'. w'xy'. (y'z)'$$

⇒

$$F = (w' + x'y' + z). (w + x' + y). (y + z).$$

### 3.2.5 Prime and Essential Implicants

So far we have seen a method for drawing and minimising Karnaugh maps in such a way that unnecessary (redundant) groupings can be avoided. Now let us establish some important definitions that will be used to a systematic procedure for combining squares in the process of K-map minimization. To do this, consider a function defined as  $F(A, B, C, D) = \Sigma(0, 1, 2, 3, 5, 7, 8, 9, 10, 13, 15)$ . Now we will analyze the grouping shown in the 4 variable map in Fig.

	CD			
AB \	00	01	11	10
00	m <sub>0</sub> 1	m <sub>1</sub> 1	m <sub>3</sub> 1	m <sub>2</sub> 1
01	m <sub>4</sub>	m <sub>5</sub> 1	m <sub>7</sub> 1	m <sub>6</sub>
11	m <sub>12</sub>	m <sub>13</sub> 1	m <sub>15</sub> 1	m <sub>14</sub>
10	m <sub>8</sub> 1	m <sub>9</sub> 1	m <sub>11</sub>	m <sub>10</sub> 1

Here we see that all realistic groupings are shown. Note further that each group is sufficiently large that is can not be completely covered by any other simple grouping. Each of these five groupings is defined as a Prime implicant.

- I group → covering minterms → 0, 2, 8, and 10, → B'D'
- II group → covering minterms → 0, 1, 2, and 3 → A'B'
- III group → covering minterms → 1, 5, 9, and 13 → C'D'
- IV group → covering minterms → 1, 3, 5 and 7 → A'D'
- V group → covering minterms → 5, 7, 9 and 15 → B'D'
- VI group → covering minterms → 0, 1, 8, 9 → B'C'



Thus ‘a prime implicant is a product term (or minterm) obtained by combining the maximum possible number of adjacent squares in the map.

As we examine the set of prime implicates that cover this map, it becomes obvious that some of the entries can be grouped in only one way. (Single way groupings). For example there is only one way to group  $m_{10}$  with 4 adjacent squares. ( $\Rightarrow$  I group only). Similarly there is only one way to group  $m_{15}$  with the adjacent square ( $\Rightarrow$  V group only). The resultant terms from these groupings are defined as essential prime implicants.

Thus, ‘if a minterm in a square is covered by only one prime implicant, the prime implicant is said to be essential’. The two essential prime implicant  $\Rightarrow B'D'$  and  $BD$  cover 8 minterms. The remaining three viz  $m_1, m_3$  and  $m_9$  must be considered next.

The prime implicant table shows that  $m_3$  can be covered either with  $A'B'$  or with  $A'D$ .  $m_9$  can be covered either with  $C'D$  or with  $B'C'$ .

$m_1$  can be covered with any of form prime implicates  $AB', AD, B'C'$  or  $C'D$ .

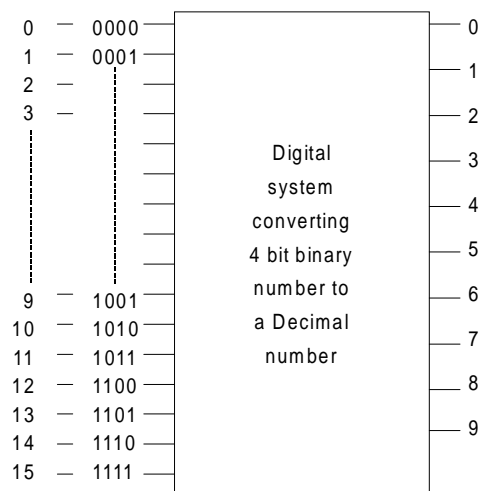
Now the simplified expression is obtained from the sum of two essentials prime implicates and two prime implicant that cover minterms.  $m_1, m_3,$  and  $m_9$ . It means there one four possible ways to write the simplified expression.

- (a)  $BD + B'D' + AB' + CD$
- (b)  $BD + B'D' + AB' + B'C'$
- (c)  $BD + B'D' + A'D + CD$
- (d)  $BD + B'D' + AD + B'C'$

The simplified expression is thus obtained from the logical sum of all the essential prime implicants plus the prime implicants that may be needed to cover any remaining minterms not covered by simplified prime implicants.

### 3.2.6 Don't care Map Entries

Many times in digital system design, some input combinations must be considered as cases that “Just don't happen”, and there are cases when the occurrence of particular combinations will have no effect on the system, and if those combinations do occur, “you don't care”. For example, consider the case where the outputs of a 4-bit binary counter; which happens to home a possible range from 0000 to 1111, is to be converted to a decimal display having the range of 0, 1, 2, ....., 9. The converter might be a digital system having binary 4-bit inputs and decimal upto as shown Fig. 3.2.6. In this particular case, the input combinations 1001, 1010, 1011, 1100, 1101, 1110, 1111 are to be considered by combinations that just can not be accepted by the digital system if it is function properly.



**Fig. 3.2.6**

Therefore, when this digital system is being designed, these minterms in the map are treated in a special way. That is a  $\phi$  or a  $\times$  (cross) is entered into each square to signify “don’t care” MIN/MAX terms.

Reading a map, or grouping a map with don’t care entries is a simple process.

‘Group the  $\phi$  don’t care  $\Rightarrow x$ ) with a 1 grouping if and only if this grouping will result in greater simplification; otherwise treat it as if it were a 0 entry.

**Example 3.8.** Simplify the following Boolean function.

$$F(A, B, C, D) = \Sigma(0, 1, 2, 10, 11, 14) \\ d(5, 8, 9)$$

**Solution.** The K-map for the given function is shown with entries X (don’t care) in squares corresponding to combinations 5, 8 and 9.

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

As discussed above, the 1’s and d’s (Xs) are combined in order to enclose the maximum number of adjacent squares with 1. As shown in K-map in Fig (0), by combining 1’s and d’s (Xs), three quads can be obtained. The X in square 5 is left free since it does not contribute in increasing the size of any group. Therefore the

I Quad covers minterms 0, 2, 10 and d8

II Quad covers minterms 0, 1 and d 8, 9.

III Quad covers minterms 0, 1 and d 8, 9.

A pair covers minterms 10 and 14.

So,

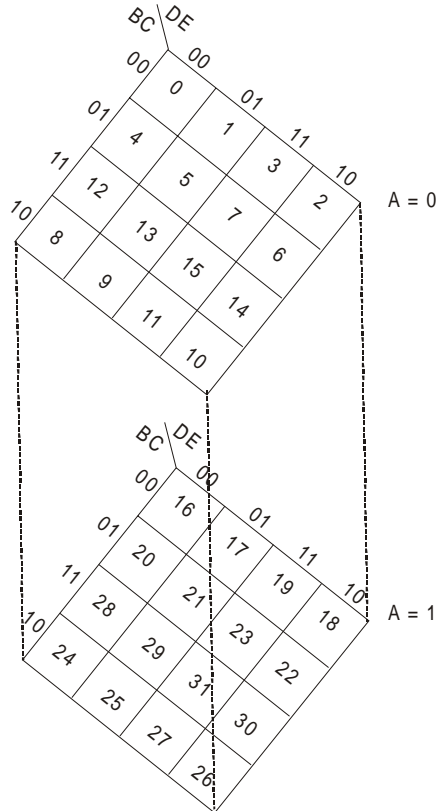
$$F = \begin{matrix} \text{B'D} & + & \text{AB'} & + & \text{B'C'} & + & \text{ACD'} \\ \text{Due} & & \text{II} & & \text{III} & & \text{Due} \\ \text{to I} & & \text{quad} & & \text{Quad} & & \text{to} \\ \text{quad.} & & & & & & \text{Pair.} \end{matrix}$$

### 3.2.7 Five Variable K-Map

The Karnaugh map becomes three dimensional when solving logic problems with more than four variables. A three dimensional K-map will be used in this section.

		DE						DE			
BC		00	01	11	10	BC		00	01	11	10
00		0	1	3	2	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
		A = 0						A = 1			

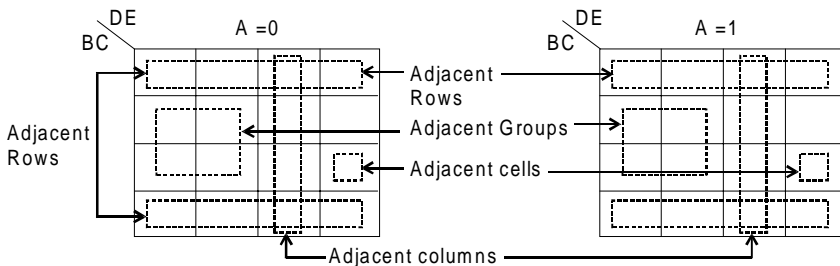
The 5 variable M-map contains  $2^5 = 32$  squares. Instead of representing a single 32-square map, two 16-square K-maps are generally used. If the variable are A, B, C, D and E, the two identical 16-square maps contain B, C, D and E variable with one 16-sq. map for  $A = 1$  and other 16-square map for  $A = 0 \Rightarrow (A)$ . This is shown in Fig. 3.2.7 (a)



**Fig. 3.2.7 (a)**

The minimization procedure described so far with respect to functions of two, three or four variables. Can be extended to the case of five variables.

It is noted that in order to identify the adjacent grouping in the 5-variable maps, we must imagine that the two maps are superimposed on one another as shown. Every square in one map is adjacent to the corresponding square in the other map, because only one variable, changes between such corresponding squares. Thus rows and columns for one map is adjacent to the corresponding row and column on the other map and same rules are used for adjacencies with in one 16 square map. This is illustrate in figure:

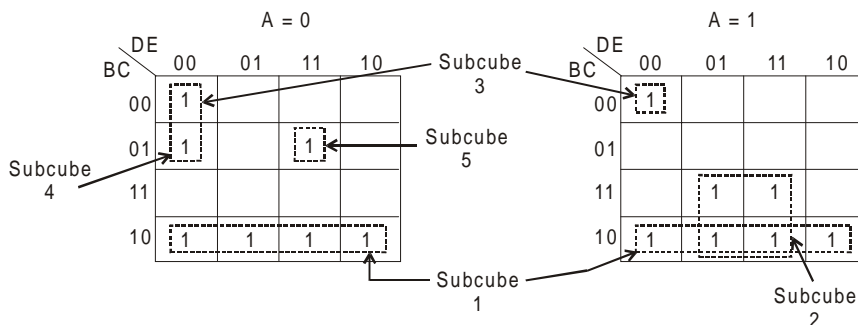


**Fig. 3.2.7 (b)**

**Example 3.9.** Simplify the given function.

$$F(A, B, C, D, E) = E (0, 4, 7, 8, 9, 10, 11, 16, 24, 25, 26, 27, 29, 31)$$

**Solution.** We make two 4-variable maps and fill minterms 0-15 in map corresponding to  $A = 0$  and 16 to 31 corresponding to  $A = 1$



We have 5-subcubes after grouping adjacent squares.

Subcube 1 is an octate which gives  $BC'$

Subcube 2 is a quad which gives  $ABC' \rightarrow$  In the map corresponding to  $A = 1$

Subcube 3 is a pair which gives  $B'CD'E'$

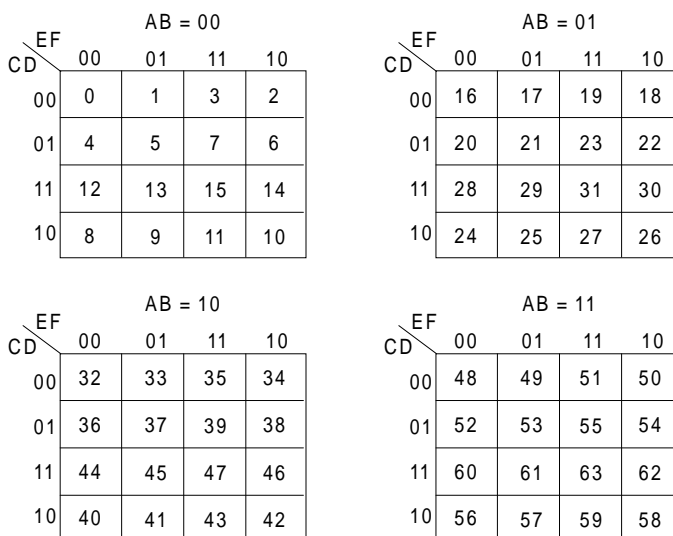
Subcube 4 is a pair which gives  $AB'CE'E' \rightarrow$  In the map corresponding to  $A = 0$

Subcube 5 is a single squares which gives  $A'B'C'D'E \rightarrow$  In the map corresponding to  $A = 0$ .

$$\Rightarrow F(A, B, C, D, E) = BC' + ABC' + B'CD'B' + ABDE' + A'B'CD'E$$

### 3.2.8 Six variable K-Map

A six variable K-map contains  $2^6 = 64$  squares. These square are divided into four identical 16-square maps as shown in Fig. 3.2.8 (a). It the variables are A, B, C, D, E and F, then each 16 square map contains C, D, E, and F as variables along with anyone of 4 combinations of A and B.



**Fig. 3.2.8 (a)**

In order to identify the adjacent groupings in the 6-variable maps, we must imagine that the 4 maps are superimposed on one another. Figure shows different possible adjacent squares:

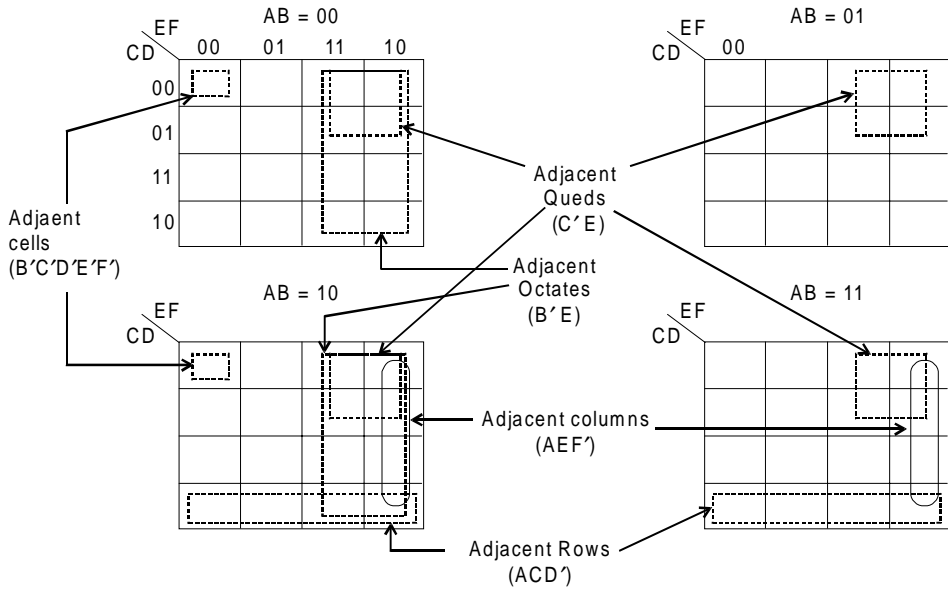
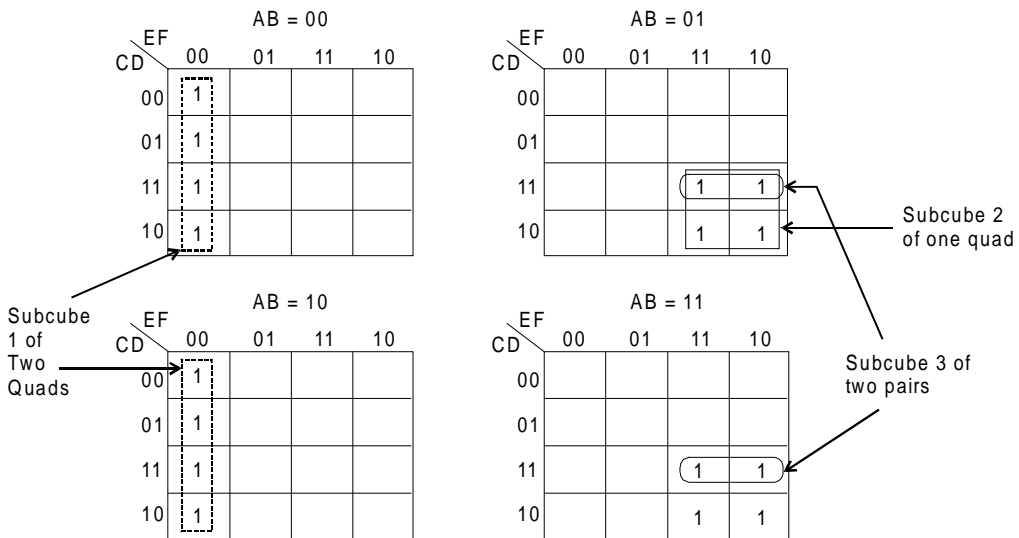


Fig. 3.2.8 (b)

**Example 3.10.** Simplify the given Boolean function.

$$F(A, B, C, D, E, F) = \Sigma(0, 4, 8, 12, 26, 27, 30, 31, 32, 36, 40, 44, 62, 63)$$

**Solution.** We make four 4-varible maps and fill the minterms 0-15 in map corresponding to  $AB = 00$ , 16-31 corresponding to  $AB = 01$ , 32 – 47 corresponding to  $AB = 10$  and 48 to 63 corresponding to  $AB = 11$ .



We have 3-subcubes after grouping adjacent square.

1. Subcubes 1 contains two quads gives  $B'EF'$
2. subcube 2 is form of one quad gives  $A'BCE$
3. subcube 3 is form of two pairs gives  $BCDE$

$\Rightarrow F(A, B, C, D, E, F) = B'EF' + A'BCE + BCDE.$

Maps with seven or more variables needs too many squares and are impractical to use. The alternative is to employ computer programs specifically written to facilitate the simplification of Boolean functions with a large number of variables.

### 3.2.9 Multi Output Minimization

Finding the optimal cover for a system of output expressions all of which are a function of the some variables is somewhat tedious task. This task is basically one of identifying all possible PIs that cover each implicated minterm in each O/P expression, then carrying out a search for the minimal cost cover by using 'shared' terms.

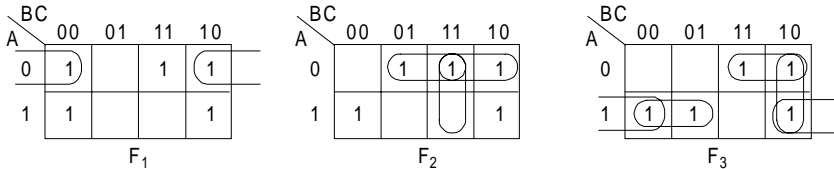
Suppose you were given the following system of expressions and asked to find the optimal cover for the complete system, implying that you must find how to optimally share terms between the expressions.

$$F_1(A, B, C) = \Sigma(0, 2, 3, 5, 6)$$

$$F_2(A, B, C) = \Sigma(1, 2, 3, 4, 7)$$

$$F_3(A, B, C) = \Sigma(2, 3, 4, 5, 6)$$

$\Rightarrow$  we first generate the maps for three expressions as shown



Then we make up an implicant table as shown in Fig. 3.2.9, showing how each minterm can be covered:

Minterm	F <sub>1</sub>	F <sub>2</sub>	F <sub>3</sub>
m <sub>0</sub>	$ABC' / \textcircled{AC'}$	-	-
m <sub>1</sub>	-	$ABC' / \textcircled{AC'}$	-
m <sub>2</sub>	$ABC' / \boxed{AB} / AC' / BC'$	$ABC' / \boxed{AB}$	$ABC' / \boxed{AB} / B'C'$
m <sub>3</sub>	$ABC' / \boxed{AB}$	$ABC' / \boxed{AB} / AC' / BC$	$ABC' / \boxed{AB}$
m <sub>4</sub>	-	$\textcircled{ABC'}$	$\textcircled{ABC'} / AB' / AC'$
m <sub>5</sub>	$\textcircled{ABC}$	-	$\textcircled{ABC} / AB' / AC'$
m <sub>6</sub>	$\textcircled{ABC}$	-	$ABC' / \textcircled{AC'} / AC'$
m <sub>7</sub>	-	$ABC' / \textcircled{BC}$	-

Fig. 3.2.9 Implicant Table

We first scan the table for rows with only a single entry. These are related to essential implicants ( $m_0, m_1, m_7$ ). We take the largest grouping and update the table (In table by making circle).

Next scan the rows for those which have two entries, selecting the functions that have only a single way grouping option ( $m_4$  under  $F_2$  and  $m_5$  and  $m_6$  under  $F_1$ ). It means have to find the common term. We take the common term and update the table (In table by making ovals).

Finally, scan the rows for those rows which have two entries, selecting the functions that have only a single way grouping option or we find the common term. We take the common term and update the table (In table by making rectangular boxes).

Now using implicant table, the three functions can be written as:

$$\begin{aligned} F_1 &= A'C' + A'B + A'B + AB'C + BC' \\ &= A'C' + A'B + BC' + AB'C \\ F_2 &= A'C + A'B + A'B + AB'C' + BC \\ &= A'C + A'B + AB'C' + BC \\ F_3 &= A'B + A'B + AB'C' + BC \\ &= A'B + BC' + AB'C' + AB'C \end{aligned}$$

We see;  $F_3$  is totally generated from shared terms from  $F_1$  and  $F_2$  with considerable saving over a combinational function by function reduction.

In summary, we can say that many times multiple outputs are derived from the same input variables. In this case, we simplify and draw logic diagram of each function separately. Sometimes, the simplified output functions may have common terms. The common term used by one O/P function can be shared by other output functions. This sharing of common terms reduces the total number of gates.

### 3.3 MINIMIZATION USING QUINE-MCCLUSKEY (TABULAR) METHOD

The K-map method is suitable for simplification of Boolean functions up to 5 or 6 variables. As the number of variables increases beyond this, the visualization of adjacent squares is difficult as the geometry is more involved.

The 'Quine-McCluskey' or 'Tabular' method is employed in such cases. This is a systematic step by step procedure for minimizing a Boolean expression in standard form.

#### Procedure for Finding the Minimal Expression

1. Arrange all minterms in groups, such that all terms in the same group have same number of 1's in their binary representation. Start with the least number of 1's and continue with grouping of increasing number of 1's the number of 1's in each term is called the index of that term *i.e.*, all the minterms of some index are placed in a some group. The lowest of value index is zero. Separate each group by a thick line. This constitutes the I stage.
2. Compare every term of the Lowest index (say  $i$ ) group with each term in the successive group of index (say,  $i + 1$ ). If two minterms differ only one variable, that variable should be removed and a dash (-) is placed at the position, thus a new term with only less literal is formed. If such a situation occurs, a check mark (✓) is

placed next to both minterms. After all pairs of terms with indices  $i$  and  $(i + 1)$  have been considered, a thick line is drawn under the last terms.

When the above process has been repeated for all the groups of I stage, one stage of elimination have been completed. This constitutes the II stage.

3. The III stage of elimination should be repeated of the nearly formed groups of second stage. In this stage, two terms can be compared only than they have dashes in some positions.

The process continues to next higher stages until no further comparisons are possible. (i.e., no further elimination of literals).

4. All terms which remain unchecked (No ✓ sign) during the process are considered to be prime implicants (PIs). Thus, a set of all PIs of the function is obtained.
5. From the set of all prime implicants, a set of essential prime implicants (EPIs) must be determined by preparing prime implicant chart as follow.
  - (a) The PIs should be represented m rows and each minterm of the function in a column.
  - (b) Crosses should be placed in each row to show white composition of minterms that makes the PIs.
  - (c) A complete PIs chart should be inspected for columns containing only a single cross. PIs that cover minterms with a single cross in their column are called EPIs.
6. The minterms which are not covered by the EPIs are taken into consideration and a minimum cover is obtained form the remaining PIs.

Now to clarify the above procedure, lets do an example step by step.

**Example 3.11.** Simplify the given function using tabular method.

$$F = A, B, C, D = \Sigma (0, 2, 3, 6, 7, 10, 12, 13)$$

**Solution.** 1. The minterms of the function are represented in binary form. The binary represented are grouped into a number of sections interms of the number of 1's index as shown in Table 3.3.1 (a).

**Table 3.3.1 (a)**

Minterms	Binary ABCD	No. of 1's	Minterms Group	Index	Binary ABCE
$m_0$	0000	0	$m_0$	0	0000 ✓
$m_2$	0010	1	$m_2$	1	0010 ✓
$m_3$	0011	2	$m_8$		1000 ✓
$m_6$	0110	2	$m_3$	2	0011 ✓
$m_7$	0111	3	$m_6$		0110 ✓
$m_8$	1000	1	$m_{10}$		1010 ✓
$m_{10}$	1010	2	$m_{12}$	3	1100 ✓
$m_{12}$	1100	2	$m_7$		0111 ✓
$m_{13}$	1101	3	$m_{13}$		1101 ✓

2. Compare each binary term with every term in the adjacent next higher category. If they differ only by one position put a check mark and copy the term into the next column with (-) in the place where the variable is unmatched, which is shown in next Table. 3.3.1 (b<sub>1</sub>)



**Table 3.31 (b<sub>1</sub>)**

<i>Minterm Group</i>	<i>Binary</i>				
	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	
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	PI
3, 7	0	-	1	1	✓
6, 7	0	1	1	-	✓
12, 13	1	1	0	-	PI

**Table 3.3.1 (b<sub>2</sub>)**

<i>Minterm Group</i>	<i>Binary A</i>	<i>B</i>	<i>C</i>	<i>D</i>
0, 2, 8, 10	- 0 - 0	PI		
0, 8, 2, 10	- 0 - 0	PI eliminated		
2, 3, 6, 7	0 - 0 -	PI		
2, 6, 3, 7	0 - 1 -	PI eliminated.		

- Apply some process to the resultant column of Table 3.3.1(b<sub>1</sub>) and continue until no further elimination of literals. This is shown in Table (3.3.1(b)) above.
- All terms which remain unchecked are the PIs. However note that the minterms combination (0, 2) and (8, 10) form the same combination (0, 2, 8, 10) as the comp.. (0, 8 and (2, 10). The order in which these combinations are placed does not prove any effect. Moreover as we know that  $x + x = x_1$  thus we can eliminate one of these combinations.  
The same occur with combination (2, 3) and (6, 7).

- Now we prepare a PI chart to determine EPIs as follows shown in Table 3.3.1 (c).

**Table 3.3.1 (c)**

<i>Prime Implicants</i>	<i>Minterms</i>								
	<i>0</i>	<i>2</i>	<i>3</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>10</i>	<i>12</i>	<i>13</i>
(8, 12)						×		×	
(12, 13) *								×	×
(0, 2, 8, 10) *	×	×				×	×		
(2, 3, 6, 7) *		×	×	×	×				
	✓		✓	✓	✓		✓		✓

- (a) All the PIs are represented in rows and each minterm of the function in a column.
  - (b) Crosses are placed in each row to show the composition of minterms that make PIs.
  - (c) The column that contains just a single cross, the PI corresponding to the row in which the cross appear is essential. Prime implicant. A tick mark is put against each column which has only one cross mark. A star (\*) mark is placed against each. EPI.
6. All the minterms have been covered by EPIs.

Finally, the sum of all the EPIs gives the function in its minimal SOP form

EPIs.	Binary representation				Variable Representation
	A	B	C	D	
12, 13	1	1	0	-	ABC'
0, 2, 8, 10	-	0	-	0	BD'
2, 3, 6, 7	0	-	1	-	AC

Therefore

$$F = ABC' + BD' + AC.$$

If don't care conditions are also given along with the provolone friction, they are also used to find the prime implicating, but it is not compulsory to include them in the final simplified expression.

**Example 3.12.** Simplify the given function using tabular method.

$$F(A, B, C, D) = \Sigma(0, 2, 3, 6, 7) \\ d(5, 8, 10, 11, 15)$$

**Solution.** 1. Step 1 is shown in Table 3.3.2(a). The don't care minterms are also included.

**Table 3.3.2 (a)**

Minterms	Binary ABCD	No. of 1's	Minterms Group	Index	Binary ABCD
$m_0$	0000	0	$m_0$	0	0000 ✓
$m_2$	0010	1	$m_2$		0010 ✓
$m_3$	0011	2	$m_8$	1	1000 ✓
$m_5$	0101	2	$m_3$		0011 ✓
$m_6$	0110	2	$m_5$	2	0101 ✓
$m_7$	0111	3	$m_6$		0110 ✓
$m_8$	1000	1	$m_{10}$		1010 ✓
$m_{10}$	1010	2	$m_7$	3	0111 ✓
$m_{11}$	1011	3	$m_{11}$		1011 ✓
$m_{15}$	1111	4	$m_{15}$	4	1111 ✓

- 2. Step 2 is shown in Table 3.3.2 (b<sub>1</sub>).
- 3. Step 3 is shown in Table 3.3.2 (b<sub>2</sub>).

**Table 3.3.2 (b<sub>1</sub>)**

<i>Minterm Group</i>	<i>Binary</i>			
	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
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 ✓
3, 7	0	-	1	1 ✓
3, 11	-	0	1	1 ✓
5, 7	0	1	-	1 PI
6, 7	0	1	1	- ✓
10, 11	1	0	1	- ✓
7, 15	-	1	1	1 ✓
11, 15	1	-	1	1 ✓

**Table 3.3.2 (b<sub>2</sub>)**

<i>Minterm Group</i>	<i>Binary</i>				
	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	
0, 2, 8, 10	-	0	-	0	PI
0, 8, 2, 10	-	0	-	0	PI Eliminated
2, 3, 6, 7	0	-	1	-	PI
2, 3, 10, 11	-	0	1	-	PI
2, 6, 3, 7	0	-	1	-	PI Eliminated
2, 10, 3, 11	-	0	1	-	PI Eliminated
3, 7, 11, 15	-	-	1	1	PI
3, 11, 7, 15	-	-	1	1	PI Eliminated

4. All the terms which remain unchecked are PIs. Moreover one of two same combinations is eliminated.
5. Step 5 is to prepare a PI chart to determine EPIs as shown in Table 3.3.2 (c).  
 Note, however that don't care minterms will not be listed as column headings in the chart as they do not have to be covered by the minimal (simplified) expression.

**Table 3.3.2 (c)**

Prime Implicants	Minterms				
	0	2	3	6	7
(5, 7)					×
(0, 2, 8, 10) *	×	×			
(2, 3, 6, 7) *		×	×	×	×
(2, 3, 10, 11)		×	×		
(3, 7, 11, 15)			×		×
	✓			✓	

6. All the minterms have been covered by EPIs.

Therefore

$$F(A, B, C, D) = BD' + AC$$

**Example 3.13.** Simplify the given function using tabular method:

$$F(A, B, C, D, E, F, G) = S(20, 28, 38, 39, 52, 60, 102, 103, 127)$$

**Solution.** Step 1 is shown in Table 3.3.3 (a).

Minterms	Binary ABCDEFG	No. of 1's	Minterms Group	Index	Binary ABCDEFG
$m_{20}$	0010100	2	$m_{20}$	2	0010100 ✓
$m_{28}$	0011100	3	$m_{28}$		0011100 ✓
$m_{38}$	0100110	3	$m_{38}$	3	0100110 ✓
$m_{39}$	0100111	4	$m_{52}$		0110100 ✓
$m_{52}$	0110100	3	$m_{39}$	4	0100111 ✓
$m_{60}$	0111100	4	$m_{60}$		0111100 ✓
$m_{102}$	1100110	4	$m_{102}$		1100110 ✓
$m_{103}$	1100111	5	$m_{103}$	5	1100111 ✓
$m_{127}$	1111111	7	$m_{127}$	7	1111111 PI

2. Step 2 is shown in Table 3.3.3 ( $b_1$ ).

3. Step 3 is shown in Table 3.3.3 ( $b_2$ ).

**Table 3.3.3 ( $b_1$ )**

Minterms Group	Binary						
	A	B	C	D	E	F	G
20, 28	0	0	1	-	1	0	0 ✓
20, 52	0	-	1	0	1	0	0 ✓
28, 60	0	-	1	1	1	0	0 ✓
38, 39	0	1	0	0	1	1	- ✓
38, 102	-	1	0	0	1	1	0 ✓
52, 60	0	1	1	-	1	0	0 ✓
39, 103	-	1	0	0	1	1	1 ✓
102, 103	1	1	0	0	1	1	- ✓

**Table 3.3.2 (b<sub>2</sub>)**

Mintesms Group	Binary							
	A	B	C	D	E	F	G	
20, 28, 52, 60	0	-	1	-	1	0	0	PI
20, 52, 28, 60	0	-	1	-	1	0	0	PI Eliminated
38, 39 102, 103	-	1	0	0	1	1	-	PI
38, 102, 39, 103	-	1	0	0	1	1	-	PI Eliminated

- All the terms which remain unchecked are PIs. Moreover one of two same combinations is eliminated.
- PI chart to determine EPIs is shown in Table 3.3.3 (c).

**Table 3.3.3 (c)**

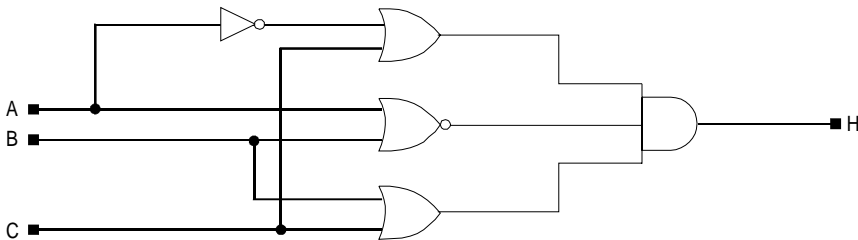
Prime Implicants	Minterms								
	20	28	38	39	52	60	102	103	127
127 *									×
(20, 28, 52, 60) *	×	×			×	×			
(38, 39, 102, 103) *			×	×			×	×	
	✓	✓	✓	✓	✓	✓	✓	✓	✓

- All the minterms have been covered by EPIs.

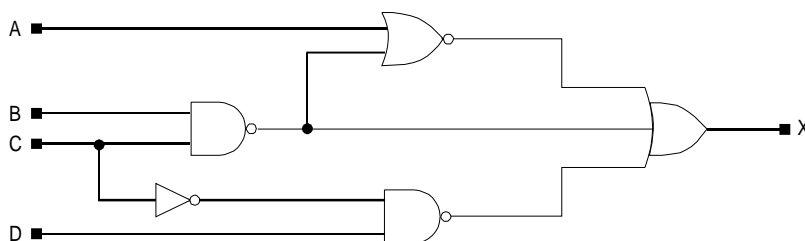
Therefore  $F(A, B, C, D, E, F, G) = ABCDEFG + A'CEF'G' + BC'D'EF$

**3.4 EXERCISES**

- Using Boolean algebra simplify each of the following logic expressions as much as possible:
  - $Z = A(A + AB)(A + ABC)(A + ABCD)$
  - $C = (X_1X_2' + X_2'X_3)'$
- Draw the simplest possible logic diagram that implements the output of the logic diagram given below.



- Write the logic expression and simplify it as much as possible and draw a logic diagram that implements the simplified expression.



4. Obtain the simplified expression in s-of-p for the following Boolean functions:
  - (a)  $xy + x'y'z' + x'yz'$
  - (b)  $ABD + A'C'D' + A'B + ACD + AB'D'$
  - (c)  $x'z + w'xy' + w(x'y + xy')$
  - (d)  $F(x, y, z) = \Sigma(2, 3, 6, 7)$
  - (e)  $F(A, B, C, D) = \Sigma(7, 13, 14, 15)$
5. Use a K-map to simplify each of the following logic expressions as much as possible:
  - (i)  $F = AB' + A'B + AB$
  - (ii)  $G = X'Y'Z' + X'YZ' + XY'Z' + X'Y'Z' + XYZ'$
  - (iii)  $H = A'B'CD + AB'C'D' + A'B'C'D' + ABC'D + A'B'C'D + AB'C'D + ABCD$
  - (iv)  $W = X'Y'Z + X'YZ + XYZ + XY'Z + X'YZ'$
6. Simplify the following logic expressions using K-maps and tabular method.
  - (a)  $F(A, B, C) = A'C + B'C + AB'C'$
  - (b)  $G(A, B, C, D) = B'CD + CD' + A'B'C'D + A'B'C$
7. Simplify the Boolean function  $F_{(ABCDE)} = \Sigma(0, 1, 4, 5, 16, 17, 21, 25, 29)$
8. Simplify the following Boolean expressions using K-maps and Tabular method.
  - (i)  $BDE + B'C'D + CDE + ABCE + ABC + BCDE$
  - (ii)  $ABCE + ABCD + BDE + BCD + CDE + BDE$
  - (iii)  $F_{(ABCDEF)} = \Sigma(6, 9, 13, 18, 19, 27, 29, 41, 45, 57, 61)$
9. Draw Karnaugh maps for the following expressions:
 
$$F = A'.B'.C' + A'.B'.C + A.B'.C + A.B.C$$

$$F = A'.B.C' + A.B.C' + A'.B.C + A.B.C + A.B'.C'$$

$$F = A.B.C'.D' + A.B'.C'.D + A'.B.C.D + A'.B'.C'.D + A'.B'.C'.D$$

$$+ A'.B'.C.D + A'.B'.C.D' + A'.B.C.D'$$
10. Simplify the following logic expressions using karnaugh maps. Draw logic diagrams for them using only (a) NAND, (b) NOR gates, assuming inputs A, B, C, and D only are available.
 
$$Y = A'.B.C'.D' + A.B.C'.D + A.B.C.D + A'.B'.C.D + A.B'.C'.D$$

$$+ A'.B'.C.D' + A'.B.C.D'$$

$$\begin{aligned}
 Y &= A'.B'.C'.D' + A'.B'.C.D + A'.B'.C.D' + A'.B.C.D' + A.B.C.D + A.B'.C.D \\
 Y &= A'.B'.C'.D' + A.B.C'.D' + A.B.C'.D + A'.B'.C'.D + A'.B'.C.D' + A.B.C'.D \\
 &\quad + A'.B'.C.D + A.B'.C.D + A.B'.C'.D' + AB'.C.D' \\
 Y &= A.B.C'.D' + A'.B'.C'.D + A.B'.C'.D' + A.B.C.D + A.B.C.D'
 \end{aligned}$$

11. The institute's pool room has four pool tables lined up in a row. Although each table is far enough from the walls of the room, students have found that the tables are too close together for best play. The experts are willing to wait until they can reserve enough adjacent tables so that one game can proceed unencumbered by nearby tables. A light board visible outside the pool room shows vacant tables. The manager has developed a digital circuit that will show an additional light whenever the experts' desired conditions arise. Give a logic equation for the assertion of the new light signal. Simplify the equation using a K-Map.

12. Simplify the Boolean functions using tabular method and verify result with K-map.

(a)  $F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

(b)  $F(w, x, y, z) = \Sigma(2, 3, 12, 13, 14, 15)$

(c)  $F(A, B, C, D) = \Sigma(4, 6, 7, 15)$

(d)  $F(A, B, C, D) = \Sigma(7, 13, 14, 15)$

(e)  $F(x, y, z) = \Sigma(7, 13, 14, 15)$

13. Simplify the Boolean function F using the don't care conditions d, in (I) SOP and (II) POS:

$$F = A'B'D + A'CD + A'BC \qquad d = A'BC'D + ACD + AB'D'$$

$$F = w'(x'y + x'y' + xyz) + x'z'(y + w) \qquad d = w'x(y'z + yz') + wyz$$

$$F = ACE + A'CD'E' + A'C'DE \qquad d = DE' + A'D'E + AD'E'$$

14. Use a Karnaugh map to simplify each of the following logic expressions as much as possible.

(a)  $F = A'B'CD + AB'C'D' + A'B'CD' + ABC'D + ABCD$

**Solution.**  $F = ABD + A'B'D + B'C'$

(b)  $G = A'C + B'C + AB'C' + A'B$

**Solution.**  $G = AB' + A'B + A'C$  or  $AB' + A'B + B'C$

(c)  $H = B'CD + CD' + A'B'C'D' + A'B'C$

**Solution.**  $H = B'CD + CD' + A'B'C'D' + A'B'C$

(d)  $F = (A' + B + C')(A + B + C)(A + B + C')$

**Solution.**  $F = B + AC'$

15. Use a Karnaugh map to simplify each of the following logic expressions as much as possible.

(a)  $W = (AB'C')'(AB'C)(ABC)'$

(b)  $M = X_2X_3 + X'_1X'_2X_3 + X'_3 + X_1X'_2X_3$

16. Using Boolean Algebra simplify

(a)  $(A + \bar{B})(A + C)$  (b)  $\bar{A}B + \bar{A}BC + \bar{A}BCD + \bar{A}BCDE$

(c)  $AB + \overline{ABC} + A$  (d)  $(A + \bar{A})(AB + \overline{ABC})$

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

17. Use a karnaugh map to simplify each function to a minimum sum-of-products form:

(a)  $X = \overline{ABC} + \overline{ABC} + \overline{ABC}$  (b)  $X = AC [\bar{B} + A (B + \bar{C})]$

(c)  $X = \overline{DEF} + \overline{DEF} + \overline{DEF}$

18.

A	B	C	F <sub>1</sub>
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

A	B	C	F <sub>2</sub>
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Transfer the input-output specifications for F<sub>1</sub> and F<sub>2</sub> given above to 3 variable Karnaugh maps.

19. Using a Karnaugh map simplify the following equations

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

(b)  $X = \bar{A}BC + \bar{A}CD + \bar{A}BC + \bar{B}CD + \bar{A}BC + \bar{A}BD + \bar{A}BCD$

(c)  $X = \bar{D} (\bar{A}[\bar{C} + \bar{B}C] + A [\bar{C} + \bar{B}C]) + BCD$

(d)  $X = \bar{A}BC + \bar{B}CD + \bar{A}BD + ABCD + \bar{A}CD + \bar{A}BCD$

20. Simplify the following using Boolean Algebra

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

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

(c)  $z = x.y + w.\bar{y} + w.x + x.y.v$

(d)  $z = (x + y).(x + \bar{w}).[y.(x + \bar{w}) + \bar{y}]$

21. Consider the function

$z = f(x, y, w, v) = (x.v + \bar{x}.w).[\bar{y}.(w + y.\bar{v})]$

(a) Draw a schematic diagram for a circuit which would implement this function.

22. Simplify the Boolean function by tabular method



$$F(A, B, C, D, E) = \Sigma(0, 1, 4, 5, 16, 17, 21, 25, 29)$$

23. Simplify the following function in (a) s-o-p  
and (b) p-o-s

$$F(A, B, C, D) = \Pi(3, 4, 6, 7, 11, 12, 13, 14, 15)$$

24. Simplify the Boolean function using tabular method.

$$F(A, B, C, D, E) = \Sigma(0, 1, 4, 5, 16, 17, 21, 25, 29, 30)$$

25. Simplify the Boolean function using tabular method

$$F(A, B, C, D, E, F) = \Sigma(6, 9, 13, 18, 19, 27, 29, 41, 45, 57, 61, 63)$$

# 4 CHAPTER

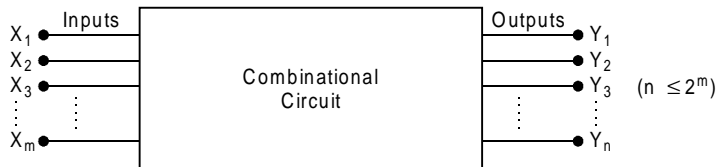
## COMBINATIONAL LOGIC

---

### 4.0 INTRODUCTION

Combinational logic circuits are circuits in which the output at any time depends upon the combination of input signals present at that instant only, and does not depend on any past conditions.

The block diagram of a combinational circuit with  $m$  inputs and  $n$  outputs is shown in Fig. 4.0.



**Fig. 4.0** Block Diagram of combinational Logic circuit.

In particular, the output of particular circuit does not depend upon any past inputs or outputs i.e. the output signals of combinational circuits are not fed back to the input of the circuit. Moreover, in a combinational circuit, for a change in the input, the output appears immediately, except for the propagation delay through circuit gates.

The combinational circuit block can be considered as a network of logic gates that accept signals from inputs and generate signals to outputs. For  $m$  input variables, there are  $2^m$  possible combinations of binary input values. Each input combination to the combinational circuit exhibits a distinct (unique) output. Thus a combinational circuit can be described by  $n$  boolean functions, one for each input combination, in terms of  $m$  input variables with  $n$  is always less than or equal to  $2^m$ . [ $n \leq 2^m$ ].

Thus a combinational circuit performs a specific information processing operation which is specified by Boolean functions.

$n \leq 2^m$  represent the condition, if in a particular application there are some unused input combinations. For example we are using NBCD codes, the six combinations (1010, 1011, 1100, 1101, 1110 and 1111) are never used. So with four input variables ( $\Rightarrow m = 4$ ) we are using only 10 i/p combinations  $\Rightarrow$  10 o/ps instead of  $2^4 = 16$ .

The digital systems perform a member of information processing tasks. The basic arithmetic operations used by digital computers and calculators are implemented by combinational circuits using logic gates. We proceed with the implementation of these basic functions by first looking the simple design procedure.

**Combinational circuit Design Procedure**

It involves following steps :

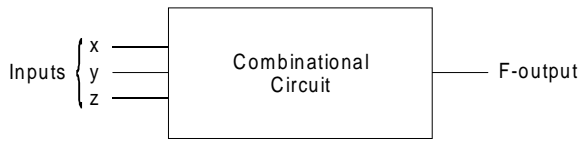
- Step 1 :** From the word description of the problem, identify the inputs and outputs and draw a block diagram.
- Step 2 :** Make a truth table based on problem statement which completely describes the operations of circuit for different combinations of inputs.
- Step 3 :** Simplified output functions are obtained by algebraic manipulation, k-map method or tabular method.
- Step 4 :** Implement the simplified expression using logic gentsis.

To explain the procedure, let us take an example that we have already been used in chapter 2.

**Example:** A TV is connected through three switches. The TV becomes 'on' when atleast two switches are in 'ON' position; In all other conditions, TV is 'OFF'.

**Solution. Step I :** The TV is connected with 3 switches; thus there are three inputs to TV, represented by variables say A, B and C. The o/p of TV is represented by variable say, F.

The block diagram is shown in Fig. 4.1 :



**Fig. 4.1**

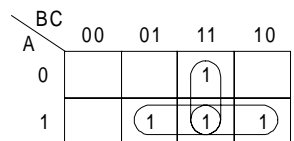
**Step 2. Truth Tables**

TV switches ← INPUTS			OUTPUTS
A	B	C	F
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

- 0 → switch off
- 1 → switch on

It means for the input combinations in which there are two or more 1's, the output F = 1 (TV is ON) and for rest combinations, output F = 0 (TV is OFF).

**Step 3 :** In general, in simplifying boolean functions upto four variables, the best method is K-map technique. Thus, using a 3 variable K-map, we can simplify the function obtained in step II.



We get  $F = AB + AC + BC$

We can observe that if the value of any two variables is equal to 1, the output is equal to 1.

**Step IV.** For implementation we need three 'AND' gates and one 'OR' gate as shown in Fig. 4.2.

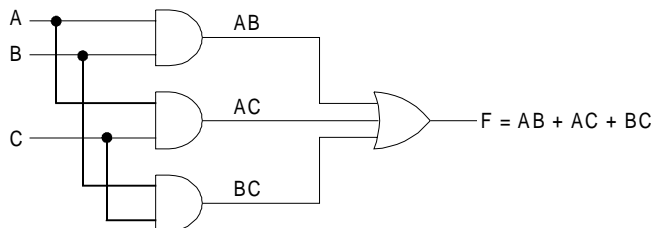


Fig. 4.2

### 4.1 ARITHMATIC CIRCUITS

The logic circuits which are used for performing the digital arithmetic operations such as addition, subtraction, multiplication and division are called 'arithmetic circuits'.

#### 4.1.1 Adders

The most common arithmetic operation in digital systems is the addition of two binary digits. The combinational circuit that performs this operation is called a half-adder.

#### Half Adder

1. Fig. 4.3 shows a half adder (HA).

It has two inputs A and B. that are two 1-bit members, and two output sum (S) and carry (C) produced by addition of two bits.

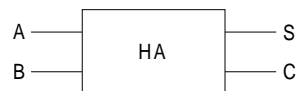


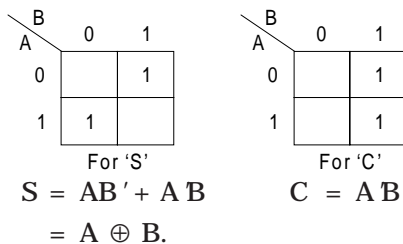
Fig. 4.3 Half Adder

**2. Truth Table :**

Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

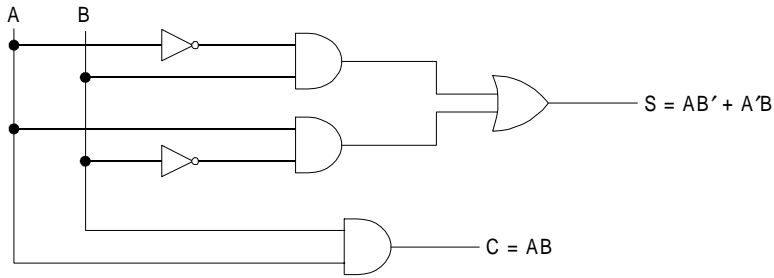
The sum output is 1 when any of inputs (A and B) is 1 and the carry output is 1 when both the inputs are 1.

**3.** Using a two variable *k*-map, separately for both outputs S and C.



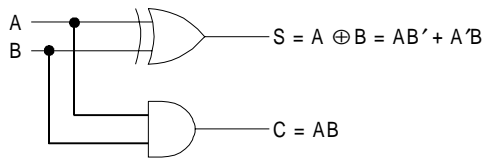
**4. Logical Implementation.**

(i) Using Basic gates (as shown in Fig. 4.4(a)).



**Fig. 4.4 (a)**

(ii) Using XOR gate as shown in Fig. 4.4 (b).



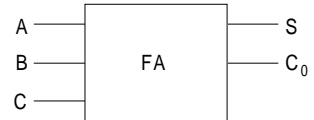
**Fig. 4.4 (b)**

Implementation using only NAND or only NOR gates is left as an exercise.

**Full Adder**

Full adder is a combinational circuit that performs the addition of three binary digits.

1. Fig. 4.5 shows a full adder (FA). It has three inputs A, B and C and two outputs S and  $C_0$  produced by addition of three input bits. Carry output is designated  $C_0$  just to avoid confusion between with i/p variable C.

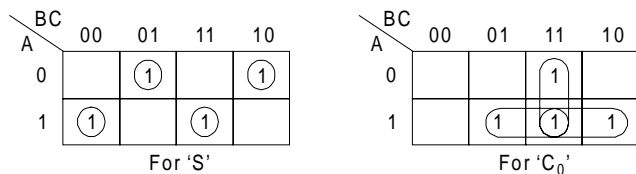


**Fig. 4.5 Full adder**

2. **Truth Table :** The eight possible combinations of three input variables with their respective outputs is shown. We observe that when all the three inputs are 1, the sum and carry both outputs, are 1.

Inputs			Output	
A	B	C	S	$C_0$
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

3. Using a three variable map for both outputs.



$$S = ABC + AB'C' + A'BC' + A'B'C \text{ and } C_0 = AB + AC + BC.$$

4. Logical Implementation. (i) Using basic gates as shown in Fig. 4.6.

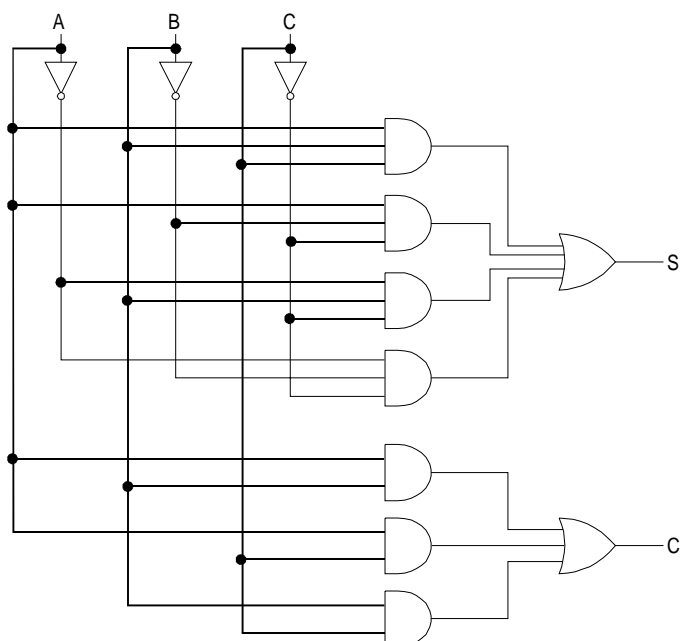


Fig. 4.6

(ii) A 'Full Adder' can also be implemented using two half adders and an 'OR' Gate as shown in Fig. 4.7

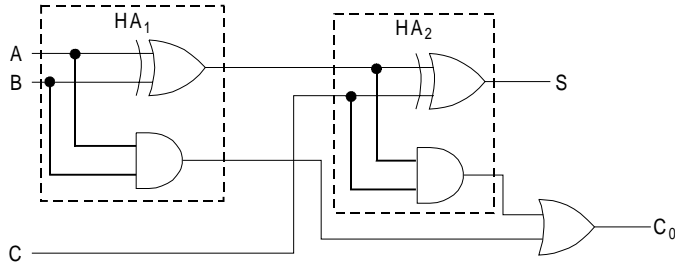
The Sum

$$\begin{aligned} S &= ABC + AB'C' + A'BC' + A'B'C \\ &= ABC + A'B'C + AB'C' + A'BC' \\ &= C(AB + A'B) + C'(AB' + A'B) \\ &= C(AB' + A'B)' + C'(AB' + A'B) \\ &= (A \oplus B) \oplus C \end{aligned}$$

and the carry

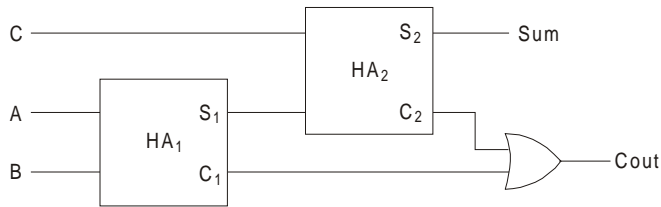
$$\begin{aligned} C_0 &= AB + AC + BC \\ &= AB + C(A + B) \\ &= AB + C(A + B)(A + A')(B + B') \\ &= AB + C[AB + AB' + A'B] \\ &= AB + ABC + C(AB' + A'B) \\ &= AB(1 + C) + C(A \oplus B) \\ &= AB + C(A \oplus B) \end{aligned}$$

$$\Rightarrow S = (A \oplus B) \oplus C \text{ and } C_0 = AB + C(A \oplus B)$$



**Fig. 4.7** Implementation of Full Adder.

Block Diagram representation of a full adder using two half address :



$S_1$  and  $C_1$  are outputs of first half adder ( $HA_1$ )

$S_2$  and  $C_2$  are outputs of second half adder ( $HA_2$ )

A, B and C are inputs of Full adder.

Sum and covt are outputs of full adder.

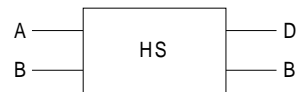
### 4.1.2 Subtractors

The logic circuits used for binary subtraction, are known as 'binary subtractors'.

Half Subtractor : The half subtractor is a combinational circuit which is used to perform the subtraction of two bits.

1. Fig. 4.8 shows a half subtractor. (HS)

It has two inputs, A (minered) and B (subtratend) and two outputs D (difference) and  $B_0$  (Borrow). [The symbol for borrow ( $B_0$ ) is taken to avoid confusion with input variable B] produced by subtractor of two bits.



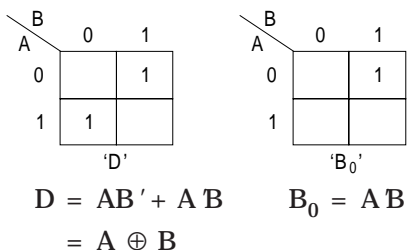
**Fig. 4.8** Half subtractor

2. **Truth Table**

The difference output is 0 if  $A = B$  and 1 is  $A \neq B$ ; the borrow output is 1 whenever  $A < B$ . If  $A < B$ , the subtraction is done by borrowing 1 from the next higher order bit.

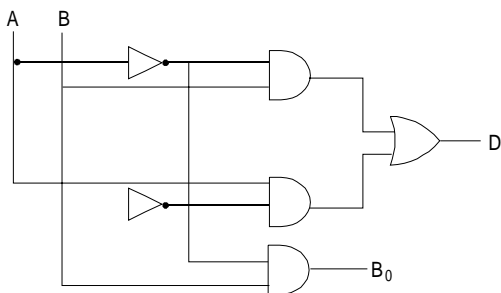
Inputs		Outputs	
A	B	D	$B_0$
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

3. Using a two variable map, for outputs D and B.

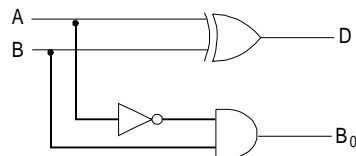


4. Logical Implementation shown in Fig. 4.9

(a) Using Basic gates      (b) using XOR gate



**Fig. 4.9 (a)** Basic gate implementation half subtractor.



**Fig. 4.9 (b)** X-OR gate implementation of half subtractor

### Full subtractor

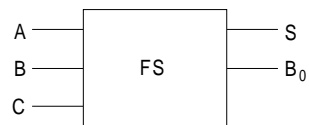
Full subtractor is a combinational circuit that performs the subtraction of three binary digits.

1. Fig. 4.10 shows a full subtractor (FS).

It has three inputs A, B and C and two outputs D and B<sub>0</sub>, produced by subtraction of three input bits.

2. **Truth Table**

The eight possible combinations of three input variables with their respective outputs is shown. We observe that when all the three inputs are 1, the difference and borrow both outputs are 1.

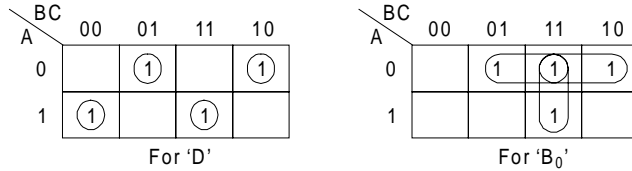


**Fig. 4.10** Full subtractor.

Inputs			Output	
A	B	C	B <sub>0</sub>	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



3. Using a three variable map for both outputs.



$$D = ABC + AB'C' + A'BC' + A'B'C, \quad B_0 = AB + A'C + BC$$

$$D = ABC + AB'C' + A'BC' + A'B'C$$

4. Logical implementation—

(i) Using basic gates : Left as an exercise.

(ii) A 'full subtractor' can also be implemented using two 'half subtractors' and an 'OR' gate as shown in Fig. 4.11.

The difference

$$\begin{aligned}
 'D' &= ABC + AB'C' + A'BC' + A'B'C \\
 &= ABC + A'B'C + AB'C' + A'BC' \\
 &= C (AB + A'B) + C' (AB' + A'B) \\
 &= C (AB' + A'B)' + C' (AB' + A'B) \\
 &= C (A \oplus B)' + C' (A \oplus B) \\
 &= (A \oplus B) \oplus C
 \end{aligned}$$

and the borrow

$$\begin{aligned}
 B_0 &= AB + A'C + BC \\
 &= AB + C (A' + B) \\
 &= AB + C (A' + B) (A + A) (B + B) \\
 &= AB + C [AB + AB + A'B] \\
 &= AB + A'BC + C (AB + A'B) \\
 &= AB (C + 1) + C (A \oplus B)' \\
 &= AB + C (A \oplus B)'
 \end{aligned}$$

$$\Rightarrow D = (A \oplus B) \oplus C \text{ and } B_0 = AB + C (A \oplus B)'$$

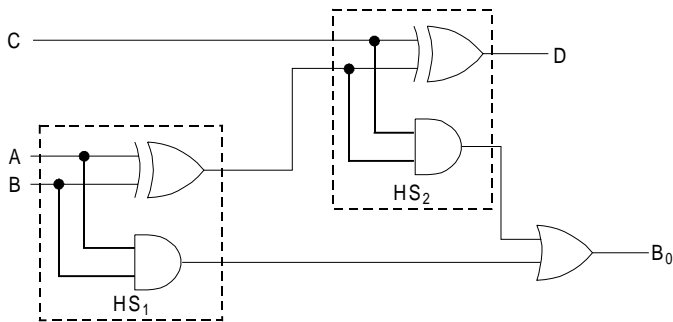


Fig. 4.11

Block Diagram Representation of a full subtractor using two half subtractors :

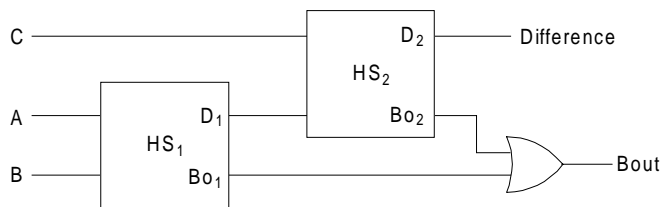


Fig. 4.11 (a)

$D_1$  and  $B_{01}$  are outputs of first half subtractor ( $HS_1$ )  
 $D_2$  and  $B_{02}$  are outputs of second half subtractor ( $HS_2$ )  
 A, B and C are inputs of full subtractor.  
 Difference and Bout are outputs of full subtractor.

### 4.1.3 Code Converters

In the previous study of codes, coding was defined as the use of groups of bits to represent items of information that are multivalued. Assigning each item of information a unique combination of bits makes a transformation of the original information. This we recognize as information being processed into another form. Moreover, we have seen that there are many coding schemes exist. Different digital systems may use different coding schemes. It is sometimes necessary to use the output of one system as the input to other. Therefore a sort of code conversion is necessary between the two systems to make them compatible for the same information.

'A code converter is a combinational logic circuit that changes data presented in one type of binary code to another type of binary code.' A general block diagram of a code converter is shown in Fig. 4.12.

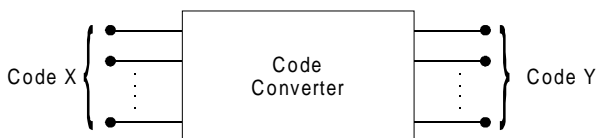


Fig. 4.12 Code converter

To understand the design procedure; we will take a specific example of 4-bit Binary to Gray code conversion.

1. The block diagram of a 4-bit binary to gray code converter is shown in Fig. 4.13.

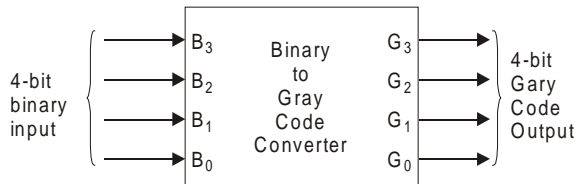


Fig. 4.13

If has four inputs ( $B_3 B_2 B_1 B_0$ ) representing 4-bit binary numbers and four outputs ( $G_3 G_2 G_1 G_0$ ) representing 4-bit gray code.

2. Truth table for binary to gray code converters.

Binary Inputs				Gray code Outputs			
$B_3$	$B_2$	$B_1$	$B_0$	$G_3$	$G_2$	$G_1$	$G_0$
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

3. Now we solve all the gray outputs distantly with respect to binary inputs From the truth table; the logic expressions for the gray code outputs can be written as

$$G_3 = \Sigma (8, 9, 10, 11, 12, 13, 14, 15)$$

$$G_2 = \Sigma (4, 5, 6, 7, 8, 9, 10, 11)$$

$$G_1 = \Sigma (2, 3, 4, 5, 10, 11, 12, 13)$$

$$G_0 = \Sigma (1, 2, 5, 6, 9, 10, 13, 14).$$

The above expressions can be simplified using K-map

**Map for  $G_3$ :**

From the octet, we get

$$G_3 = B_3$$

	$B_1B_0$	00	01	11	10
$B_3B_2$	00				
	01				
	11	1	1	1	1
	10	1	1	1	1

**Map for  $G_2$ :**

From the two quads, we get

$$\begin{aligned} G_2 &= B_3' B_2 + B_3 B_2' \\ &= B_3 \oplus B_2. \end{aligned}$$

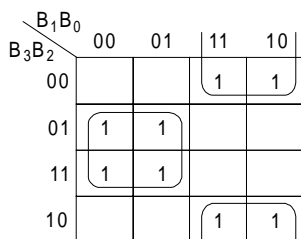
	$B_1B_0$	00	01	11	10
$B_3B_2$	00				
	01	1	1	1	1
	11				
	10	1	1	1	1

**Map for G<sub>1</sub>:**

From the two quads, we get

$$G_1 = B_2 B_1' + B_2' B_1$$

$$= B_2 \oplus B_1$$

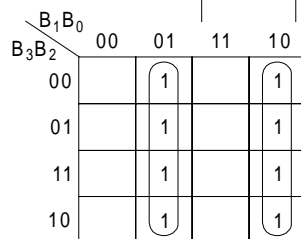


**Map for G<sub>0</sub>:**

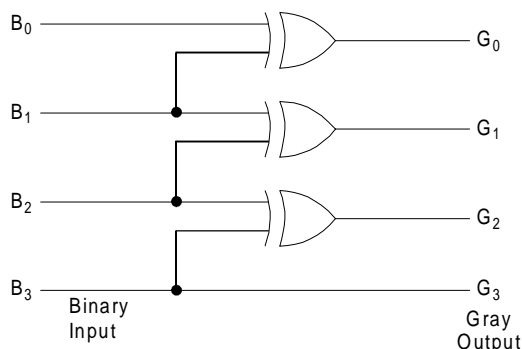
From the two quads, we get

$$G_0 = B_1 B_0' + B_1' B_0$$

$$= B_1 \oplus B_0$$



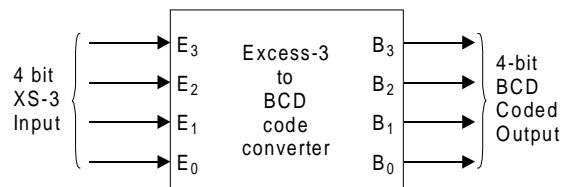
- Now the above expressions can be implemented using X-OR gates to yield the desired code converter circuit shown in Fig. 4.14.



**Fig. 4.14**

Let us see one more example of XS-3 to BCD code converter.

- The block diagram of an XS-3 to BCD code converter is shown in Fig. 4.15. It has four inputs (E<sub>3</sub>, E<sub>2</sub>, E<sub>1</sub>, E<sub>0</sub>) representing 4 bit XS-3 number and four outputs (B<sub>3</sub>B<sub>2</sub> B<sub>1</sub> B<sub>0</sub>) representing 4-bit BCD code.



**Fig. 4.15**

- Truth Table for XS-3 to BCD code converter.

XS-3 codes are obtained from BCD code by adding 3 to each coded number. Moreover 4 binary variables may have 16 combinations, but only 10 are listed. The six not listed are don't care-combinations (since in BCD codes, we use only 10 members

viz. 0, 1, 2, ....9). Since they will never occur, we are at liberty to assign to the output variable either a 1 or a 0, whichever gives a simpler circuit. In this particular example, the unused i/o combinations are listed below the truth table.

Min Terms	Excess-3 Inputs				BCD Outputs				Decimal Equivalent
	$E_3$	$E_2$	$E_1$	$E_0$	$B_3$	$B_2$	$B_1$	$B_0$	
$m_3$	0	0	1	1	0	0	0	0	0
$m_4$	0	1	0	0	0	0	0	1	1
$m_5$	0	1	0	1	0	0	1	0	2
$m_6$	0	1	1	0	0	0	1	1	3
$m_7$	0	1	1	1	0	1	0	0	4
$m_8$	1	0	0	0	0	1	0	1	5
$m_9$	1	0	0	1	0	1	1	0	6
$m_{10}$	1	0	1	0	0	1	1	1	7
$m_{11}$	1	0	1	1	1	0	0	0	8
$m_{12}$	1	1	0	0	1	0	0	1	9
	Unused I/Ps				Outputs				
$m_0$	0	0	0	0	x	x	x	x	
$m_1$	0	0	0	1	x	x	x	x	
$m_2$	0	0	1	0	x	x	x	x	
$m_{13}$	1	1	0	1	x	x	x	x	
$m_{14}$	1	1	1	0	x	x	x	x	
$m_{15}$	1	1	1	1	x	x	x	x	

\* XS-3 is also a class of BCD codes.

3. Now we solve all the BCD outputs. From the truth table, the logic expressions for the BCD coded outputs can be written as :

$$B_3 = \Sigma (m_{11}, m_{12}), d (m_0, m_1, m_2, m_{13}, m_{14}, m_{15})$$

$$B_2 = \Sigma (m_7, m_8, m_9, m_{10}), d (m_0, m_1, m_2, m_{13}, m_{14}, m_{15})$$

$$B_1 = \Sigma (m_5, m_6, m_9, m_{10}), d (m_0, m_1, m_2, m_{13}, m_{14}, m_{15})$$

$$B_0 = \Sigma (m_4, m_6, m_8, m_{10}, m_{12}), d (m_0, m_1, m_2, m_{13}, m_{14}, m_{15}).$$

These expressions can be simplified using k-map →

Map for  $B_3$ →

	$E_1E_0$	00	01	11	10
$E_3E_2$	00	X	X		X
	01				
	11	1	X	X	X
	10			1	

$$\Rightarrow B_3 = E_3 E_2 + E_3 E_1 E_0$$

Map for  $B_2$ →

	$E_1E_0$	00	01	11	10
$E_3E_2$	00	X	X		X
	01			1	
	11		X	X	X
	10	1	1		1

$$\Rightarrow B_2 = E_2' E_0' + E_2 E_1 E_0$$

Map for B<sub>3</sub>→

E <sub>1</sub> E <sub>0</sub> \ E <sub>3</sub> E <sub>2</sub>	00	01	11	10
00	X	X		X
01		1		1
11		X		X
10		1		1

$$\Rightarrow B_1 = E_1' E_0 + E_1 E_0'$$

$$= E_1 \oplus E_0$$

$$\Rightarrow B_3 = E_3 E_2 + E_3 E_1 E_0$$

$$B_2 = E_2' E_0 0' + E_2 E_1 E_0$$

$$B_1 = E_1 \oplus E_0$$

$$B_0 = E_0'$$

Map for B<sub>2</sub>→

E <sub>1</sub> E <sub>0</sub> \ E <sub>3</sub> E <sub>2</sub>	00	01	11	10
00	X	X		X
01	1			1
11	1	X	X	X
10	1			1

$$\Rightarrow B_0 = E_0'$$

4. The expressions for BCD outputs (B<sub>3</sub> B<sub>2</sub> B<sub>1</sub> B<sub>0</sub>) can be implemented for terms of inputs (E<sub>3</sub> E<sub>2</sub> E<sub>1</sub> E<sub>0</sub>) to form a XS-3 to BCD code converter circuit.

The implementation is left as an exercise.

#### 4.1.4 Parity Generators and Checkers

When digital data is transmitted from one location to another, it is necessary to know at the receiving end, whether the received data is free of error. To help make the transmission accurate, special error detection methods are used.

To detect errors, we must keep a constant check on the data being transmitted. To check accuracy we can generate and transmit an extra bit along with the message (data). This extra bit is known as the parity bit and it decides whether the data transmitted is error free or not. There are two types of parity bits, namely even parity and odd parity that we have discussed in chapter 1 under error detecting codes.

Fig. 4.16 shows an error detecting circuit using a parity bit.

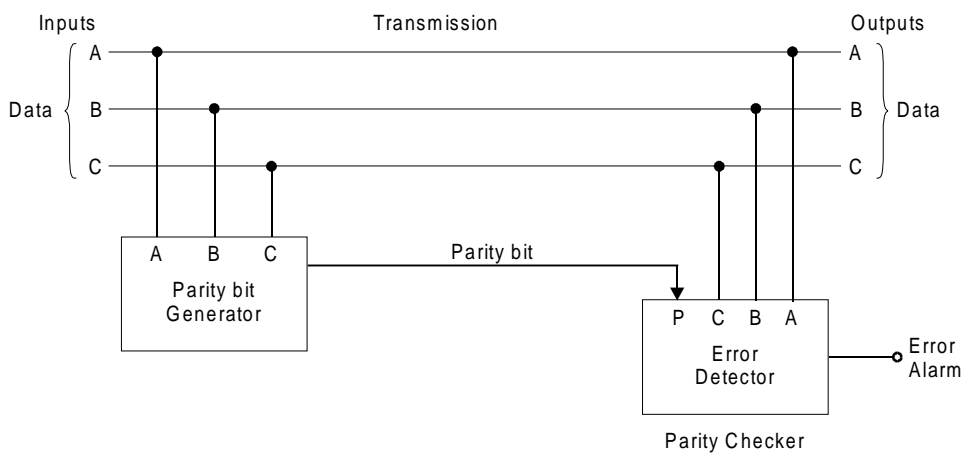


Fig. 4.16

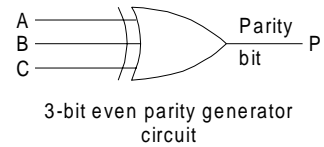
In this system three parallel bits A, B and C and being transmitted over a long distance. Near the input they are fed into a parity bit generator circuit. This circuit generates what is called a parity bit. It may be either even or odd. For example, if it is a 3-bit even parity generator, the parity bit generated is such that it makes total member of 1s even. We can make a truth table of a 3-bit even parity generator circuit.

Truth Table for a 3-bit even parity generator.

Inputs Data			Output Even parity bit
A	B	C	P
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	1
1	1	1	1

Next, the truth table is converted to a logic circuit shown in Fig. 4.17.

$$\begin{aligned}
 P &= A B' C + A B C' + A B' C' + A B C \\
 &= A' (B' C + B C) + A (B' C' + B C) \\
 &= A' (B \oplus C) + A (B \oplus C) \\
 &= A' (B \oplus C) + A (B \oplus C)' \\
 &= A \oplus (B \oplus C) = (A \oplus B) \oplus C = A \oplus B \oplus C.
 \end{aligned}$$



**Fig. 4.17**

The generated parity bit is transmitted with the data and near the output it is fed to the error detector (parity checker) circuit. The detector circuit checks for the parity of transmitted data. As soon as the total number of 1's in the transmitted data are found 'odd' it sounds an alarm, indicating an error. If total member of 1s are even, no alarm sounds, indicating no error.

In above example we are transmitting 4 bits. (3 bits of message plus 1 even parity bit). So, it is easy to understand that. Error detector is nothing but a 4 bit even-parity checker circuit. Fig. 4.18 (a) shows a truth table of a 4 bit even parity checker circuit.

Inputs Transmitted Data with parity bit				Outputs Parity error check
A	B	C	P	E
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0

(Contd.)

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

Fig. 4.18 (a)

Now below : We convert this truth table into logic circuit shown in Fig. 4.18(b).

$$\begin{aligned}
 E &= A'B'CP + A'B'CP' + A'BCP' + A'BCP + \\
 &\quad AB'CP' + AB'CP + ABCP' + ABCP \\
 &= A'B'(C \oplus P) + A'B(C \oplus P)' + AB'(C \oplus P)' + AB(C \oplus P) \\
 &= (C \oplus P)(A \oplus B)' + (C \oplus P)'(A \oplus B) \\
 &= (A \oplus B) \oplus (C \oplus P)
 \end{aligned}$$

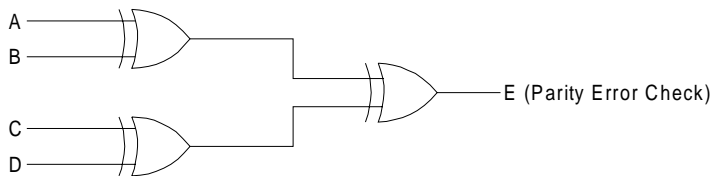


Fig. 4.18 (b) 4-bit even parity checker.

If E = 1, Alarm sounds means error.

If E = 0, No alarm sounds means no error.

Now, it is possible to implement the parity generator with the circuit of parity checker. If the input P is connected to logic-0, causing the value of C to pass through the gate unchanged. (because  $C \oplus 0 = C$ ). The advantage of this is that the same circuit can be used for both parity generation and checking.

### 4.2 MSI AND LSI CIRCUITS

When designing logic circuits, the “discrete logic gates”; *i.e.*, individual AND, OR, NOT *etc.* gates, are often neither the simplest nor the most economical devices we could use. There are many standard MSI (medium scale integrated) and LSI (large scale integrated) circuits, or functions available, which can do many of the things commonly required in logic circuits. Often these MSI and LSI circuits do not fit our requirements exactly, and it is often necessary to use discrete logic to adapt these circuits for our application.

However, the number and type of these LSI and VLSI (very large scale integrated) circuits is steadily increasing, and it is difficult to always be aware of the best possible circuits available for a given problem. Also, systematic design methods are difficult to devise when the



types of logic device available keeps increasing. **In general the “best” design procedure is to attempt to find a LSI device which can perform the required function, or which can be modified using other devices to perform the required function.** If nothing is available, then the function should be implemented with several MSI devices. Only as a last option should the entire function be implemented with discrete logic gates. In fact, with present technology, it is becoming increasingly cost-effective to implement a design as one or more dedicated VLSI devices.

When designing all but the simplest logic devices, a “top-down” approach should be adopted. The device should be specified in block form, and attempt to implement each block with a small number of LSI or MSI functions. Each block which cannot be implemented directly can be then broken into smaller blocks, and the process repeated, until each block is fully implemented.

Of course, **a good knowledge of what LSI and MSI functions are available in the appropriate technology makes this process simpler.**

#### 4.2.1 The Digital Multiplexer

One MSI function which has been available for a long time is the digital selector, or multiplexer. It is the digital equivalent of the rotary switch or selector switch (*e.g.*, the channel selector on a TV set). Its function is to accept a binary number as a “selector input,” and present the logic level connected to that input line as output from the data selector.

A digital multiplexer (MUX) is a combinational circuits that selects one input out of several inputs and direct it to a single output. The particular input selection is controlled by a set of select inputs. Fig. 4.19 shows block diagram of a digital multiplexer with  $n$  inputs lines and single output line.

For selecting one out of  $n$  input, a set of  $m$  select inputs is required where

$$n = 2^m$$

On the basis of digital (binary) code applied at the select inputs, one output of  $n$  data sources is selected. Usually, an enable (or strobe) input (E) is built-in for cascading purpose. Enable input is generally active-low, *i.e.*, it performs its intended operation when it is low (logic).

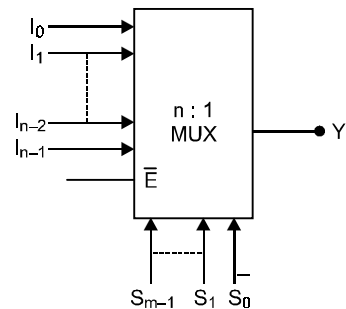
**Note.** 16:1 are the largest available ICs, therefore for larger input requirements there should be provision for expansion. This is achieved through enable/strobe input (multiplexer stacks or trees are designed).

A circuit diagram for a possible 4-line to 1-line data selector/multiplexer (abbreviated as MUX for multiplexer) is shown in Fig. 4.20. Here, the output  $Y$  is equal to the input  $I_0$ ,  $I_1$ ,  $I_2$ ,  $I_3$  depending on whether the select lines  $S_1$  and  $S_0$  have values 00, 01, 10, 11 for  $S_1$  and  $S_0$  respectively. That is, the output  $Y$  is *selected* to be equal to the input of the line given by the binary value of the select lines  $S_1S_0$ .

The logic equation for the circuit shown in Fig. 4.20 is:

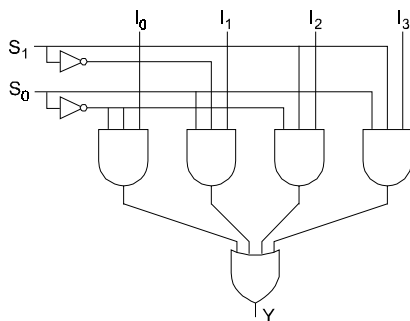
$$Y = I_0 \cdot \bar{S}_1 \cdot \bar{S}_0 + I_1 \cdot \bar{S}_1 \cdot S_0 + I_2 \cdot S_1 \cdot \bar{S}_0 + I_3 \cdot S_1 \cdot S_0$$

This device can be used simply as a data selector/multiplexer, or it can be used to perform logic functions. Its simplest application is to implement a truth table directly; *e.g.*,



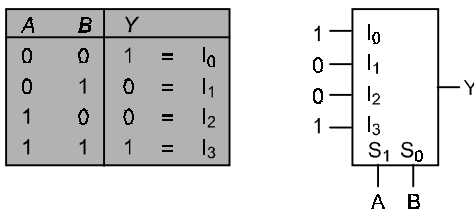
**Fig. 4.19** Block diagram of the digital multiplexer.

with a 4 line to 1 line MUX, it is possible to implement any 2-variable function directly, simply by connecting  $I_0, I_1, I_2, I_3$  to logic 1 in logic 0, as dictated by a truth table. In this way, a MUX can be used as a simple look-up table for switching functions. This facility makes the MUX a very general purpose logic device.



**Fig 4.20** A four-line to 1-line multiplexer

**Example.** Use a 4 line to 1 line MUX to implement the function shown in the following truth table ( $Y = \bar{A}.\bar{B} + A.B$ ).



**Fig. 4.21** A 4-line to 1-line MUX implementation of a function of 2 variables

Simply connecting  $I_0 = 1, I_1 = 0, I_2 = 0, I_3 = 1$ , and the inputs A and B to the  $S_1$  and  $S_0$  selector inputs of the 4-line to 1-line MUX implement this truth table, as shown in Fig. 4.21.

The 4-line to 1-line MUX can also be used to implement any function of three logical variables, as well. To see this, we need note only that the only possible functions of one variable C, are C,  $\bar{C}$ , and the constants 0 or 1. (i.e., C,  $\bar{C}$ ,  $C + \bar{C} = 1$ , and 0). We need only connect the appropriate value, C,  $\bar{C}$ , 0 or 1, to  $I_0, I_1, I_2, I_3$  to obtain a function of 3 variables. The MUX still behaves as a table lookup device; it is now simply looking up values of another variable.

**Example.** Implement the function

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

Using a 4-line to 1-line MUX.

Here, again, we use the A and B variables as data select inputs. We can use the above equation to construct the table shown in Fig. 4.22. The residues are what is “left over” in each minterm when the “address” variables are taken away. To implement this circuit, we connect  $I_0$  and  $I_3$  to C, and  $I_1$  and  $I_2$  to  $\bar{C}$ , as shown in Fig. 4.22.

Input	"Address"	Other variables (residues)
$l_0$	$\bar{A} \cdot \bar{B}$	C
$l_1$	$\bar{A} \cdot B$	$\bar{C}$
$l_2$	$A \cdot \bar{B}$	$\bar{C}$
$l_3$	$A \cdot B$	C

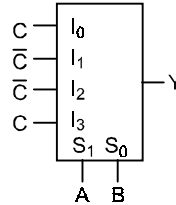


Fig. 4.22 A 4-line to 1-line MUX implementation of a function of 3 variables

In general a 4 input MUX can give any function of 3 inputs, an 8 input MUX can give any functional of 4 variables, and a 16 input MUX, any function of 5 variables.

**Example.** Use an 8 input MUX to implement the following equation:

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

Again, we will use A, B, C as data select inputs, or address inputs, connected to  $S_2$ ,  $S_1$  and  $S_0$ , respectively.

Input	"Address"	Residues
$l_0$	$\bar{A} \cdot \bar{B} \cdot \bar{C}$	$\bar{D}$
$l_1$	$\bar{A} \cdot \bar{B} \cdot C$	D
$l_2$	$\bar{A} \cdot B \cdot \bar{C}$	$D + \bar{D} = 1$
$l_3$	$\bar{A} \cdot B \cdot C$	D
$l_4$	$A \cdot \bar{B} \cdot \bar{C}$	$\bar{D}$
$l_5$	$A \cdot \bar{B} \cdot C$	$\bar{D}$
$l_6$	$A \cdot B \cdot \bar{C}$	$\bar{D} + D = 1$
$l_7$	$A \cdot B \cdot C$	D

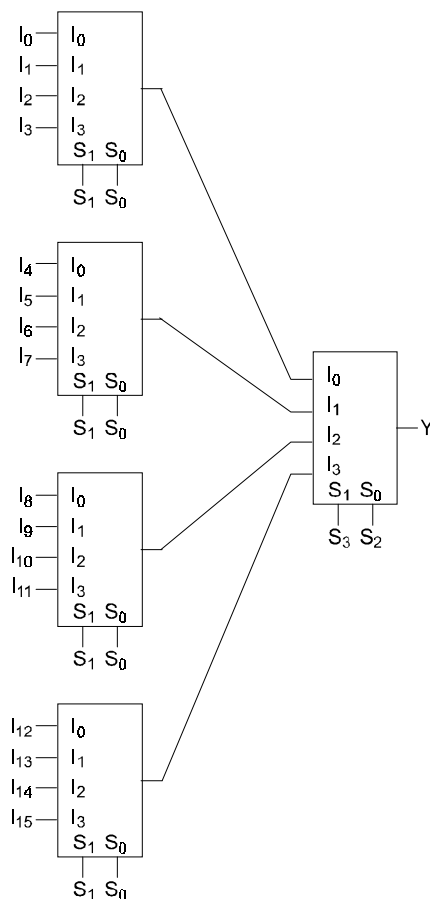


Fig. 4.23 An 8-line to 1-line MUX implementation of a function of 4 variables

Values of the address set A, B, C with no residues corresponding to the address in the above table must have logic value 0 connected to the corresponding data input. The select variables A, B, C must be connected to  $S_2$ ,  $S_1$  and  $S_0$  respectively. A circuit which implements this function is shown in Fig. 4.23.

This use of a MUX as a "table look up" device can be extended to functions of a larger number of variables; the MUX effectively removes the terms involving the variables assigned to its select inputs from the logic expression. This can sometimes be an effective way to reduce the complexity of implementation of a function. For complex functions, however, there are often better implementations, as we use PLDs (see chapter 5).

Although it is obvious how the function shown in Fig. 4.20 can be extended a  $2^n$  line to 1 line MUX, for any  $n$ , in practice, about the largest devices available are only to 16 line to 1 line functions. It is possible to use a "tree" of smaller MUX's to make arbitrarily large MUX's. Fig. 4.24 shows an implementation of a 16 line to 1 line MUX using five 4 line to 1 line MUX's.



**Fig. 4.24** A 16-line to 1-line MUX made from five 4-line to 1-line MUX's

#### 4.2.2 Decoders (Demultiplexers)

Another commonly used MSI device is the decoder. Decoders, in general, transform a set of inputs into a different set of outputs, which are coded in a particular manner; *e.g.*, certain decoders are designed to decode binary or BCD coded numbers and produce the correct output to display a digit on a 7 segment (calculator type) display. Decoders are also available to convert numbers from binary to BCD, from binary to hexadecimal, etc.

Normally, however, the term “decoder” implies a device which performs, in a sense, the inverse operation of a multiplexer. A decoder accepts an  $n$  digit number as its  $n$  “select” inputs and produces an output (usually a logic 0) at one of its  $2^n$  possible outputs. Decoders are usually referred to as  $n$  line to  $2^n$  line decoders; *e.g.* a 3 line to 8 line decoder. This type of decoder is really a binary to unary number system decoder. Most decoders have inverted outputs, so the selected output is set to logic 0, while all the other outputs remain at logic 1. As well, most decoders have an “enable” input  $\bar{E}$ , which “enables” the operation of the decoder—when the  $\bar{E}$  input is set to 0, the device behaves as a decoder and selects the output determined by the select inputs; when the  $\bar{E}$  input is set to 1, the outputs of the decoder are *all* set to 1. (The bar over the E indicates that it is an “active low” input; that is, a logic 0 enables the function).

The enable input also allows decoders to be connected together in a treelike fashion, much as we saw for MUX's, so large decoders can be easily constructed from smaller devices. The enable input also allows the decoder to perform the inverse operation of a MUX; a MUX selects as output one of  $2^n$  inputs, the decoder can be used to present an input to one of  $2^n$  outputs, simply by connecting the input signal to the  $\bar{E}$  input; the signal at the selected output will then be the same as the input at  $\bar{E}$ —this application is called “demultiplexing.” The demultiplexer (DEMUX) performs the reverse operation of a multiplexer. A demultiplexer is a circuit that accepts single input and transmit it over several (one of  $2^n$  possible) outputs.

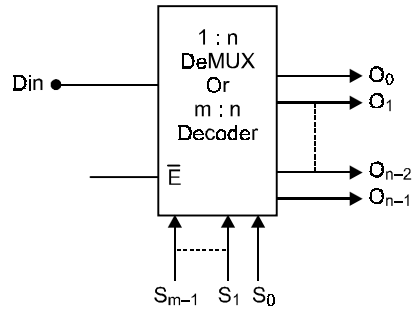


Fig. 4.25 Block diagram of the demultiplexer/decoder

In the block diagram (Fig. 4.25) a demultiplexer, the number of output lines is  $n$  and the number of select lines is  $m$ .

where  $n = 2^m$

One the basis of select input code, to which output the data will be transmitted is determined. There is an active-low (low-logic) enable/data input. The output for these devices are also active-low.

**Note.** 4-line to 16-line decoders are the largest available circuits in ICs.

A typical 3 line to 8 line decoder with an enable input behaves according to the following truth table, and has a circuit symbol as shown in Fig. 4.26.

$E$	$S_2$	$S_1$	$S_0$	$O_0$	$O_1$	$O_2$	$O_3$	$O_4$	$O_5$	$O_6$	$O_7$
1	x	x	x	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1
0	1	1	0	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	0

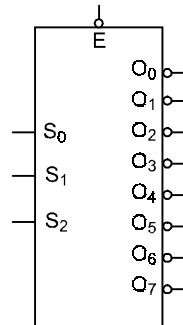


Fig. 4.26 An 3-line to 8-line decoder

Note that, when the  $\bar{E}$  input is enabled, an output of 0 is produced corresponding to each minterm of  $S_2, S_1, S_0$ . These minterm can be combined together using other logic gates to form any required logic function of the input variables. In fact, several functions can be produced at the same time. If the selected output was a logic 1, then the required minterms could simply be ORed together to implement a switching function directly from its minterm form. Using de Morgans theorem, we can see that when the outputs are inverted, as is normally the case, then the minterm form of the function can be obtained by NANDing the required terms together.

**Example.** An implementation the functions defined by the following truth table using a decoder and NAND gates is shown in Fig. 4.27.

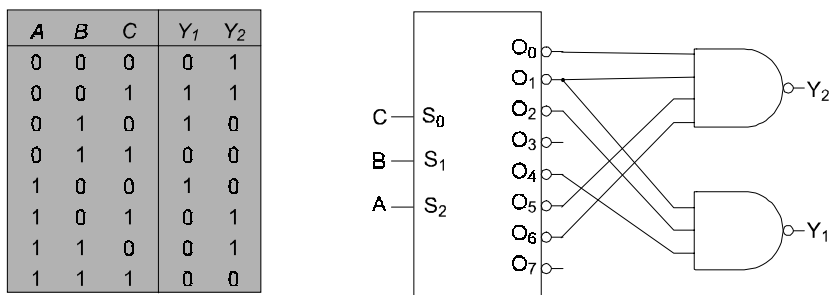


Fig. 4.27

**IMPLEMENTATION EXAMPLES OF COMBINATIONAL LOGIC DESIGN USING MUX/DEMUX**

We have already seen how to implement combinational circuits using MUX/DEMUX. The standard ICs available for multiplexers are 2:1, 4:1, 8:1 and 16:1. The different digital ICs are given in appendix B, but for sake of convenience some of the MUX/DEMUX ICs are given here in Tables A and B.

**Table A: Standard multiplexer ICs**

IC No.	Description	Output
74157	Quad. 2:1 Multiplexer	Same as input
74158	Quad 2:1 MUX	Inverted input
74153	Dual 4:1 MUX	Same as input
74352	Dual 4:1 MUX	Inverted input
74151A	8:1 MUX	Complementary outputs
74152	8:1 MUX	Inverted input
74150	16:1 MUX	Inverted input

**Table B: Standard Demultiplexer/Decoder ICs**

IC No.	Description	Output
74139	Dual 1:4 Demultiplexer (2-line-to-4-line decoder)	Inverted input
74155	Dual 1:4 Demultiplexer (2-line-to-4-line decoder)	1Y-Inverted I/P 2Y-Same as I/P
74138	1:8 Demultiplexer (3-line-to-8-line decoder)	Inverted I/P
74154	1:16 Demultiplexer (4-line-to-16-line decoder)	Same as input

When using the multiplexer as a logic element either the truth table or one of the standard forms of logic expression must be available. The design procedure for combinational circuits using MUX are as follows:

**STEP 1:** Identify the decimal number corresponding to each minterm in the expression. The input lines corresponding to these numbers are to be connected to logic 1 (high).

**STEP 2 :** All other input lines except that used in step 1 are to be connected to logic 0 (low).

**STEP 3 :** The control inputs are to be applied to select inputs.

**Example.** Implement the following function with multiplexer:

$$Y = F(A, B, C, D) = \Sigma m(0, 1, 3, 4, 8, 9, 15)$$

**Solution. STEP 1 :** The input lines corresponding to each minterms (decimal number) are to be connected to logic 1.

Therefore input lines 0, 1, 3, 4, 8, 9, 15 have to be connected to logic 1.

**STEP 2 :** All other input lines except 0, 1, 3, 4, 8, 9, 15 are to be connected to logic 0.

**STEP 3 :** The control inputs A, B, C, D are to be applied to select inputs.

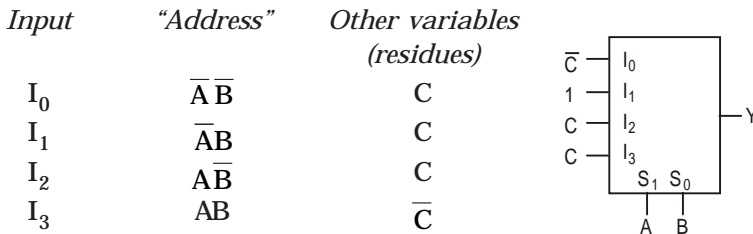
**Note:** Although the given procedure is simple to implement but the 16 to 1 multiplexers are the largest available ICs, therefore to meet the larger input needs there should be provision for expansion. This is achieved with the help of enable/stroke inputs and multiplexer stacks or trees are designed.

**Example.** Implement the following function with a 4x1 multiplexer.

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

**Solution.** Given  $Y = F(A, B, C) = \Sigma m(1, 3, 5, 6)$   
 $= \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC$

We use the A and B variables as data select inputs. We can use the above equation to construct the table shown in Fig. 4.28. The residues are what is “left over” in each minterm when the “address” variables are taken away.



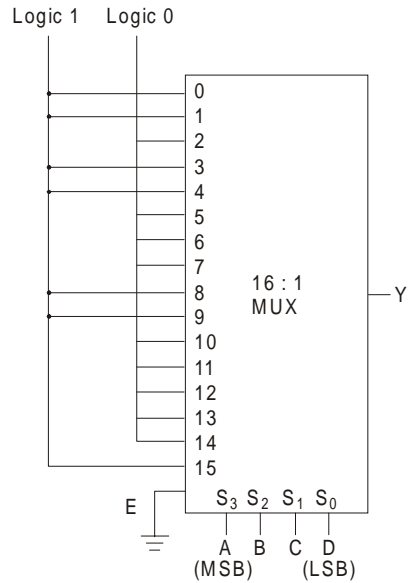
**Fig. 4.28** A 4-line to 1-line MUX implementation of a function of 3 variables.

To implement this circuit, we connect I<sub>0</sub>, I<sub>1</sub> and I<sub>2</sub> to C and I<sub>3</sub> to  $\bar{C}$  as shown in Fig. 4.28.

**Example.** Using four-input multiplexer, implement the following function

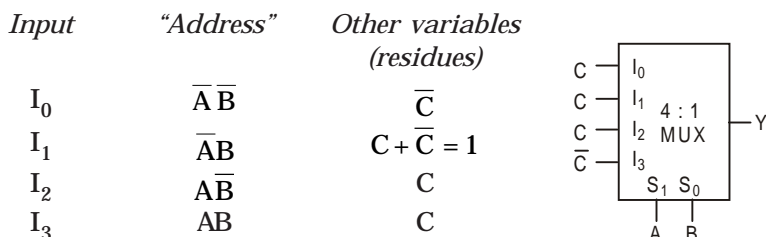
$$Y = F(A, B, C) = \Sigma m(0, 2, 3, 5, 7)$$

Control variables A, B.



**Solution.** Given  $Y = F(A, B, C) = \Sigma m(0, 2, 3, 5, 7)$   
 $= \overline{A}\overline{B}\overline{C} + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\overline{C} + ABC$

We can use the above equation to construct the table shown in Fig. 4.29. The residues are what is “left over” in each minterm when the “address/control” variables are taken away.



**Fig. 4.29** A 4-line to 1-line MUX implementation of a function of 3 variables

To implement this function, we connect  $I_0$  to  $\overline{C}$ ,  $I_1$  to 1 and  $I_2$  and  $I_3$  to  $C$ , as shown in Fig. 4.29.

**Example.** Design a full adder using 8:1 multiplexer.

**Solution.** The truth table of a full adder is given as

A	B	C	S	$C_F$
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

$$S(A, B, C) = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC = \Sigma m(1, 2, 4, 7)$$

$$C_F(A, B, C) = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC = \Sigma m(3, 5, 6, 7)$$

The implementation for summation expression is

**Step 1:** The input lines corresponding to 1, 2, 4, 7 are to be connected to logic 1.

**Step 2:** Other input lines are to be connected to logic 0.

**Step 3:** Control inputs A, B, C are to be applied to select inputs. Fig. 4.30 A.

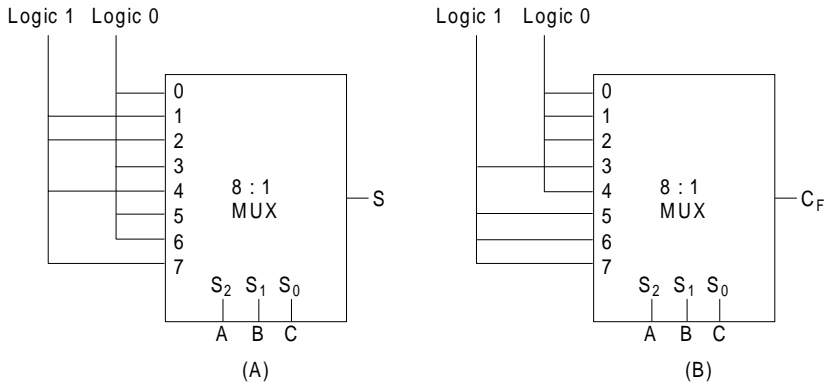
Similarly for carry expression.

**Step 1:** The input lines corresponding to 3, 5, 6, 7 are to be connected to logic 1.

**Step 2:** Other input lines are to be connected to logic 0.

**Step 3:** Control inputs A, B, C are to be applied to select inputs. Fig. 4.30 B.





**Fig. 4.30** Full adder implementation using 8:1 Multiplexer.

**Example.** Implement a full adder with a decoder and two OR-gates.

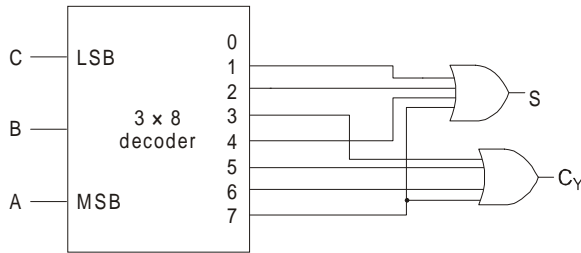
**Solution.** From the previous example we note that expression for summation is given by

$$S(A, B, C) = \Sigma m(1, 2, 4, 7)$$

and expression for carry is given by

$$C_F(A, B, C) = \Sigma m(3, 5, 6, 7)$$

The combinational logic of full adder can be implemented with due help of 3-line to 8-line decoder/1:8 demultiplexer as shown in Fig. 4.31.



**Fig. 4.31** Full adder implementation using 3 x 8 decoder.

**Example.** A combinational circuit is defined by the following Boolean functions. Design circuit with a decoder and external gates.

**Solution.**  $Y_1 = F_1(A, B, C) = \overline{A}\overline{B}\overline{C} + AC$

$$Y_2 = F_2(A, B, C) = A\overline{B}C + \overline{A}C$$

Given  $Y_1 = \overline{A}\overline{B}\overline{C} + AC$

First we have to write the expression in minterms, if the expression is not in the form of minterms by using  $(x + \overline{x} = 1)$

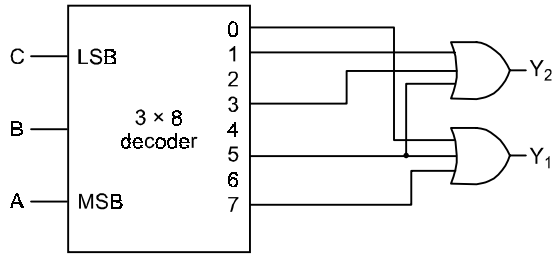


Fig. 4.32 Function implementation using 3x8 decoder.

Therefore

$$\begin{aligned}
 Y_1 &= \bar{A}\bar{B}\bar{C} + AC \\
 Y_1 &= \bar{A}\bar{B}\bar{C} + AC(B + \bar{B}) \\
 Y_1 &= \bar{A}\bar{B}\bar{C} + ABC + A\bar{B}C \\
 Y_1 &= \Sigma m(0, 5, 7) \\
 Y_2 &= \bar{A}\bar{B}C + \bar{A}C \\
 Y_2 &= \bar{A}\bar{B}C + \bar{A}C(B + \bar{B}) \\
 Y_2 &= \bar{A}\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C \\
 Y_2 &= \Sigma m(1, 3, 5)
 \end{aligned}$$

The combinational logic for the boolean function can be implemented with the help of 3-line to 8-line decoder as shown in Fig 4.32.

**Example.** Realise the given function using a multiplexer

$$Y(A, B, C, D) = \Pi M(0, 3, 5, 9, 11, 12, 13, 15)$$

**Solution.** To implement the given function, first we have to express the function in terms of sum of product. i.e.,

$$Y(A, B, C, D) = \Sigma m(1, 2, 4, 6, 7, 8, 10, 14)$$

Now the given function in this form can be realized as

**Step 1:** Input lines corresponding to 1, 2, 4, 6, 7, 8, 10, 14 are to be connected to logic 1.

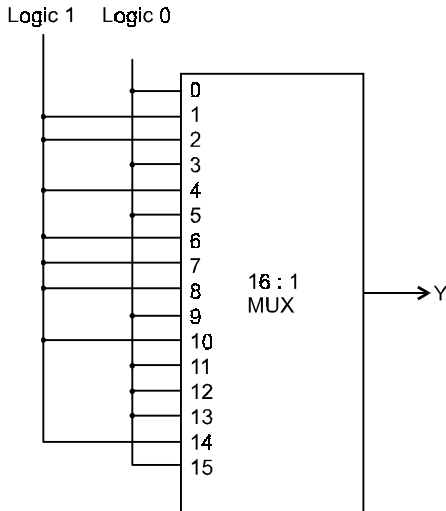


Fig. 4.33 A 16-line to 1-line MUX implementation.

**Step 2:** Other input lines are to be connected to logic 0.

**Step 3:** Control inputs A, B, C, D are to be applied to select inputs.

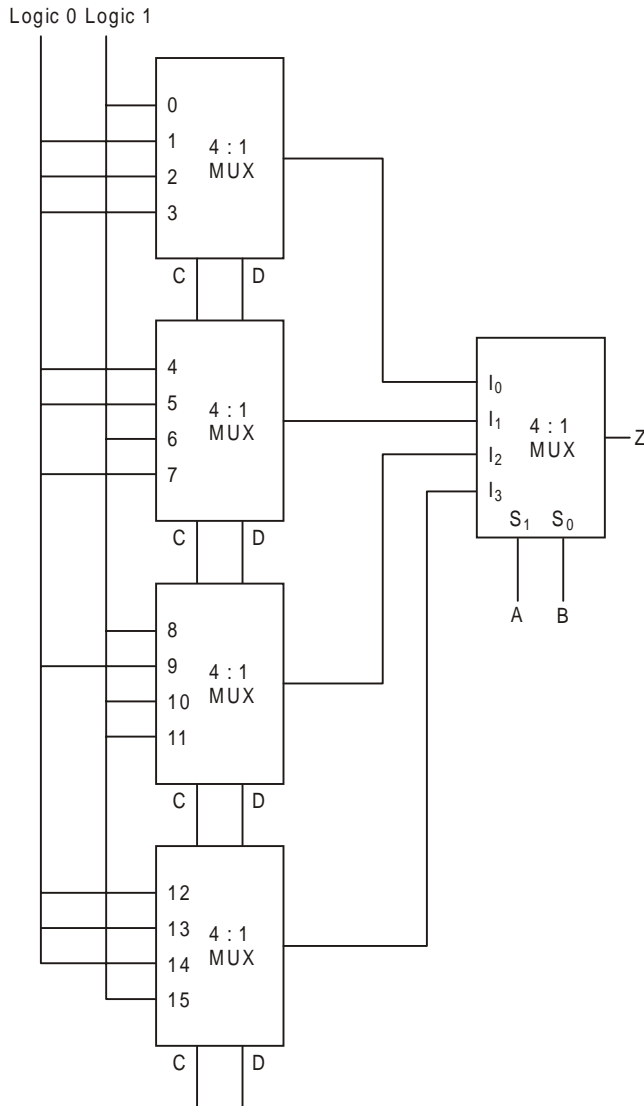
**Example.** Realize the following boolean expression using 4:1 MUX(S) only.

$$Z = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}BC\overline{D} + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}\overline{C}D + ABCD$$

**Solution.** Given  $Z = \Sigma m (0, 6, 8, 10, 11, 15)$

To implement the given boolean expression we must have 16 input and 4 selection inputs.

Since 4:1 mux has 4 input lines and two selection lines. Therefore we can use 4, 4:1 MUX with their select lines connected together. This is followed by a 4:1 MUX to select one of the four outputs. The select lines of the 4:1 MUX (final) are driven from inputs A, B. The complete circuit is shown in Fig. 4.34.



**Fig. 4.34** A 4-line to 1-line MUX implementation of a function of 4 variable.

### 4.2.3 Encoders

The encoder is another example of combinational circuit that performs the inverse operation of a decoder. It is designed to generate a different output code for each input which becomes active. In general, the encoder is a circuit with  $m$  input lines ( $m \leq 2^n$ ) \* (\*  $m < 2^n \rightarrow$  If unused input combinations occur.) and  $n$  output lines that converts an active input signal into a coded output signal. In an encoder, the number of outputs is less than the number of inputs. The block diagram of an encoder is shown in Fig. 4.35.

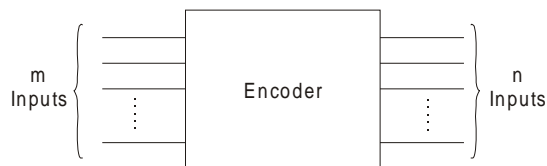


Fig. 4.35 Block Diagram of an Encoder

An example of an encoder is an octal to binary encoder. An octal to binary encoder accepts eight inputs and produces a 3-bit output code corresponding to the activated input. The truth table for the octal to binary encoder is shown in table.

Inputs								Outputs		
$O_0$	$O_1$	$O_2$	$O_3$	$O_4$	$O_5$	$O_6$	$O_7$	$Y_2$	$Y_1$	$Y_0$
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

It has eight inputs, one for each octal digit and three outputs that generate the corresponding binary number.

The truth table shows that  $Y_0$  must be 1 whenever the input  $O_1$  or  $O_3$  or  $O_5$  or  $O_7$  is high. Thus,

$$Y_0 = O_1 + O_3 + O_5 + O_7$$

Similarly

$$Y_1 = O_2 + O_3 + O_6 + O_7 \text{ and}$$

$$Y_2 = O_4 + O_5 + O_6 + O_7.$$

Using these three expressions, the circuit can be implemented using three 4-input OR gates as shown in Fig. 4.36.

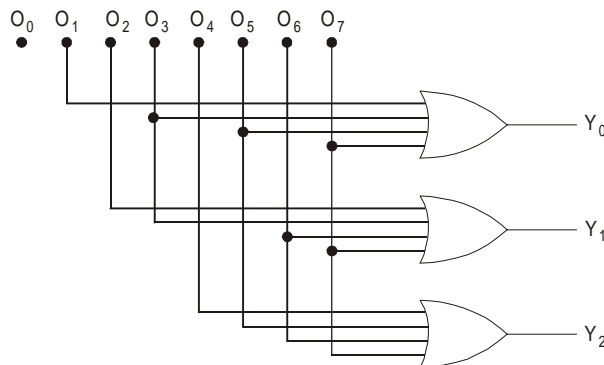


Fig. 4.36 Octal to binary encoder.

The encoder has two limitations:

1. Only one input can be active at any given time. If two or more inputs are equal to 1 at the same time, the O/P is undefined. For example if  $O_2$  and  $O_5$  are active simultaneously, the o/p of encoder will be 111 that is equal to binary 7. This does not represent binary 2 or 5.
2. The output with all 0's is generated when all inputs are '0', and is also true when  $O_0 = '1'$ .

The first problem is taken care by a circuit, called as 'priority encoder'. It establishes a priority to ensure that only one input is active (High) at a given time.

The second problem is taken care by an extra line in the encoder output, called 'valid output indicator' that specifies the condition that none of the inputs are active.

### Priority Encoder

A priority encoder is an encoder that includes priority function. If two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence. To understand priority encoder, consider a 4 to 2 line encoder which gives priority to higher subscript number input than lower subscript number. The truth table is given below.

Truth Table of 4 to 2 line priority encoder:

Inputs				Outputs		
$D_0$	$D_1$	$D_2$	$D_3$	$Y_1$	$Y_2$	$V$
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

The Xs are don't care conditions. Input  $D_3$  has the highest priority, so regardless of values of other inputs, when this input is 1, the output  $Y_1 Y_2 = 11$ .  $D_2$  has next priority level. The o/p is 10 if  $D_2$  is 1, provided  $D_3 = 0$ , irrespective of the values of the other two lower-priority inputs. The o/p is 01 if  $D_1$  is 1, provided both  $D_2$  and  $D_3$  are 0, irrespective of the value of lower-priority input  $D_0$ . The o/p is  $D_0$  if  $00 = 1$ , provided all other inputs are 0.

A valid output indicator,  $V$  is set to 1, only when one or more of the inputs are equal to 1. If all the inputs are 0,  $V$  is equal to 0. and the other two outputs if the circuit are not used.

Now, simplifying using k-map the outputs can be written as :

$$Y_1 = D_2 + D_3$$

$$Y_2 = D_3 + D_1 D_2'$$

$$V = D_0 + D_1 + D_2 + D_3.$$

The logic diagram for a 4 to 2 line priority encoder with 'valid output indicator' is shown below in Fig. 4.37.

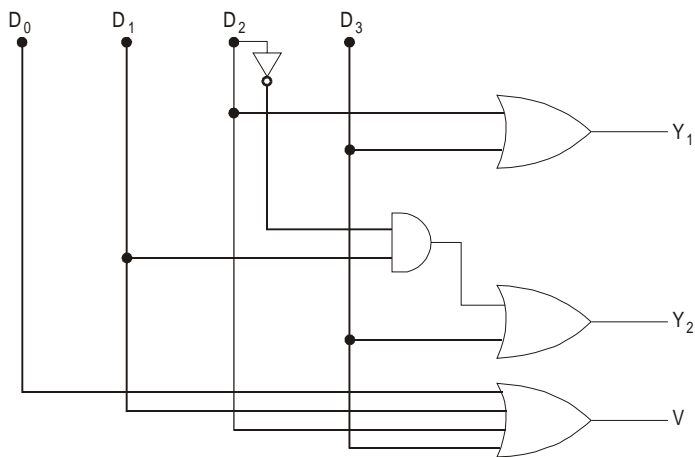


Fig. 4.37

### 4.2.4 Serial and Parallel Adders

In section 4.1.1 we have discussed the full-adder circuit. Full adder is a combinational circuit that adds three binary digits. When we add two numbers of any length, the terms we have to deal with are :

Input carry, Augend, Addend, sum and output carry. We simply start adding two binary digits from LSB (rightmost positioned bits). At this position, the input carry is always equal to zero. After addition, we get sum and output carry. This output carry works as the input carry to the next higher positioned augend and addend bits. Next we add augend and addend bits along with the input carry that again produces sum and output carry. The process repeats upto MSB position (leftmost positioned bits).

We observe that in the process of addition we are actually adding three digits – the input carry, the augend bit and the addend bit. And, we are getting two outputs the sum and the output carry.

This can be illustrated by the following example. Let the 4-bits words to be added be represented by:

$$A_3 A_2 A_1 A_0 = 1 1 0 1 \text{ and } B_3 B_2 B_1 B_0 = 0 0 1 1.$$

Significant Place	4	3	2	1	
Input Carry	1	1	1	0	← Carry-In
Augend Word	1	1	0	1	
Addend Word	0	0	1	1	
Sum output carry	0	0	0	0	
Carry-out	1	1	1	1	

Now if we compare this with the full adder circuit, we can easily observe that the two inputs (A and B) are augend and addend bits with the third input (c) as the input carry. Similarly two outputs are sum (s) and output carry (C<sub>0</sub>).

In general, the sum of two *n*-bit numbers can be generated by using either of the two methods : the serial addition and the parallel addition.

### Serial Adder

In serial addition, the addition operation is carried out bit by bit. The serial adder uses one full adder circuit and some storage device (memory element) to hold generated output carry. The diagram of a 4 bit serial adder is shown in Fig. 4.38.

The two bits at the same positions in augend and addend word are applied serially to A and B inputs of the full adder respectively. The single full adder is used to add one pair of bits at a time along with the carry  $C_{in}$ . The memory element is used to store the carry output of the full adder circuit so that it can be added to the next significant position of the numbers in the augend and addend word. This produces a string of output bits for sum as  $S_0, S_1, S_2$  and  $S_3$  respectively.

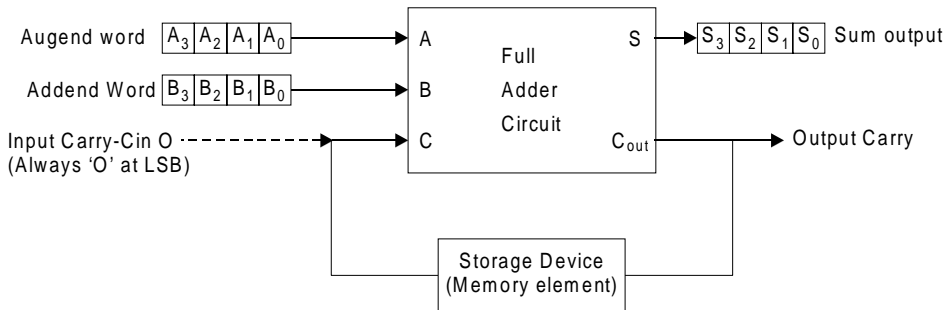


Fig. 4.38 4-bit serial adder.

### Parallel Adder

To add two n-bit numbers, the parallel method uses n full adder circuits and all bits of addend and augend bits are applied simultaneously. The output carry from one full adder is connected to the input carry of the full adder one position to its left.

The 4-bit adder using full adder circuit is capable of adding two 4-bit numbers resulting in a 4-bit sum and a carry output as shown in Fig. 4.39.

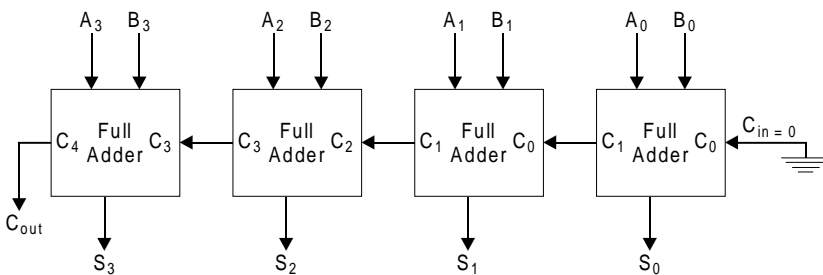


Fig. 4.39 4-bit binary parallel adder.

The addition operation is illustrated in the following example. Let the 4-bit words to be added be represented by  $A_3 A_2 A_1 A_0 = 1 0 1 0$  and  $B_3 B_2 B_1 B_0 = 0 0 1 1$ .

Subscript $i$	3	2	1	0
Input carry $C_i$	0	1	0	0
Augend $A_i$	1	0	1	0
Addend $B_i$	0	0	1	1
Sum $S_i$	1	1	0	1
Output carry $C_{i+1}$	0	0	1	0

← Significant place.

In a 4-bit parallel adder, the input to each full adder will be  $A_i$ ,  $B_i$  and  $C_i$  and the outputs will be  $S_i$  and  $C_{i+1}$ , where  $i$  varies from 0 to 3.

In the least significant stage,  $A_0$ ,  $B_0$  and  $C_0$  (which is 0) are added resulting in sum  $S_0$  and carry  $C_1$ . This carry  $C_1$  becomes the carry input to the second stage. Similarly, in the second stage,  $A_1$ ,  $B_1$  and  $C_1$  are added resulting in  $S_1$  and  $C_2$ ; in the third stage,  $A_2$ ,  $B_2$  and  $C_2$  are added resulting in  $S_2$  and  $C_3$ ; in the fourth stage  $A_3$ ,  $B_3$  and  $C_3$  are added resulting in  $S_3$  and  $C_4$  which is the output carry. Thus the circuit results in a sum ( $S_3, S_2, S_1, S_0$ ) and a carry output ( $C_{out}$ ).

An alternative block diagram representation of a 4 bit binary parallel adder is shown in Fig. 4.40.

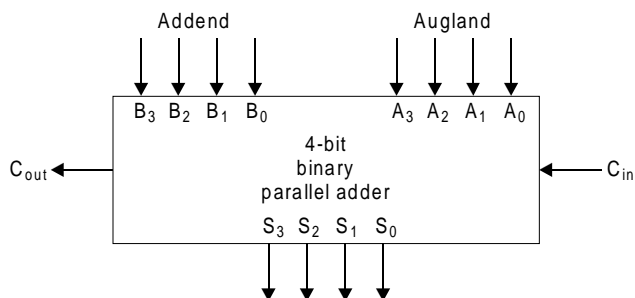


Fig. 4.40 4-bit binary parallel adder.

**Propagation Delay:** Though the parallel binary adder is said to generate its output immediately after the inputs are applied, its speed of operation is limited by the carry propagation delay through all stages. In the parallel binary adder, the carry generated by the adder is fed as carry input to  $(i + 1)$ th adder. This is also called as 'ripple carry adder'. In such adders, the output ( $C_{out}, S_3, S_2, S_1, S_0$ ) is available only after the carry is propagated through each of the adders, i.e., from LSB to MSB adder through intermediate adders. Hence, the addition process can be considered to be complete only after the carry propagation delay through adders, which is proportional to number of stages in it; one of the methods of speeding up this process is look-ahead carry addition, which eliminates the ripple carry delay. This method is based on the carry generating and the carry propagating functions of the full adder.

### 4-bit Look-ahead Carry Generator

The look-ahead carry generator is based on the principle of looking at the lower order bits of addend and augend if a higher order carry is generated. This reduces the carry delay by reducing the number of gates through which a carry signal must propagate. To explain its operation consider the logic diagram of a full adder circuit Fig. 4.41.

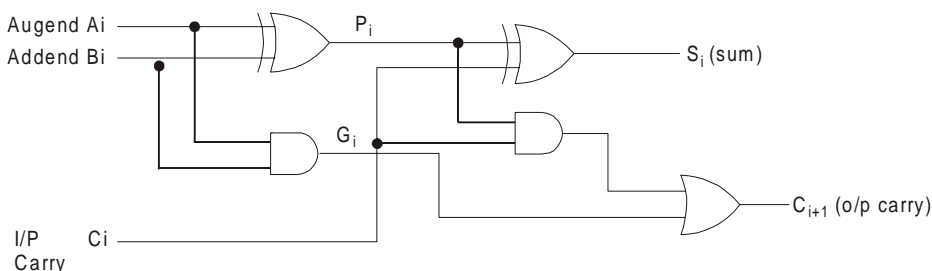


Fig. 4.41 Full Adder



We define two new intermediate output variable  $P_i$  and  $G_i$ .

$P_i = A_i \oplus B_i$  ; called carry propagate, and

$G_i = A_i \cdot B_i$  called carry generate.

We now write the Boolean function for the carry output of each stage and substitute for each  $C_i$  its value from the previous equations :

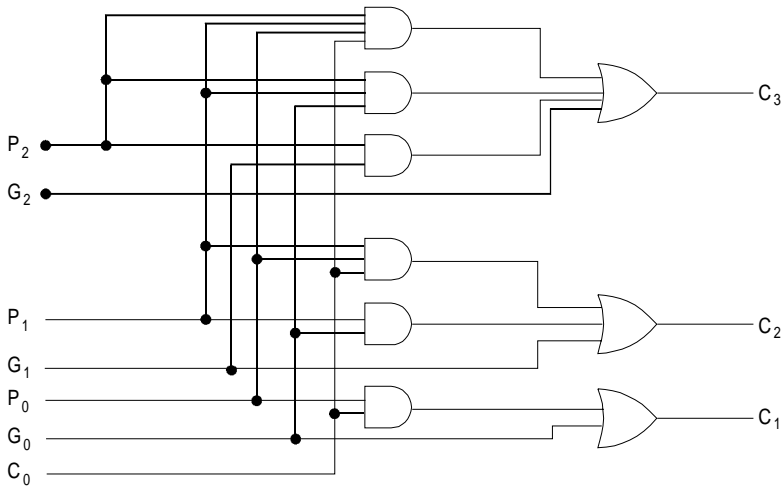
$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0.$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_0) \\ = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0.$$

Note that  $C_3$  does not have to wait for  $C_2$  and  $C_1$  to propagate; in fact  $C_3$  is propagated at the same time as  $C_1$  and  $C_2$ .

Next we draw the logic diagram of this 4 bit look-ahead carry generator as shown in Fig. 4.42.



**Fig. 4.42** 4-bit look-ahead carry generator.

**4-bit binary parallel adder with a look-ahead carry generator (FAST ADDER)**

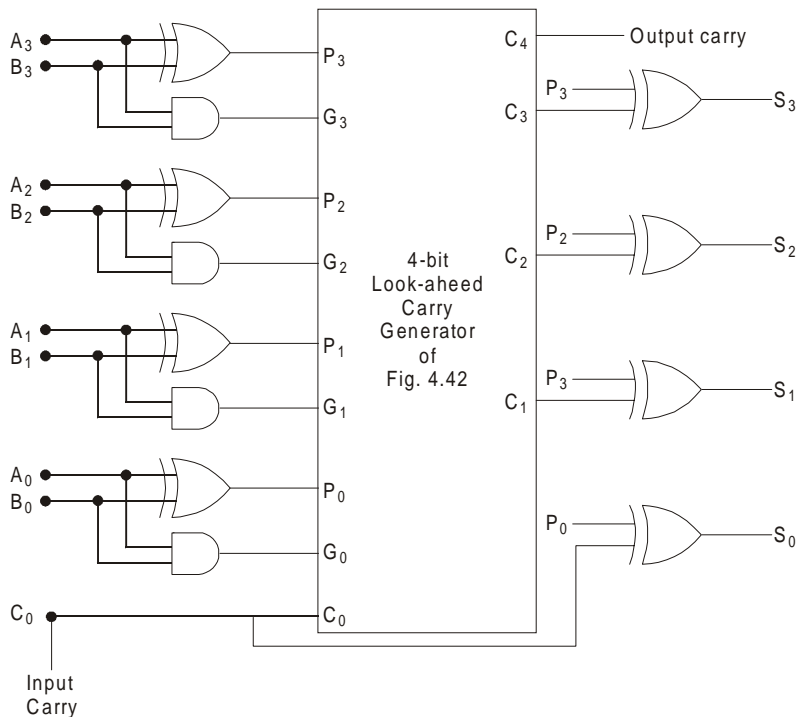
In the 4-bit look ahead carry generator. We have seen that all the carry outputs are generated simultaneously with the application of Augend word, addend word and the input carry. What is remaining are the sum outputs. From the newly defined full adder circuit of Fig. 4.41, we notice that the sum output  $S_i = P_i \oplus C_i$ .

$$\Rightarrow \begin{aligned} S_0 &= P_0 \oplus C_0 = A_0 \oplus B_0 \oplus C_0 \\ S_1 &= P_1 \oplus C_1 = A_1 \oplus B_1 \oplus C_1 \\ S_2 &= P_2 \oplus C_2 = A_2 \oplus B_2 \oplus C_2 \\ S_3 &= P_3 \oplus C_3 = A_3 \oplus B_3 \oplus C_3. \end{aligned}$$

Similarly carry output  $C_{i+1} = G_i + P_i C_i$

$$\Rightarrow \text{Final o/p carry} \Rightarrow C_4 = G_3 + P_3 C_3.$$

Using the above equations, the 4-bit binary parallel adder with a look ahead carry generator can be realized as shown in Fig. 4.43.



**Fig. 4.43** Four bit binary parallel adder with look-ahead carry generator.

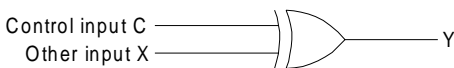
From the diagram, note that the addition of two 4 bit numbers can be done by a look ahead carry generator in a 4 gate propagation time (4 stage implementation). Also, it is important to realize that the addition of n-bit binary numbers takes the same 4-stage propagation delay.

### 4-bit Parallel Adder/Subtractor

The 4-bit binary parallel adder/subtractor can be realized with the same circuit taking into consideration the 2's complement method of subtraction and the controlled inversion property of the exclusive or gate.

The subtraction of two binary number by taking 2's complement of the subtrahend and adding to the minuend. The 2's complement of the subtrahend can be obtained by adding 1 to the 1's complement of the subtrahend.

From the example OR operation, we know when one of the input is low the output is same as the other input and when are of the input is high the output is the complement of the other input.



$$Y = C X' + C X$$

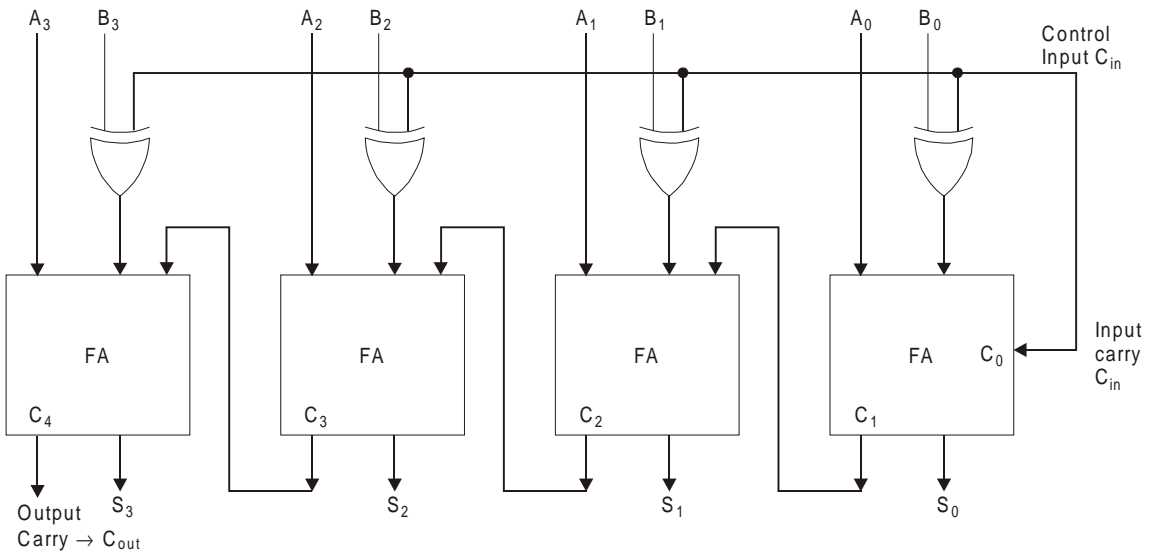
Naturally, if

$$C = 0, Y = X$$

$$C = 1, Y = X'$$

The 4-bit binary parallel adder/subtractor circuit is shown in Fig. 4.44. that perform the operation of both addition and subtraction. It has two four bit inputs  $A_3 A_2 A_1 A_0$  and  $B_3 B_2 B_1 B_0$ . The control input line  $C$ , connected with the input carry of the LSB of the full adder, is used to perform both operations.

To perform subtraction, the  $C$  (control input) is kept high. The controlled inverter produces the 1's complement of the addend ( $B_3' B_2' B_1' B_0'$ ). Since 1 is given to input carry of the LSB of the adder, it is added to the complemented addend producing 2's complement of the addend before addition.



**Fig. 4.44** Bit binary parallel adder/subtractor.

Now the augend ( $A_3 A_2 A_1 A_0$ ) will be added to the 2's complement of addend ( $B_3 B_2 B_1 B_0$ ) to produce the sum, i.e., the difference between the addend and augend, and  $C_{out}$  (output carry), i.e. the borrow output of the 4-bit subtractor.

When the control input 'C' is kept low, the controlled inverter allows the addend ( $B_3 B_2 B_1 B_0$ ) without any change to the input of full adder, and the input carry  $C_{in}$  of LSB of full adder, becomes zero. Now the augend ( $A_3 A_2 A_1 A_0$ ) and addend ( $B_3 B_2 B_1 B_0$ ) are added with  $C_{in} = 0$ . Hence, the circuit functions as 4-bit adder resulting in sum  $S_3 S_2 S_1 S_0$  and carry output  $C_{out}$ .

#### 4.2.5 Decimal Adder

A BCD adder is a combinational circuit that adds two BCD digits in parallel and produces a sum digit which is also in BCD. The block diagram for the BCD adder is shown in Fig. 4.45. This adder has two 4-bit BCD inputs  $A_8 A_4 A_2 A_1$  and  $B_8 B_4 B_2 B_1$  and a carry input  $C_{in}$ . It also has a 4-bit sum output  $S_8 S_4 S_2 S_1$  and a carry output  $C_{out}$ . Obviously the sum output is in BCD form. (This is why subscripts 8, 4, 2, 1 are used).

If we consider the arithmetic addition of two decimal digits in BCD, the sum output can not be greater than  $9 + 9 + 1 = 19$ . (Since each input digit does not exceed 9 and 1 being the possible carry from previous stage).

Suppose we apply two BCD digits to a 4-bit binary parallel adder. The adder will form the sum in binary and produce a sum that will range from 0 to 19. But if we wish to design a BCD adder, it must be able to do the following.

1. Add two 4-bit BCD numbers using straight binary addition.
2. If the four bit sum is equal to or less than 9, the sum is in proper BCD form.
3. If the four bit sum is greater than 9 or if a carry is generated from the sum, the sum is not in BCD form. In this case a correction is required that is obtained by adding the digit 6 (0110) to the sum produced by binary adder.

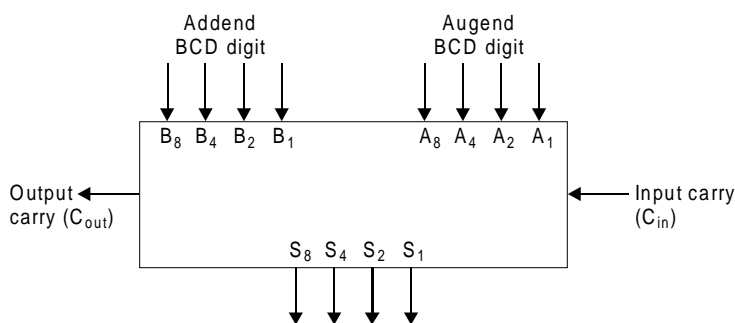


Fig. 4.45 Block diagram of a BCD adder.

The table shows the results of BCD addition with needed correction.

Decimal Digit	Uncorrected BCD sum produced by Binary Adder:					Corrected BCD sum produced by BCD Adder:					
	K	Z <sub>8</sub>	Z <sub>4</sub>	Z <sub>2</sub>	Z <sub>1</sub>	C	S <sub>8</sub>	S <sub>4</sub>	S <sub>2</sub>	S <sub>1</sub>	
0	0	0	0	0	0	0	0	0	0	0	No Correction Required
1	0	0	0	0	1	0	0	0	0	1	
2	0	0	0	1	0	0	0	0	1	0	
3	0	0	0	1	1	0	0	0	1	1	
4	0	0	1	0	0	0	0	1	0	0	
5	0	0	1	0	1	0	0	1	0	1	
6	0	0	1	1	0	0	0	1	1	0	
7	0	0	1	1	1	0	0	1	1	1	
8	0	1	0	0	0	0	1	0	0	0	
9	0	1	0	0	1	0	1	0	0	1	
10	0	1	0	1	0	1	0	0	0	0	Correction Required
11	0	1	0	1	1	1	0	0	0	1	
12	0	1	1	0	0	1	0	0	1	0	
13	0	1	1	0	1	1	0	0	1	1	
14	0	1	1	1	0	1	0	1	0	0	
15	0	1	1	1	1	1	0	1	0	1	
16	1	0	0	0	0	1	0	1	1	0	
17	1	0	0	0	1	1	0	1	1	1	
18	1	0	0	1	0	1	1	0	0	0	
19	1	0	0	1	1	1	1	0	0	1	

The binary numbers are listed, labeled as  $K Z_8 Z_4 Z_2 Z_1$ .  $K$  is the output carry and subscript under  $Z$  represent the weight 8, 4, 2 and 1. The first table lists the binary sums as produced by a 4-bit binary adder. For representing them in BCD, they must appear as second table.

From the two tables it is clear that upto 9, the binary sum is same as the BCD sum, so no correction is required. When the sum is greater than 9, the binary sum is different from BCD sum, means correction is required. Therefore, a BCD adder must include the correction logic in its internal construction. Moreover, the circuit is required to develop a logic in its internal construction that indicates for needed correction.

This later logic can be developed by observing the two table entries. From tables it is clear that the correction is required when  $K = 1$  or  $Z_8 Z_4 = 1$  or  $Z_8 Z_2 = 1$ .

or when  $k = 1$  or  $Z_8 (Z_4 + Z_2) = 1$ .

$K = 1$ , means the result is 16 or above,

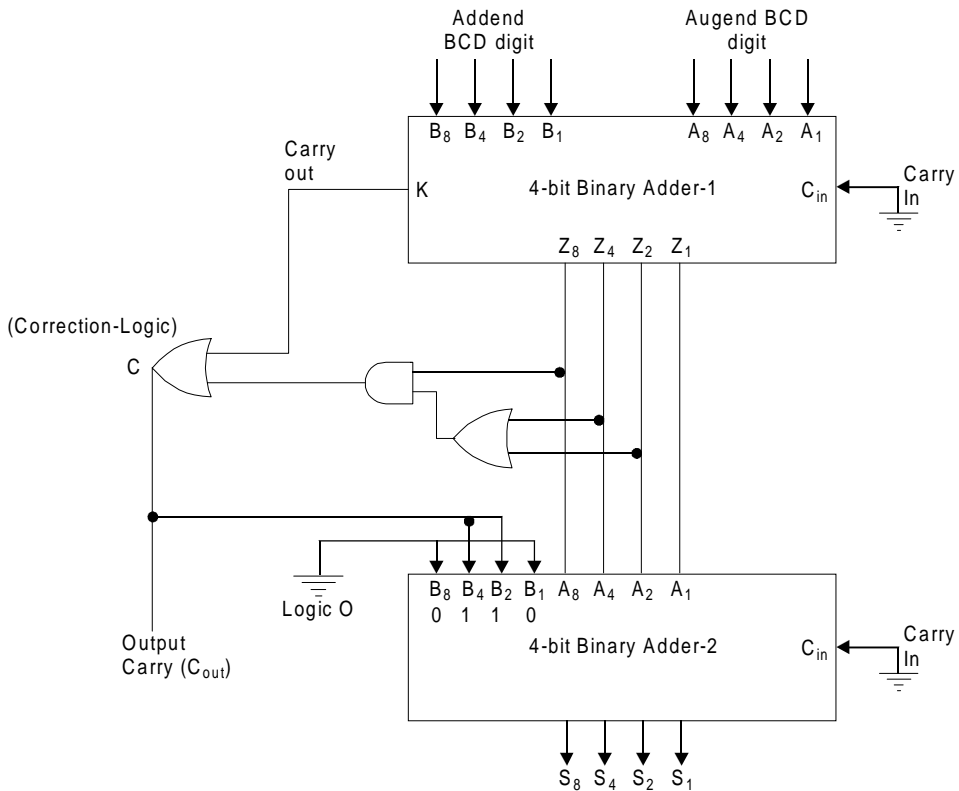
$Z_8 Z_4 = 1$ , means the result is 12 or above and

$Z_8 Z_2 = 1$ , means the result is 10 or above.

Therefore, the condition for correction can be written as :

$$C = K + Z_8 (Z_4 + Z_2)$$

*i.e.*, whenever  $C = 1$ , we need correction  $\Rightarrow$  Add binary 0110 (decimal 6) to the sum produced by 4 bit binary adder. It also produce an output carry for the next stage. The BCD adder can be implemented using two 4-bit binary parallel adders as shown in Fig. 4.46.



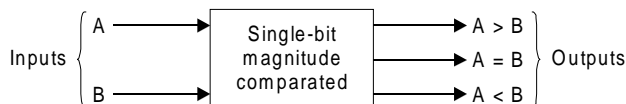
**Fig. 4.46** BCD Adder using two 4-bit binary adders along with the correction logic C.

Here  $A_8 A_4 A_2 A_1$  and  $B_8 B_4 B_2 B_1$  are the BCD inputs. The two BCD inputs with input carry  $C_{in}$  are first added in the 4-bit binary adder-1 to produce the binary sum  $Z_8, Z_4, Z_2, Z_1$  and output carry  $K$ . The outputs of adder-1 are checked to ascertain whether the output is greater than 9 by AND-OR logic circuitry. If correction is required, then a 0110 is added with the output of adder-1. Now the 4-bit binary adder-2 forms the BCD result ( $S_8 S_4 S_2 S_1$ ) with carry out  $C$ . The output carry generated from binary adder-2 can be ignored, since it supplies information already available at output carry terminal  $C$ .

### 4.2.6. Magnitude Comparator

A magnitude comparator is a combinational circuit designed primarily to compare the relative magnitude of the two binary numbers  $A$  and  $B$ . Naturally, the result of this comparison is specified by three binary variables that indicate, whether  $A > B$ ,  $A = B$  or  $A < B$ .

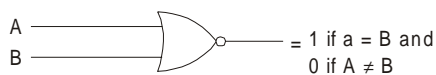
The block diagram of a single bit magnitude comparator is shown in Fig. 4.47.



**Fig. 4.47** Block diagram of single bit magnitude comparator.

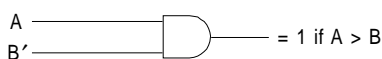
To implement the magnitude comparator the properties of Ex-NOR gate and AND gate is used.

Fig. 4.48(a) shows an EX-NOR gate with two inputs  $A$  and  $B$ . If  $A = B$  then the output of Ex-NOR gate is equal to 1 otherwise 0.

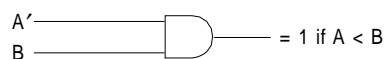


**Fig. 4.48 (a)**

Fig. 4.48 (b) and (c) shows AND gates, one with  $A$  and  $B'$  as inputs and another with  $A'$  and  $B$  as their inputs.



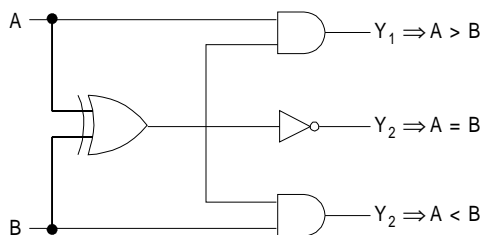
**Fig. 48 (b)**



**Fig. 4.48 (c)**

The AND gate output of 4.48(b) is 1 if  $A > B$  (i.e.  $A = 1$  and  $B = 0$ ) and 0 if  $A < B$  (i.e.  $A = 0$  and  $B = 1$ ). Similarly the AND gate output of 4.48(c) is 1 if  $A < B$  (i.e.  $A = 0$  and  $B = 1$ ) and 0 if  $A > B$  (i.e.  $A = 1$  and  $B = 0$ ).

If the EX-NOR gate and two AND gates are combined as shown in Fig. 4.49(a), the circuit with function as single bit magnitude comparator. For EX-NOR implementation.



**Fig. 4.49 (a)** Single bit magnitude comparator.

We have used EX-OR followed by an inverter.

Truth table of a single bit magnitude comparator.

Inputs		Output		
A	B	$Y_1$	$Y_2$	$Y_3$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

It clearly shows  $Y_1$  is high when  $A > B$ .

$Y_2$  is high when  $A = B$

$Y_3$  is high when  $A < B$ .

The same principle can be extended to an n-bit magnitude comparator.

#### 4-bit Magnitude Comparator

Consider two numbers A and B, with four digits each.

$$A = A_3 A_2 A_1 A_0$$

$$B = B_3 B_2 B_1 B_0$$

- (a) The two numbers are equal if all pairs of significant digits are equal i.e. if  $A_3 = B_3$ ,  $A_2 = B_2$ ,  $A_1 = B_1$  and  $A_0 = B_0$ . We have seen that equality relation is generated by EX-NOR gate. Thus

$$x_i = A_i \cdot B_i = A_i B_i + A_i' B_i', \quad i = 0, 1, 2, 3.$$

Where  $x_i$  represents the equality of two numbers

$$x_i = 1, \text{ if } A = B.$$

$$x_i = 0, \text{ otherwise.}$$

It follows an AND operation of all variables.

$$\Rightarrow (A = B) = x_3 x_2 x_1 x_0 = 1 \text{ only if all pairs are equal.}$$

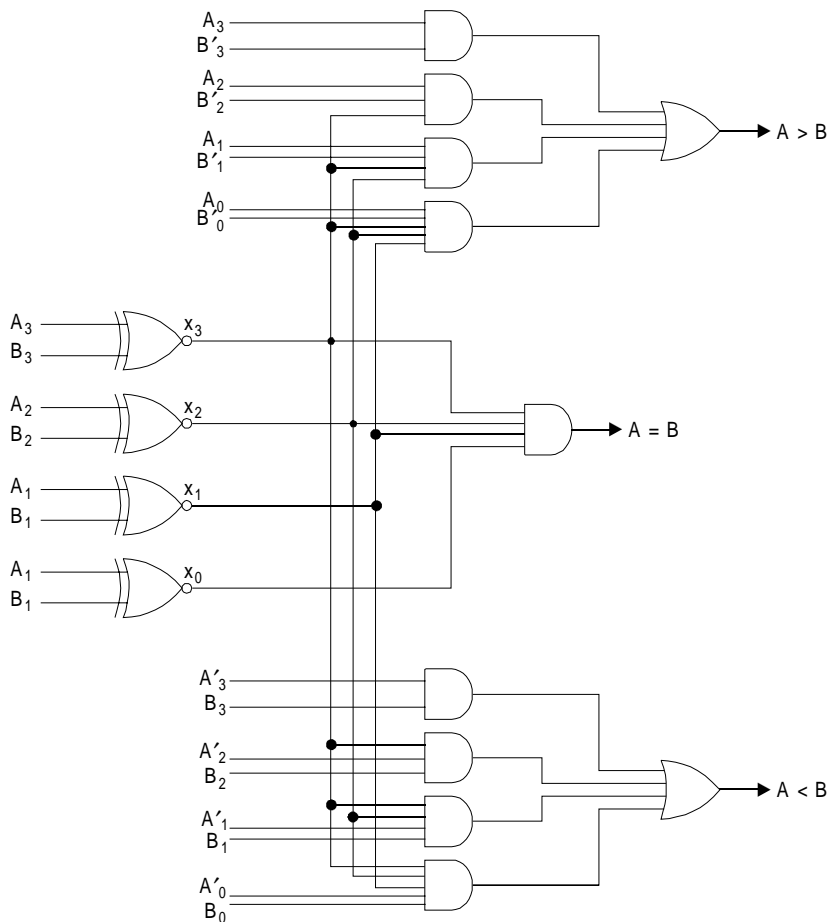
- (b) To determine if  $A > B$  or  $A < B$ , we check the relative magnitude of pairs of significant digits starting from MSB. If the two digits are equal, we compare the next lower significant pair of digits. The comparison follows until a pair of unequal digits are reached. If the corresponding digit of A is 1 and that of B is 0, we say that  $A > B$ . If the corresponding digit A is 0 and that of B is 1  $\Rightarrow A < B$ .

This discussion can be expressed logically as :

$$(A > B) = A_3 B_3' + x_3 A_2 B_2' + x_3 x_2 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'$$

$$(A < B) = A_3' B_3 + x_3 A_2' B_2 + x_3 x_2 A_1' B_1 + x_3 x_2 x_1 A_0' B_0.$$

The logical implementation is shown in Fig. 4.49(b)



**Fig. 4.49(b)** Logical implementation of or 4-bit magnitude comparator.

### 4.3 HAZARDS

In digital circuits it is important that undesirable glitches (spikes) on signal should not occur. Therefore in circuit design one must be aware of the possible sources of glitches (spikes) and ensure that the transitions in a circuit will be glitch free. The glitches (spikes) caused by the structure of a given circuit and propagation delays in the circuit are referred to as *hazards*. Hazards occur in combinational circuits, where they may cause a temporary false-output value.

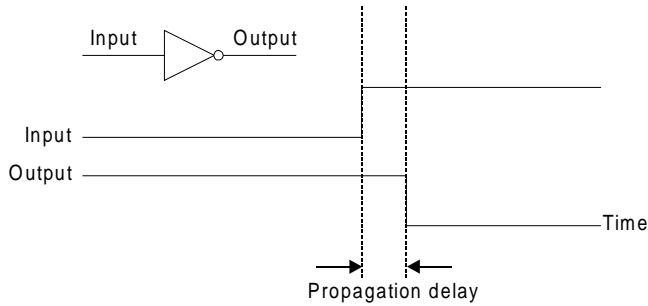
#### 4.3.1 Hazards in Combinational Circuits

Hazards is **unwanted** switching transients appearing in the output while the input to a combinational circuit (network) changes. The reason of hazard is that the different paths from input to output have different propagation delays, since there is a finite propagation delay through all gates. Fig. 4.50 depicts the propagation delay in NOT gate.

In the circuit analysis, dynamic behaviour is an important consideration. The propagation delay of circuit varies and depends upon two factors.

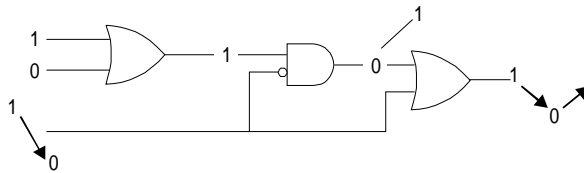
- Path of change through circuit.
- Direction of change within gates.





**Fig. 4.50**

The glitches (spikes) are momentary change in output signal and are property of circuit, not function as depicted in Fig. 4.51.



**Fig. 4.51**

Hazards/Glitches (spikes) are dangerous if

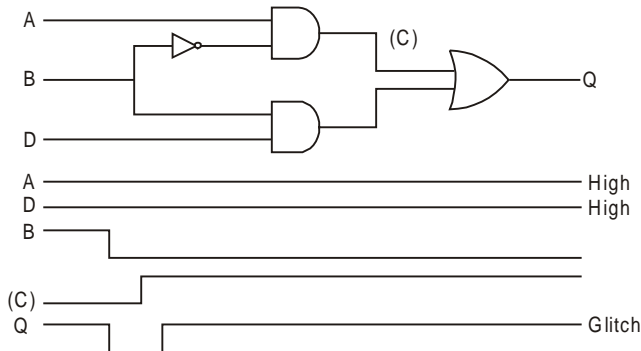
- Output sampled before signal stabilizes
- Output feeds asynchronous input (immediate response)

The usual solutions are :

- Use synchronous circuits with clocks of sufficient length
- Minimize use of circuits with asynchronous inputs
- Design hazard free circuits.

**Example.** Show that the combinational circuit  $Q = \overline{A}B + BD$  having hazards.

**Solution.** For  $Q = \overline{A}B + BD$ ; if B and D are 1 then Q should be 1 but because of propagation delays, if B changes stage then Q will become unstable for a short time, as follows :



**Fig. 4.52** Therefore the given combinational circuit having hazards.

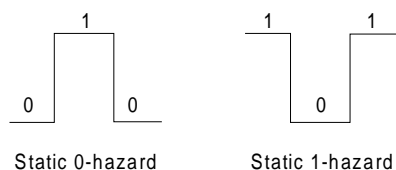
### 4.3.2 Types of Hazards

Two types of hazards are illustrated in Fig. 4.53.

- Static hazard
- Dynamic hazard.

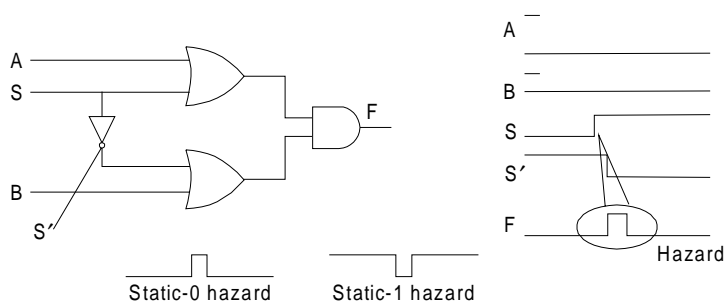
#### 1. Static 1 (0) hazard

A static hazard exists if, in response to an output change and for some combination of propagation delays, a network output may momentarily go to 0 (1) when it should remain a constant 1 (0). We say the network has a static 1 (0) hazard. Example of static hazard is shown in Fig. 4.54 (a).



**Fig. 4.53**

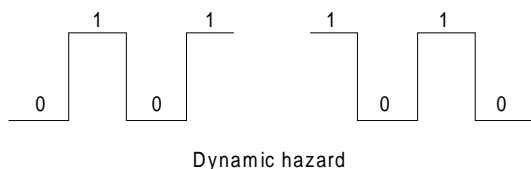
**Example.**



**Fig. 4.54 (a)**

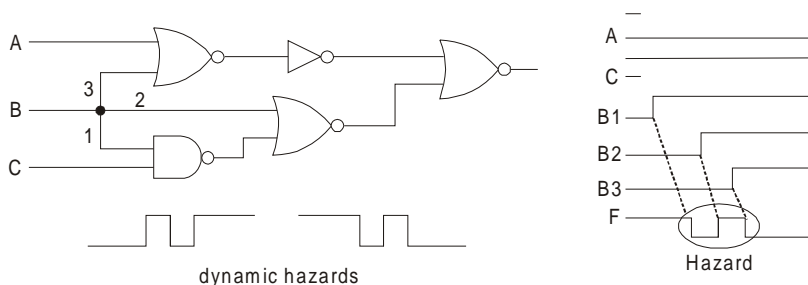
#### 2. Dynamic Hazard

A different type of hazard may occur if, when the output is suppose to change from 0 to 1 (or 1 to 0), the output may change three or more times, we say the network has a dynamic hazard. Example of dynamic hazard is shown in Fig. 4.54 (b).



**Fig. 4.54 (b)**

**Example:**



**Fig. 4.54 (b)**

### 4.3.3 Hazard Free Realizations

The occurrence of the hazard can be detected by inspecting the Karnaugh Map of the required function. Hazards like example (Fig. 4.52) are best eliminated logically. The Fig. 4.52 is redrawn here (Fig. 4.55).

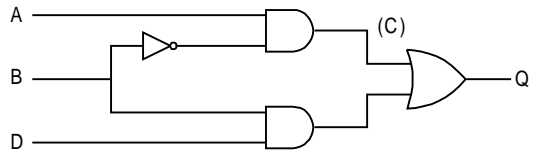


Fig. 4.55

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

The Karnaugh Map of the required function is given in Fig. 4.56.

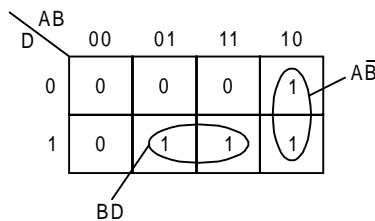


Fig. 4.56 K-Map of the given circuit.

Whenever the circuit move from one product term to another there is a possibility of momentary interval when neither term is equal to 1, giving rise to an undesirable output. The remedy for eliminating hazard is to enclose the two minterms with another product term that overlaps both groupings.

The covering the hazard causing the transition with a redundant product term (AD) will eliminate the hazard. The K-Map of the hazard-free circuit will be as shown in Fig. 4.57.

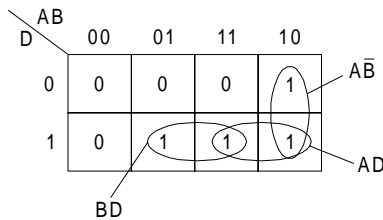


Fig. 4.57 K-Map of the hazard-free circuit.

Therefore the hazard free Boolean equation is  $Q = A\bar{B} + BD + AD$ .

The Fig. 4.58 shows the hazard free realization of circuit shown in Fig. 4.55.

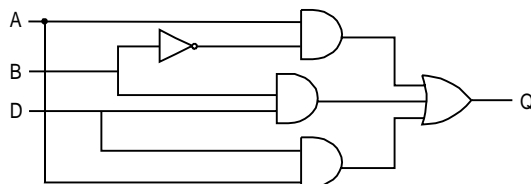


Fig. 4.58 Hazard free circuit.

Now we will discuss elimination of static hazards with examples.

#### Eliminating a static-1 hazard

Let the example circuit is  $F = A\bar{C} + \bar{A}D$ . The K-map of the circuit is given in Fig. 4.59.

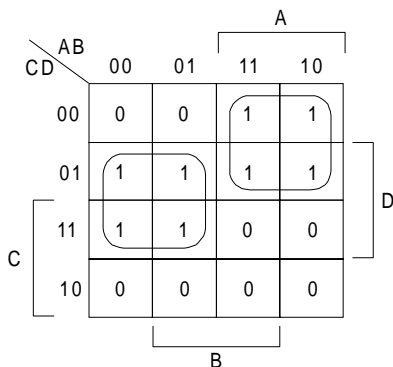
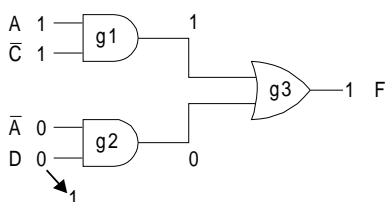


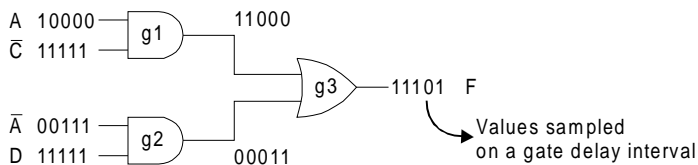
Fig. 4.59 K-Map of the example circuit.

By inspecting Karnaugh Map of the required function, we notice the following points.

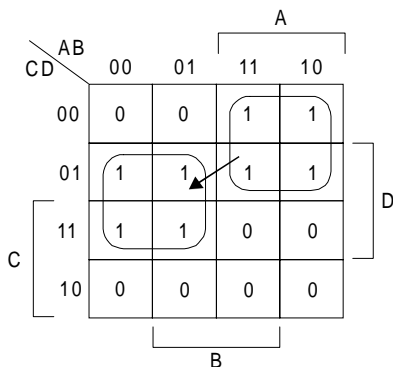
- Input change within product term (ABCD = 1100 to 1101)



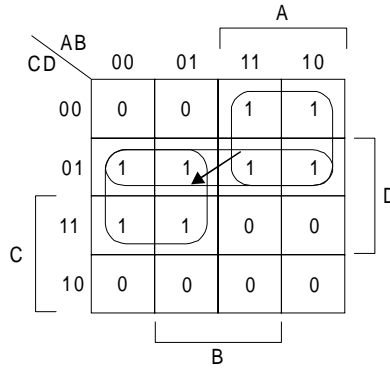
- Input change that spans product terms (ABCD = 1101 to 0101)



- Glitch only possible when move between product terms.



From above three points it is clear that addition of redundant prime implicants so that all movements between adjacent on-squares remains in a prime implicant will remove hazard.



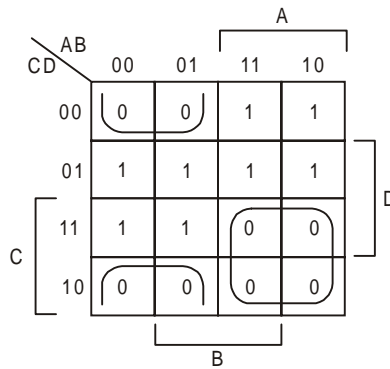
**Fig. 4.60** K-Map of the hazards free circuit.

Therefore  $F = A\bar{C} + \bar{A}D$  becomes  $F = A\bar{C} + \bar{A}D + \bar{C}D$ .

Note that when a circuit is implemented in sum of products with AND-OR gates or with NAND gates, the removal of static-1 hazard guarantees that no static-0 hazards or dynamic hazards will occur.

**Eliminating a static-0 hazard**

Let the example circuit is  $F = (\bar{A} + \bar{C})(A + D)$ . The K-Map of the circuit is given in Fig. 4.61.



**Fig. 4.61** K-Map of the example circuit.

By inspecting Karnaugh Map of the required function, we see that occurrence of static-0 hazard from  $ABCD = 1\ 10$  to  $0110$ . It can be remove by adding the term  $(\bar{C} + D)$ .

**4.3.4 Essential Hazard**

Similar to static and dynamic hazards in combinational circuits, essential hazards occur in sequential circuits. Essential hazards is a type of hazard that exists only in asynchronous sequential circuits with two or more feedbacks. Essential hazard occurs normally in toggling type circuits. It is an error generally caused by an excessive delay to a feedback variable in response to an input change, leading to a transition to an improper state. For example, an excessive delay through an inverter circuit in comparison to the delay associated with the feedback path may cause essential hazard. Such hazards cannot be eliminated by adding reducant gates as in static hazards. To avoid essential hazard, 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.

#### 4.3.5 Significance of Hazards

A glitch in an asynchronous sequential circuit can cause the circuit to enter an incorrect stable state. Therefore, the circuitry that generates the next-state variables must be hazard free. It is sufficient to eliminate hazards due to changes in the value of a single variable because the basic premise in an asynchronous sequential circuit is that the values of both the primary inputs and the state variables must change one at a time.

In synchronous sequential circuits the input signal must be stable within the setup and hold times of flip-flops. It does not matter whether glitches (spikes) occur outside the setup and hold times with respect to the clock signal.

In combinational circuits, there is no effect of hazards, because the output of a circuit depends solely on the values of the inputs.

#### 4.4 FAULT DETECTION AND LOCATION

Digital system may suffer two classes of faults : temporary faults, which occur due to noise and the non-ideal transient behaviour of switching components and second is permanent faults. which result from component failures. The transient behaviour of switching is studied in hazards. while permanent faults testing and diagnosis consists of the following two subproblems.

1. The fault detection problem.
2. The fault-location problem.

There are various methods used for the fault detection and location these are :

1. Classical method.
2. Fault table method.
3. Boolean differences method.
4. Path sensitizing method.

##### 4.4.1 Classical Method

The process of applying tests and determining whether a digital circuit is fault free or not is generally known as fault detection.

One way of determining whether a combinational circuit operates properly is by applying to the circuit all possible input combinations and comparing the resultant outputs with either the corresponding truth table or a faultless version of the same circuit. Any deviation indicates the presence of some fault. If a known relationship exists between the various possible faults and the deviations of output pattern then we can easily diagnose the fault and classify it within a subset of faults whose effects on the circuit output are identical.

This classical method is generally very long, and consequently impractical. Moreover, for most circuits the fault can be detected or even, to locate them by considerably shorter tests, whereas by classical method there are large number of tests have to be done which are unnecessary. Such tests are referred to as either Fault detection or Fault-location tests. These tests are used for detecting the presence of a fault, locating them and diagnosing them.

#### Assumptions about the type of digital circuit to be considered

1. It is assumed that the digital circuits under study are combinational circuits, testing procedures are developed for circuits composed of loop-free interconnection of

AND, OR NOT, NAND, and NOR gates ; that is feedback loops are not allowed in the circuits being tested.

2. It is assumed that the response delays of all the gates element are the same.
3. Since single fault will be detected in all cases, while multiple fault may not be detected in digital circuit so it is assumed that most circuits are reliable enough so that the probability of occurrence of multiple faults in rather small.

**Assumption about type of Fault to be considered**

1. The faults considered here are assumed to be fixed or permanent or non transient faults by which mean that without having them repaired, the fault will be permanently there.
2. Most of the faults occurred in currently used circuits such as RTL, DTL, TTL, are those which cause a wire to be (or appear logically to be) stuck at zero or stuck-at-one (abbreviated s-a-0 and s-a-1). Restricting our consideration to just this class of faults is technically justified, since most circuit failures fall in this class, and many other failures exhibit symptomatically identical effect.
3. A multiple fault is defined as the simultaneous occurrence of any possible combination of s-a-0 and s-a-1 faults.

**4.4.2 The Fault Table Method**

The most classic method for constructing a set of tests to detect and locate a prescribed list of permanent logic faults, single or multiple, in a combinational circuit is the fault table method.

“A Fault table is a table in which there is a row for every possible test (i.e. input combination) and a column for every fault. A 1 is entered at the intersection of the ‘ith’ row and the ‘jth’ column if the fault corresponding to the ‘jth’ column can be detected by the ith test.”

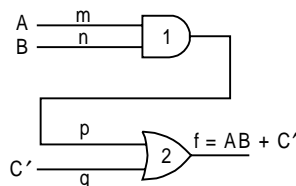
Before starting with fault table method we should know detectable and undetectable faults and a minimal complete test set of circuit.

A Fault of a combinational circuit is said to be detectable if there exists a test by which we can judge whether or not the circuit has such a fault, otherwise, we call the fault undetectable.

A test set of a circuit is said to be complete if it detects every fault of circuit under consideration. A minimal complete test set of circuit is complete test set that contains a minimum number of tests. Let on consider ‘n’ inputs to the circuit, then there are  $2^n$  number of rows. Thus a subset of  $2^n$  rows constitute a complete test set for the circuit.

**The Fault table**

Let  $x_1, x_2, \dots, x_n$  be inputs to a combinational circuit and let ‘f’ be its fault free output. Let ‘f’ be its fault free output. Let  $f_\alpha$  denote the output of the circuit in the presence of fault  $\alpha$ . consider an example, the circuit shown belows in Fig. 4.62.



**Fig. 4.62** Circuit under test.

**Circuit to be tested**

Suppose that any one of its wires  $m, n, p,$  and  $q$  may have  $s-a-0$  or  $s-a-1$  Fault. We shall denote  $s-a-0$  and  $s-a-1$  faults for wire ' $m$ ' as  $m_0$  and  $m_1$  respectively. Similar notation is used for the other wires. The truth table for this circuit is shown below, where column  $f$  denotes the fault-free output for example, columns  $fm_0$  and  $fm_1$  denotes the circuit outputs in the presence of faults  $m_0$  and  $m_1$  on wire  $m$ , and so on.

Inputs ABC	$f = (mn + q)$	Outputs in presence of Faults							
	$= (p + q)$ $= AB + C'$	$fm_0$ ( $m = 0$ )	$fn_0$ ( $n = 0$ )	$fp_0$ ( $p = 0$ )	$fq_0$ ( $q = 0$ )	$fm_1$ ( $m = 1$ )	$fn_1$ ( $n = 1$ )	$fp_1$ ( $p = 0$ )	$fq_1$ ( $q = 1$ )
0 0 0	1	1	1	1	0	1	1	1	1
0 0 1	0	0	0	0	0	0	0	1	1
0 1 0	1	1	1	1	0	1	1	1	1
0 1 1	0	0	0	0	0	1	0	1	1
1 0 0	1	1	1	1	0	1	1	1	1
1 0 1	0	0	0	0	0	0	1	1	1
1 1 0	1	1	1	1	1	1	1	1	1
1 1 1	1	0	0	0	1	1	1	1	1

(Truth table for the fault-free output and output in presences of faults)

An input combination is referred to as a test for fault  $f_\alpha$  if, in response to that input combination, the output of the correctly operating circuit is different from that of the circuit impaired by fault ' $f_\alpha$ '. If we have observed the ' $fm_0$ ' column, there is only one fault for input combination 1 1 1, for other input combination there is no fault since  $f = fm_0$  (for input combination 0 0 0, 0 0 1..... 1 1 0) where as for input combination 1 1 1,  $f \neq fm_0$ . On the other hand, Fault  $f_{q1}$  can be detected by the tests 0 0 1, 0 1 1, and 1 0 1, and so on, for other faults. More precisely, an input combination  $a_1, a_2, \dots, a_n$  is a test for detecting fault  $f_\alpha$  if and only if.

$$f(a_1, a_2, \dots, a_n) \oplus f_\alpha(a_1, a_2, \dots, a_n) = 1$$

where  $f(a_1, a_2, \dots, a_n)$  and  $f_\alpha(a_1, a_2, \dots, a_n)$  denote, respectively, the fault-free output and the incorrect output in response to the input  $a_1, a_2, \dots, a_n$ .

From the fault table, it is clear that column ( $m_1, m_0$ ) and ( $p_1, q_1$ ) are identical.

Since these four groups are identical. Thus these Faults are indistinguishable faults of the fault table. we combine them, choosing one function from each group, and delete rest of them, In this way we can get fault table from truth table as:

**Fault table for minimal set of fault detection test**

Inputs ABC	Possible Faults				
	$\{m_0, n_0, p_0\}$ $(f \oplus f_1)$	$q_0$ $(f \oplus f_2)$	$m_1$ $(f \oplus f_3)$	$n_1$ $(f \oplus f_4)$	$\{p_1, q_1\}$ $(f \oplus f_5)$
0 0 0		1			
0 0 1					1
0 1 0		1			
0 1 1		1			1

(Contd.)



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

(where  $f_1 = f_{m0} = f_{r0} = f_{p0}$ ,  $f_2 = f_{q0}$ ,  $f_3 = f_{m1}$ ,  $f_4 = f_{n1}$  and  $f_5 = f_{p1} = f_{q1}$  and  $f$  is fault free output of the circuit).

### Equivalent Faults

As we have seen that columns  $f_{m0}$ ,  $f_{r0}$ , and  $f_{p0}$  are identical, and so are columns  $f_{p1}$ , and  $f_{q1}$ . In other words, the circuit output in the presence of fault  $f_{p1}$  is identical with the output in the presence of  $f_{q1}$ . Hence is no input combination which can distinguish fault  $f_{p1}$  from  $f_{q1}$ . Such faults are called *equivalent faults*. Equivalent faults are indistinguishable whereas faults that are not equivalent are said to be distinguishable faults. The problem of finding a minimal set of tests is now reduced to the problem of finding a minimal set of rows so that every columns has a 1 entry in a least one row of the set. Such a set of rows will be said to *cover the fault table*.

### Covering the fault table for determination of a minimal set of fault detection

The minimal test set found from the fault table with the help of following two rules:

1. Delete any row that is covered by, or is same as, some other row.
2. Delete any column that covers, or is same as, some other column.

We can easily verify that rows 010 and 100 can be removed because they are equivalent to 000 similarly row 001 can be removed since it is dominated by row 101. Also row 110 does not detect any fault and can be removed from the table.

From the fault table. It is seen that test 011, 101 and 111 covers column  $(m_0, n_0, p_0)$ ,  $m_1, n_1, (p_1, q_1)$  except  $q_0$  are *essential test*. The essential test are determined by observing which faults can be detected by just a single test. A fourth test 000 detects  $q_0$  and thus completes the set of tests that detect all single faults in the circuit being tested.

It is convenient to distinguish between a fault-detection test, which refers to a single application of values to the input terminals, and a fault-detection experiment, which refers to a set of tests leading to a definite conclusion as to whether or not the circuit operates correctly for all input combination. In above example the experiment consists of four test as follows (through not necessarily in this order):

1. Apply 011: if the output is T = 1, circuit faulty and the experiment may be stopped; if T = 0, apply second test.
2. Apply 101: if T = 1, circuit faulty; otherwise, apply next test.
3. Apply 111: if T = 0, circuit is faulty; otherwise, apply next test.
4. Apply 000: if T = 0, circuit is faulty, if T = 1, circuit operates correctly.

The fault table provides a tool for the determination of a minimal set of fault detection test for combinational logic circuits. But for larger circuits this method become cumbersome due to large size of fault table, computation time and memory requirements.

For last example the fault detection table becomes in reduced form as:

Test	Faults				
	$\{m_0, n_0, p_0\}$	$q_0$	$m_1$	$n_1$	$\{p_1, q_1\}$
0 0 0	1				
0 1 1	(1)				
1 0 1	(1)				
1 1 1	(1)				

Thus the minimal complete test set for detections of all single faults of this circuit is  $\{0, 3, 5, 7\}$ .

### 4.4.3 Fault detection by Path Sensitizing

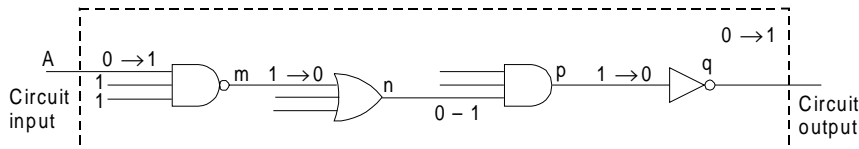
Here we shall show that a fault-detection test may be found by examining the paths of transmission from the location of an assumed fault to one of its primary output.

⇒ In a logic circuit, a *primary output* is a line whose signal output is accessible to the exterior of the circuit, and primary input is a line that is not fed by any other line in the circuit.

⇒ A transmission path, or simply path of a combinational circuit, is a connected directed graph containing no loops from a primary input to one of its primary output.

**Path Sensitizing:** A path is said to be *sensitized* if the input to the gates along this path are assigned value so as to propagate any change on the faulty line along the chosen path to the output terminal of the path.

The main idea behind the path-sensitizing procedure will be illustrated by a circuit which detects a *s-a-1* fault at its one input as shown in Fig. 4.63.



**Fig. 4.63** A circuit describing a sensitized path.

In this circuit it is assumed that this path is the only from A to the circuit output and a test is done which detects a *s-a-1* fault at input A. In order to test a *s-a-1* fault at A, if is necessary to apply a 0 to A and 1's to all remaining inputs to the AND and NAND gates, and 0's to all remaining inputs to the OR and NOR gates in the path. This ensure that all the gates will allow the propagation of the signal from A to the circuit output and that only this signal will reach the circuit output. The path is now said to be sensitized.

In Fig. 4.63 if input A is *s-a-1*, then m changes from 1 to 0, and this changes propagates through connections n and p and causes q to change from 0 to 1. Clearly, it detects a *s-a-1* fault at A also it detects *s-a-0* faults at m, n and p and *s-a-1* fault at q. A *s-a-0* fault at A is detected in similar manner. So *s-a-0* and *s-a-1* faults can be detected in this path.

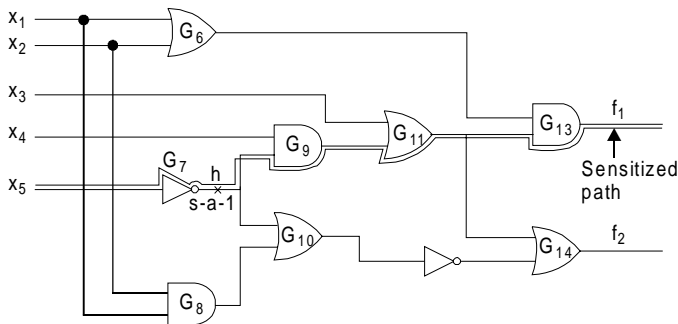
The basic principles of the path-sensitization method, which is also known as the one-dimensional path sensitization, can thus be summarized as follows:

- (1) At the site of the fault assign a logical value complementary to the fault being tested, *i.e.*, to test  $x_i$  for *s-a-1* assign  $x_i = 1$  and to test it for *s-a-0* assign  $x_i = 0$ .

- (2) Select a path from the faulty line to one of its primary output. The path is said to be sensitized if the inputs to the gates along the path are assigned values so as to propagate to the path output any faults on the wires along the path. The process is called the forward drive phase of the method.
- (3) Along the chosen path, except the lines of the path, assign a value "0" to each input of the OR and NOR gates in the path and a value 1's to each input to the NAND and AND gates in the path.
- (4) To determine the primary inputs that will produce all the necessary signals values specified in the preceding steps. This is accomplished by tracing the signals backward from each of the gates along the path to the primary inputs or circuit inputs. This process is called the backward-trace phase of the method. If on the other hand, a contradiction is encountered, choose another path which starts at faulty line and repeat the above procedure.
- (5) Apply steps 1-4 to all the single paths. If none of them is sensitizable, apply the procedure to all possible groups of three path, and so on.

These all procedure steps can be understood easily by an example

**Example.** Derive a test for a s-a-1 fault on wire h of Fig. 4.64 given below:



**Fig. 4.64**

The forward derive starts by assigning a 0 to  $h$ , and selecting a path to be sensitized. We choose to sensitize the path  $G_7 G_9 G_{11} G_{13}$  to output  $f_1$ . Clearly  $G_9$  and  $G_{13}$  are AND gates so that their other inputs must be 1, while second input to the OR gates  $G_{11}$  must be 0. This completes the forward-derive phase and path is next sensitized. Next, we must determine the primary inputs which must provide the required logical value to the gates along the path. This is accomplished by tracing back from each gate input to the corresponding primary inputs. In this example, it is quite straight forward that

$$x_5 = 1$$

$$x_4 = 1$$

$$x_3 = 0$$

and  $x_2 = 1$

We get  $G_6 = 1$  we may set-1 any one or both inputs to  $G_6$

Suppose  $x_1 = 1$

$$x_2 = 0$$

then set  $X = \{1, 0, 0, 1, 1\}$

If, in response to these inputs circuit produces the output  $f_1 = 0$ ; then fault in question does not exist.

If, on the other hand,  $f_1 = 1$ , the circuit has a fault.

Thus an erroneous outputs implies that some fault exist along the sensitized path.

### Limitations of the method

Now suppose that we observe the circuit response to the above set at output  $f_2$ .

An analysis shows that the input vector  $X = \{1, 0, 0, 1, 1\}$  sensitizes the two paths, these are

$$\left. \begin{array}{l} G_7 \ G_9 \ G_{11} \ G_{14} \\ \text{and} \\ G_7 \ G_{10} \ G_{12} \ G_{14} \end{array} \right\} \text{ which emanate from } h \text{ and terminate at } f_2.$$

It is easy, however, to verify that the fault  $h$  *s-a-1* does not propagate to  $f_2$  because there are two sensitized paths connecting it to the circuit output.

In fact  $f_2 = 1$  via  $G_{10}$  and  $G_{12}$  when  $h = 0$

and  $f_2 = 1$  via  $G_9$  and  $G_{11}$  if  $h = 1$

This does not imply that  $h$  cannot be tested via  $f_2$ . It does imply, however, that, in order to test  $h$  for *s-a-1* fault via  $f_2$ , a test must be found such that only one of the two paths will the other path will not be sensitized. One such test that sensitizen only the path  $G_7 \ G_{10} \ G_{12} \ G_{14}$  is  $X \{0, 0, 0, 0, 1\}$ .

### Advantage of the path-sensitization method

A major advantage of the path sensitization method is shown in Fig. 4.65:

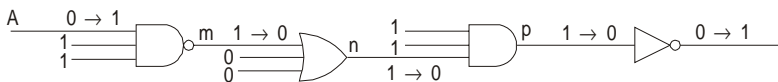


Fig. 4.65 *s-a-1* fault at A.

In many case a test for a primary input is also a test for all the wires along the sensitized path which connects inputs to the circuit output. Consequently if we can select a set of test which sensitizes a set of path containing all connections in the circuit, then it is sufficient to detect just those faults which appear on the circuit input. In fanout free circuits in which each gate output is connected to just one gate input, there is only one path from each circuit input to the output, and thus the set of paths originating at the inputs will indeed contain all connections. In circuits which contain reconvergent fanout, the problem become more complicated. In fact single-path-sensitization method does not always generates a test even if one is known to exist. So single path sensitization method can detect fault for a class of combinational circuits in which:

⇒ Each input is an independent input line to the circuits.

⇒ The fanout of every gate is '1'.

Now lets shows that single-path sensitization method does not always generates a test even if one is known to exist.

Consider the fault  $h$  *s-a-0* in the circuit shown in Fig. 4.66.

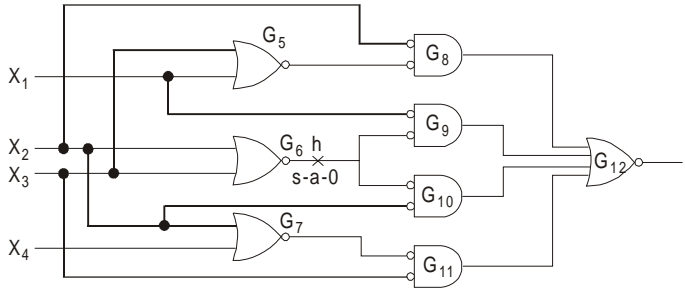


Fig. 4.66

We shall now show that it is impossible to find a test for this fault by sensitizing just a single path. Let us choose to sensitize the path  $G_6 \rightarrow G_9 \rightarrow G_{12}$ . This requires:

$G_6 = 1$ , which implies  $x_2 = 0$  and  $x_3 = 0$ .

$G_{10} = 0$  regardless of whether there is a fault or not, which implies  $x_4 = 1$ .

$G_{11} = 0$  implies  $G_7 = 1$  (since  $x_3 = 0$ ), which in turn implies  $x_4 = 0$ .

Evidently, to satisfy both  $G_{10} = 0$  and  $G_{11} = 0$ , we must set conflicting requirement on  $x_4$  and thus we have a contradiction. By the symmetry of the circuit it is obvious that an attempt to sensitize the path through  $G_{10}$  will also fail, and hence the method of one dimensional path sensitizing fails to generate the test  $x = (0, 0, 0, 0)$  which has been shown to detect this fault.

The one-dimensional path-sensitizing method failed in this case because it allows the sensitization of just a single path from the site of the fault. In fact, if we were allowed to sensitize both paths, through  $G_9$  and  $G_{10}$ , simultaneously the above test would be generated. This development of the two-dimensional path sensitizing which is also known as the  $d$ -algorithm. The basic idea in the  $d$ -algorithm is to sensitize simultaneously all possible paths from the site of the fault to the circuit outputs.

#### 4.5 EXERCISES

**Problem 1:** Develop a minimized Boolean implementation of a “ones count” circuit that works as follows. The subsystem has four binary inputs A, B, C, D and generates a 3-bit output, XYZ, XYZ is 000 if none of the inputs are 1, 001 if one input is 1, 010 if two are one, 011 if three inputs are 1, and 100 if all four inputs are 1.

- (a) Draw the truth tables for XYZ (A, B, C, D).
- (b) Minimize the functions X, Y, Z, using 4-variable K-maps. Write down the Boolean expressions for the minimized Sum of Products form of each function.
- (c) Repeat the minimization process, this time deriving Product of Sums form.

**Problem 2:** Consider a combinational logic subsystem that performs a two-bit addition function. It has two 2-bit inputs A B and C D, and forms the 3-bit sum X Y Z.

- (a) Draw the truth tables for XYZ (A, B, C, D).
- (b) Minimize the functions using 4-variable K-maps to derive minimized Sum of Products forms.
- (c) What is the relative performance to compute the resulting sum bits of the 2-bit adder compared to two full adders connected together? (Hint: which has the worst

delay in terms of gates to pass through between the inputs and the final outputs, and how many gates is this?).

**Problem 3:** Show how to implement the full adder Sum (A, B, C in) and Carry (A, B, C in) in terms of:

- (a) Two 8 : 1 multiplexers;
- (b) Two 4 : 1 multiplexers;
- (c) If you are limited to 2:1 multiplexers (and inverters) only, how would you use them to implement the full adder and how many 2:1 multiplexers would you need?

**Problem 4:** Design a combinational logic subsystem with three inputs, 13, 12, 11, and two outputs, 01, 01, that behaves as follows. The outputs indicate the highest index of the inputs that is driven high. For example, if 13 is 0, 12 is 1, 11 is 1, then 01, 00 would be (10 (i.e. 12 is the highest input set to 1).

- (a) Specify the function by filling out a complete truth table.
- (b) Develop the minimized gate-level implementation using the K-map method.
- (c) Develop an implementation using two 4 : 1 multiplexers.
- (d) Compare your implementation for (b) and (c). Which is better and under what criterion?

**Problem 5:** Design a simple combinational subsystem to the following specification. the system has the ability to pass its inputs directly to its outputs when a control input, S, is not asserted. It interchanges its inputs when the control inputs S is asserted. For example, given four inputs A, B, C, D and four outputs W, X, Y, Z when S = 0, WXYZ = ACD and when S = 1, WXYZ = BCDA. Show how to implement this functionality using building blocks that are restricted to be 2 :1 multiplexers and 2 : 1 demultiplexers.

**Problem 6:** Your task is to design a combinational logic subsystem to decode a hexadecimal digit in the range of 0 through 9. A through F to drive a seven segment display. The hexadecimal numerals are as follows :

```

0 1 2 3 4 5 6 7
8 9 A b c d e f

```

Design a minimized implementation in PLA form. That is, look for common terms among the seven output functions.

**Problem 7:** Determine number of days in a month (to control watch display) used in controlling the display of a wrist-watch LCD screen.

Inputs : month, leap year flag.

Outputs : number of days.

**Problem 8:** Consider the following functions, which are five different functions over the inputs A, B, C, D.

- (1)  $F(A, B, C, D) = \sum m(1, 2, 6, 7)$
- (2)  $F(A, B, C, D) = \sum m(0, 1, 3, 9, 11, 12, 14, 15)$
- (3)  $F'(A, B, C, D) = \sum m(2, 4, 5, 6, 7, 8, 10, 13)$
- (4)  $F(A, B, C, D) = (ABC + A'B) (C+D)$
- (5)  $F(A, B, C, D) = (A + B + C) (A + B + C' + D) (A + B' + C + D') (A' + B)$

- (a) Implement these in a single PLA structure with four inputs, five outputs, and an unlimited number of product terms, how many unique product terms are there in this PLA implementation.
- (b) If you are trying to maximize the number of shared product terms across the five functions, rather than minimizing the literal count for each function independently, how many unique terms do you obtain? Draw the new K-maps with your selection of implicants that minimizes the number of unique terms across all five functions.

**Problem 9:** Consider the following Boolean function in Product of Sums form :

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

Show how to implement this function with an 8 : 1 multiplexer, with A, B, C on the control inputs and D, its complement, and the constants 0 and 1 available as data inputs.

**Problem 10:** Design a two-bit comparator with the following inputs and outputs:

Inputs : Numbers N1 and N2 to be compared

$$N1 = AB$$

$$N2 = CD.$$

Outputs : LT, GT, EQ

$$LT = 1 \text{ when } AB < CD$$

$$GT = 1 \text{ when } AB > CD$$

$$EQ = 1 \text{ when } AB = CD$$

**Problem 11:** Design a 2X2 bit multiplier :

Inputs : Numbers N1 and N2 to be multiplied

$$N1 = A1 A0$$

$$N2 = B1 B0$$

Outputs ; products : P8, P4, P2, P0

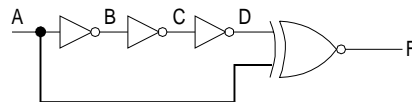
$$P0 = \text{Product with weighting } 2^0 = 1$$

$$P2 = \text{Product with weighting } 2^1 = 2$$

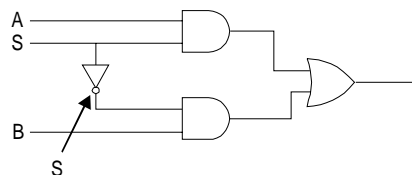
$$P4 = \text{Product with weighting } 2^2 = 4$$

$$P8 = \text{Product with weighting } 2^3 = 8.$$

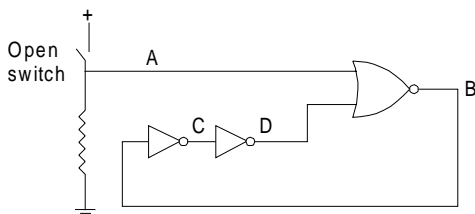
**Problem 12:** Analyse the behaviour of the Circuit below when Input A changes from one logic state to another.



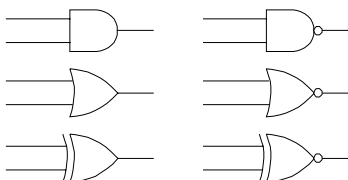
**Problem 13:** Analyse the circuit below for static hazard.



**Problem 14:** Analyse the pulse shaping circuit below :



**Problem 15:** Which of the components below can be used to build an inverter?



**Problem 16:** Consider the Equation :

$$Z = A B' C D + A B' C D' + A B C' D' + A' B C D + A B C D + A B C D' + A B' C D' + A B' C D.$$

Implement this using 2-1 multiplexers.

**Problem 17:** Use a 8-input multiplexer to generate the function

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

**Problem 18:** Implement the following function with an multiplexer.

$$y = \Sigma m(0, 1, 5, 7, 10, 14, 15)$$

**Problem 19:** Given

$$Y = \Sigma m(1, 3, 5, 6)$$

**Problem 20:** Design a 32:1 multiplexer using two 16:1 multiplexers.

**Problem 21:** Implement a 64 output demultiplexer tree using 1 × 4 DEMUX.

**Problem 22:** Realize the following functions of four variables using

- (i) 8 : 1 multiplexers
- (ii) 16 : 1 multiplexers.

**Problem 23:** Design a BCD-to Gray code converter using

- (i) 8:1 mutlplexers
- (ii) dual 4 : 1 multiplexers and some gates.

**Problem 24:** Design a Gray-to-BCD code converter using

- (i) two dual 4 : 1 multiplexers and some gates.
- (ii) one 1 : 16 demultiplexer and NAND gates.

**Problem 25:** Design a 40:1 multiplexer using 8 : 1 multiplexers.

**Problem 26:** Implement the following combinational logic circuit using a 4 to 16 line decoder.

$$Y_1 = \Sigma m (2, 3, 9)$$

$$Y_2 = \Sigma m (10, 12, 13)$$

$$Y_3 = \Sigma m (2, 4, 8)$$

$$Y_4 = \Sigma m (1, 2, 4, 7, 10, 12)$$



# PROGRAMMABLE LOGIC DEVICES

---

## 5.0 INTRODUCTION

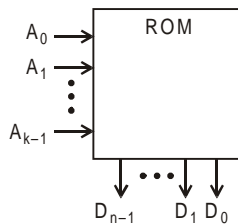
Digital circuit construction with small-scale integrated (SSI) and medium-scale integrated (MSI) logic has long been a basis of introductory digital logic design (refer Chap. 3). In recent times, designs using complex programmable logic such as programmable array logic (PLA) chips and field programmable gate arrays (FPGAs) have begun replacing these digital circuits.

This chapter deals with devices that can be programmed to realize specified logical functions. Since evolution of programmable logic devices (PLDs) started with programmable ROM, it introduces ROMs and show how they can be used as a universal logic device and how simple programmable logic devices can be derived from ROMs. It also gives an overview of Complex Programmable Logic Devices (CPLDs) and Field Programmable Gate Arrays (FPGAs).

## 5.1 READ ONLY MEMORY (ROM)

A Read Only Memory (ROM) as shown in Fig. 5.1 is a matrix of data that is accessed one row at a time. It consists of  $k$  input lines and  $n$  output lines. Each bit combination of output lines is called a word while each bit combination of input variables is called an address. The number of bits per word is equal to the number of output lines  $n$ . A ROM is describe by the number of words  $2^k$ , the number of bits per word  $n$ .

The  $A$  inputs are address lines used to select one row (called a word) of the matrix for access. If  $A = i$ , then row  $i$  is selected and its data appears on the output terminals  $D$ . In this case we say that the contents of row  $i$  are read from the memory.



**Fig. 5.1**

If there are  $k$  address lines, then there are  $2^k$  words in the ROM .The number of bits per word is called the word size. The data values of the words are called the contents of the memory and are said to be stored in the memory. The term read only refers to the property that once data is stored in a ROM, either it cannot be changed, or it is not changed very often.

ROM can be viewed as a combinational circuit with AND gates connected as a decoder and number of OR gates equal to the number of outputs. Internally a ROM contains a decoder and a storage array as shown in the Fig. 5.2.

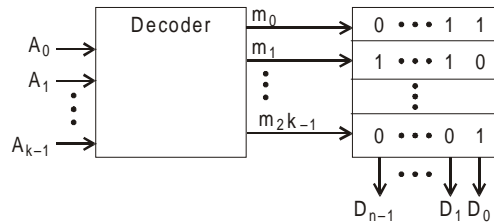


Fig. 5.2

When the address is  $i$ , the  $i$ th output of the decoder  $m_i$  is activated selecting row  $i$  of the data array. Functionally the data array can be viewed as a programmable OR array. Each column acts as a stack of OR gates as shown in the Fig. 5.3.

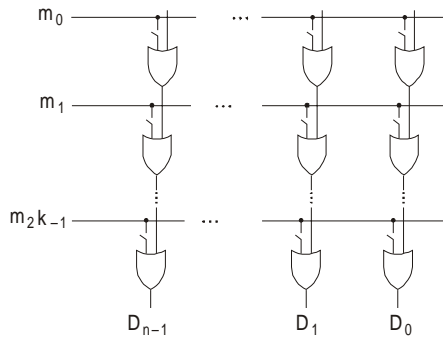


Fig. 5.3

Depending on the stored value (0/1) switch is open or closed. If a 0 is stored in a row, the switch is open and if a 1 is stored, the switch is closed. The type of ROM is determined by the way the switches are set or reset (i.e., programmed).

(I) *Mask programmed ROMs*: In the mask programmed ROMs, switch is realized at the time the ROM is manufactured. Either a connection is made by putting in a wire, or the connection is left open by not putting in a wire.

(II) *Field programmable ROMs (PROMs)*: In the field programmable ROMs, switch is realized by a fuse. When the ROM is manufactured all switches are closed since all the fuses are intact. To open a switch the fuse is blown by sending a larger than usual current through it. Once a fuse is blown, it can not be reinstalled.

(III) *Erasable ROMs (EPROMs)*: In the erasable ROMs switch is realized by a special kind of fuse that can be restored to its usual closed state, usually by the insertion of extra energy (e.g., shining ultraviolet light on the fuse). All fuses are reset when this is done.

(IV) *Electrically Programmable ROMs (EEPROMs)*: In the electrically programmable ROMs fuses are reset by the application of larger than usual currents. Sometimes subsections of the ROM can be reset without resetting all fuses.

Consider a 32×8 ROM as shown in Fig. 5.4. The ROM consists of 32 words of bit size 8. That is, there are 32 distinct words stored which may be available through eight output lines. The five inputs are decoded into 32 lines and each output of the decoder represents one of the minterms of a function of five variables. Each one of the 32 addresses selects only one output from the decoder. The 32 outputs of the decoder are connected through links to each OR gate.

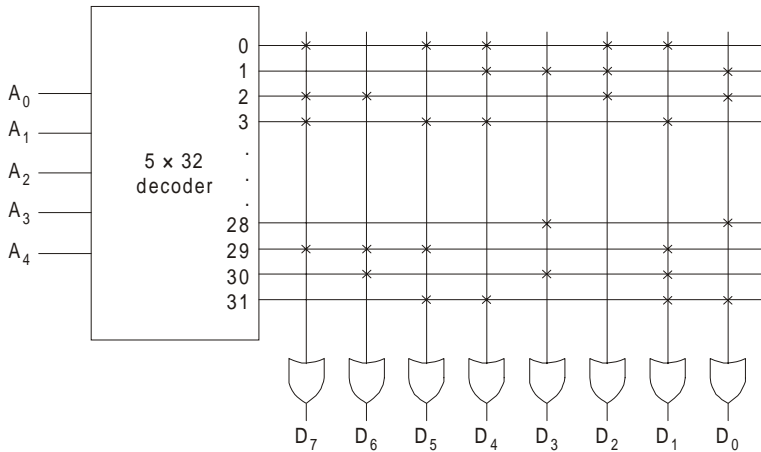


Fig. 5.4

### 5.1.1 Realizing Logical Functions with ROM

The ROM is a two-level implementation in sum of minterms form. ROMs with  $k$  address lines and  $n$  data terminals can be used to realize any  $n$  logical functions of  $k$  variables. For this one have to simply store the truth table for each function in a column of the ROM data array. The advantage of implementing logical functions by means of some form of programmable ROM, we have the possibility of reconfigurable logic. That is, the same hardware being reprogrammed to realize different logic at different times. But the disadvantage of using large ROMs to realize logical functions is that, the ROMs are much slower than gate realizations of the functions. The following example explains the procedure for realizing logical functions.

**Example.** Design a combinational circuit using a ROM that accepts a 2-bit number and generates an output binary number equal to the square of the input number.

**Step 1:** Derive the truth table for the combinational circuit. For the given example the truth table is

Inputs		Output				Equivalent decimal
$A_1$	$A_0$	$B_3$	$B_2$	$B_1$	$B_0$	
0	0	0	0	0	0	0
0	1	0	0	0	1	1
1	0	0	1	0	0	4
1	1	1	0	0	1	9

**Step 2:** If possible, reduce the truth table for the ROM by using certain properties in the truth table of the combinational circuit. For the given example, two inputs and four outputs are needed to accommodate all possible numbers.

Since output  $B_0$  is always equal to input  $A_0$ , therefore there is no need to generate  $B_0$  with a ROM. Also  $B_1$  is known as it is always 0. Therefore we need to generate only two outputs with the ROM; the other two are easily obtained.

**Step 3:** Find the minimum size of ROM from step 2

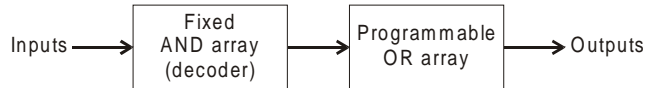
The minimum size ROM needed must have two inputs and two outputs. Two inputs specify four words, so the ROM size must be  $4 \times 2$ . The two inputs specify four words of two bits each. The other two outputs of the combinational circuit are equal to 0 ( $D_1$ ) and  $A_0(D_0)$ .

$A_1$	$A_0$	$D_3$	$D_2$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	0

ROM truth table

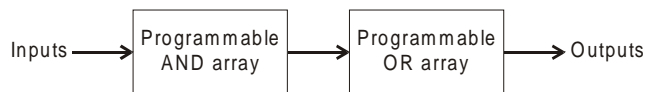
### 5.2 PROGRAMMABLE LOGIC ARRAYS

The first type of user-programmable chip that could implement logic circuits was the Programmable Read-Only Memory (PROM), in which address lines can be used as logic circuit inputs and data lines as outputs. Fig. 5.5 shows the basic configuration of PROM.



**Fig. 5.5** Basic configuration of Programmable Read-Only Memory (PROM)

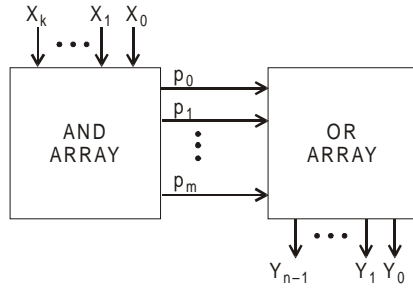
Logic functions, however, rarely require more than a few product terms, and a PROM contains a full decoder for its address inputs. PROMs are thus an inefficient architecture for realizing logic circuits, and so are rarely used in practice for that purpose. The first device developed later specifically for implementing logic circuits was the Field-Programmable Logic Array (FPLA), or simply PLA for short. A PLA consists of two levels of logic gates: a programmable “wired” AND-plane followed by a programmable “wired” OR-plane.



**Fig. 5.6** Basic configuration of Programmable Logic Array (PLA)

A PLA is structured so that any of its inputs (or their complements) can be AND’ed together in the AND-plane; each AND-plane output can thus correspond to any product term of the inputs. Similarly, each OR plane output can be configured to produce the logical sum of any of the AND-plane outputs. With this structure, PLAs are well-suited for implementing logic functions in sum-of-products form. They are also quite versatile, since both the AND terms and OR terms can have many inputs (this feature is often referred to as *wide* AND and OR gates).

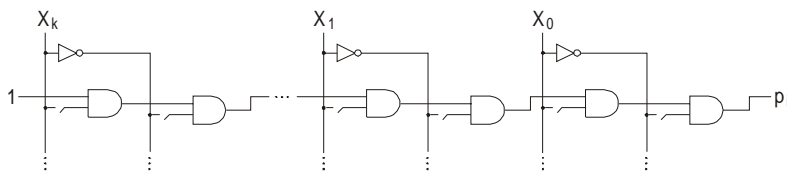
Programmable Logic Arrays (PLAs) have the same programmable OR array as a ROM, but also have a programmable AND array instead of the decoder as shown in Fig. 5.7. The programmable AND array can be used to produce arbitrary product terms, not just minterms. While the decoder produces all minterms of  $k$  variables.



**Fig. 5.7**

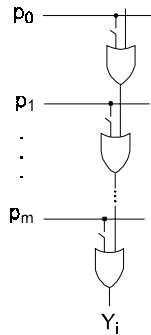
Since the number of possible product terms  $m$  in PLA is much less than the number of possible minterms  $2^k$ , so some functions may not be realizable in PLA.

The structure of a row of the programmable AND array is shown in Fig. 5.8 (a).



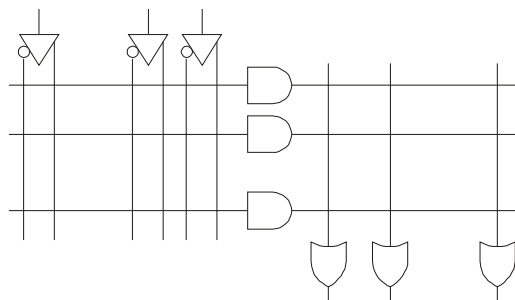
**Fig. 5.8 (a)**

and of a column of the programmable OR array is shown in Fig. 5.8 (b).



**Fig. 5.8 (b)**

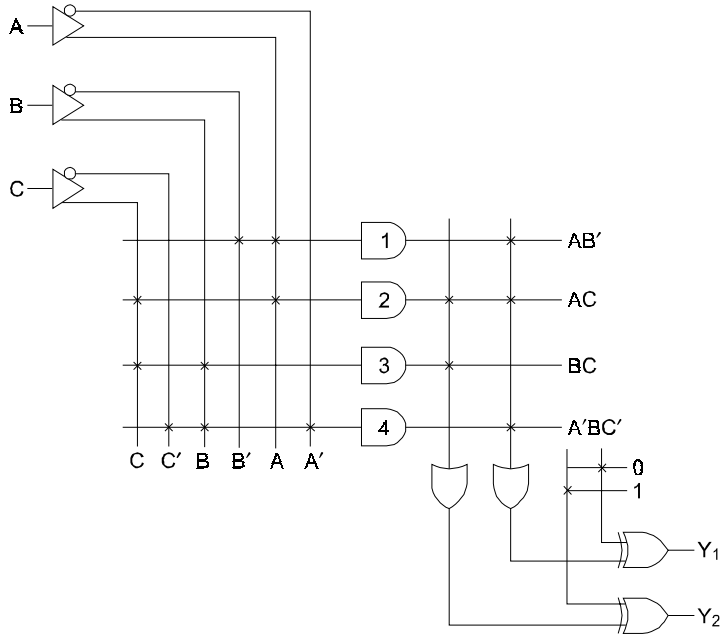
While the notation of a PLA is shown in Fig. 5.9.



**Fig. 5.9**

### 5.2.1 Realizing Logical Functions with PLAs

During implementation (or programming) a dot (or cross) is placed at an intersection if the variable is to be included in the product term or to sum term. For example a PLA with 3 inputs, 4 product terms and 2 outputs is shown.



**Example.** Implement the following with PLA.

$$P1 = A' \cdot C'$$

$$P2 = A' \cdot C$$

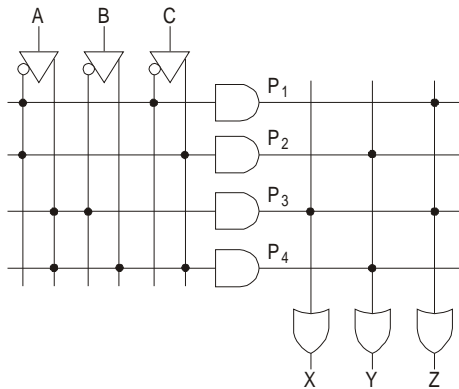
$$P3 = A \cdot B'$$

$$P4 = A \cdot B \cdot C$$

$$X = P3 = A \cdot B'$$

$$Y = P2 + P4 = A' \cdot C + A \cdot B \cdot C$$

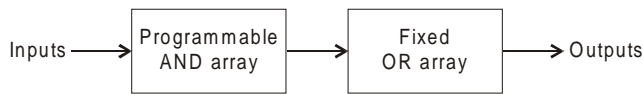
$$Z = P1 + P3 = A' \cdot C' + A \cdot B'$$



**Note:** In this implementation dot is placed at intersection but cross can be used for the same.

### 5.3 PROGRAMMABLE ARRAY LOGIC (PAL)

When PLAs were introduced in the early 1970s, by Phillips, their main drawbacks are that they are expensive to manufacture and offered poor speed-performance. Both disadvantages are due to the two levels of configurable logic, because programmable logic planes were difficult to manufacture and introduced significant propagation delays. To overcome these weaknesses, Programmable Array Logic (PAL) devices are developed. As Fig. 5.10 (a) illustrates, PALs feature only a single level of programmability, consisting of a programmable “wired” AND plane that feeds fixed OR-gates.

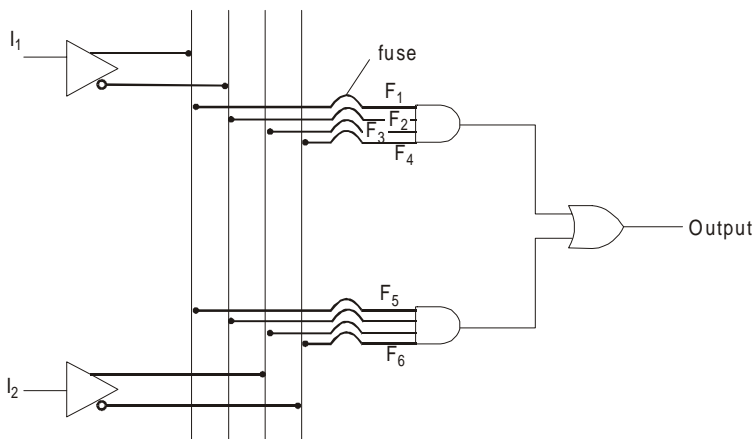


**Fig. 5.10 (a)** Basic configuration of Programmable Array Logic (PAL)

To compensate for lack of generality incurred because the OR Outputs plane is fixed, several variants of PALs are produced, with different numbers of inputs and outputs, and various sizes of OR-gates. PALs usually contain flip-flops connected to the OR-gate outputs so that sequential circuits can be realized. PAL devices are important because when introduced they had a profound effect on digital hardware design, and also they are the basis for more sophisticated architectures. Variants of the basic PAL architecture are featured in several other products known by different acronyms. All small PLDs, including PLAs, PALs, and PAL-like devices are grouped into a single category called Simple PLDs (SPLDs), whose most important characteristics are low cost and very high pin-to-pin speed-performance.

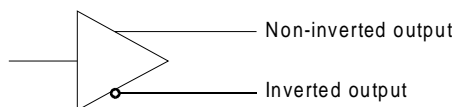
While the ROM has a fixed AND array and a programmable OR array, the PAL has a programmable AND array and a fixed OR array. The main advantage of the PAL over the PLA and the ROM is that it is faster and easier to fabricate.

Fig. 5.10 (b) represents a segment of an unprogrammed PAL.

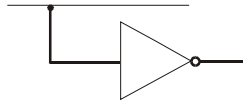


**Fig. 5.10 (b)** PAL segment

The symbol



Represents an input buffer which is logically equivalent to



A buffer is used because each PAL input have to drive many AND gate inputs. When the PAL is programmed, the fusible links ( $F_1, F_2, \dots, F_8$ ) are selectively blown to leave the desired connection to the AND gate inputs. Connections to the AND gate inputs in a PAL are represented by X's as shown

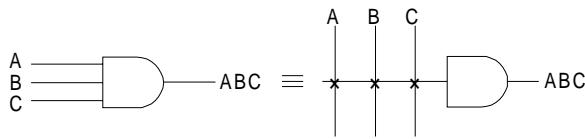
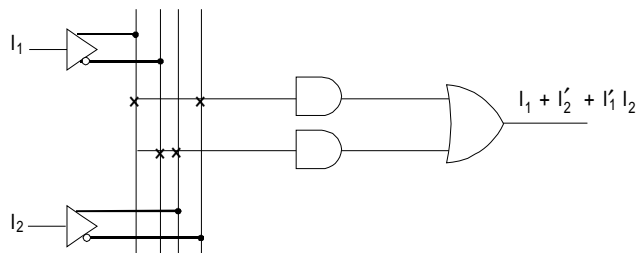


Fig. 5.10 (c) shows the use of PAL segment of Fig. 5.10 (b) to realize the function  $I_1' I_2' + I_1' I_2$ . The X's indicate that the  $I_1'$  and  $I_2'$  lines are connected to the first AND gate, and the  $I_1$  and  $I_2$  lines are connected to the other gate



**Fig. 5.10 (c)** Programmed PAL Example

In early PALs, only seven product terms could be summed into an OR gate. Therefore, not all functions could be realized with these PLAs. Also, the output was inverted in these early PALs so that what was really realized is

$$(P1 + P2 + \dots + P7)' = P1' \cdot P2' \cdot \dots \cdot P7'$$

Example of first-generation PAL is PAL 16L8 having following features.

- 10 input, 2 complemented outputs, 6 I/O pins
- Programmable (one AND term) 3-state outputs
- Seven product terms per output
- 20 pin chip
- 10 input (14 for 20V8)

A sum of products function with a small number of product terms may require a large number product terms when realized with a PAL.

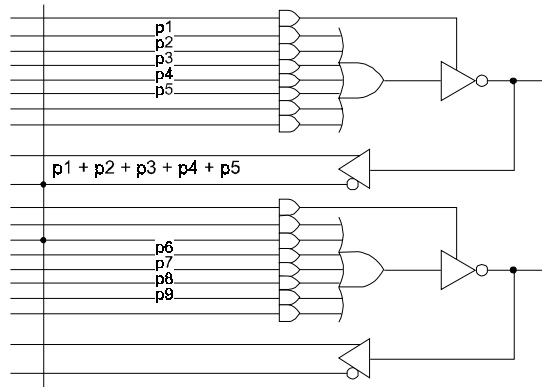
For example: To implement the function  $Z = A \cdot B \cdot C + D \cdot E \cdot F$

The inverted output of the function  $Z'$  is given as

$$\begin{aligned} Z' &= (A \cdot B \cdot C)' \cdot (D \cdot E \cdot F)' = (A' + B' + C) \cdot (D' + E' + F) \\ &= A' \cdot D' + A' \cdot E' + A' \cdot F + B' \cdot D' + B' \cdot E' + B' \cdot F \\ &\quad + C' \cdot D' + C' \cdot E' + C' \cdot F \\ &= p1 + p2 + p3 + p4 + p5 + p6 + p7 + p8 + p9 \end{aligned}$$



This has nine product terms and could not be realized in one pass with the early PALs. The only way to realize this function in a PAL is to use two passes as shown.



### 5.3.1 Commercially Available SPLDs

For digital hardware designers for the past two decades, SPLDs are very important devices. SPLDs represent the highest speed-performance FPDs available, and are inexpensive. They are also straightforward and well understood.

Two of the most popular SPLDs are the PALs produced by Advanced Micro Devices (AMD) known as the 16R8 and 22V10. Both of these devices are industry standards and are widely second-sourced by various companies. The name “16R8” means that the PAL has a maximum of 16 inputs (there are 8 dedicated inputs and 8 input/outputs), and a maximum of 8 outputs. The “R” refers to the type of outputs provided by the PAL and means that each output is “registered” by a D flip-flop. Similarly, the “22V10” has a maximum of 22 inputs and 10 outputs. Here, the “V” means each output is “versatile” and can be configured in various ways, some configurations registered and some not.

Another widely used and second sourced SPLD is the Altera Classic EP610. This device is similar in complexity to PALs, but it offers more flexibility in the way that outputs are produced and has larger AND- and OR-planes. In the EP610, outputs can be registered and the flip-flops are configurable as any of D, T, JK, or SR.

In addition to the SPLDs mentioned above many other products are commercial available. All SPLDs share common characteristics, like some sort of logic planes (AND, OR, NOR, or NAND), but each specific product offers unique features that may be particularly suitable for some applications.

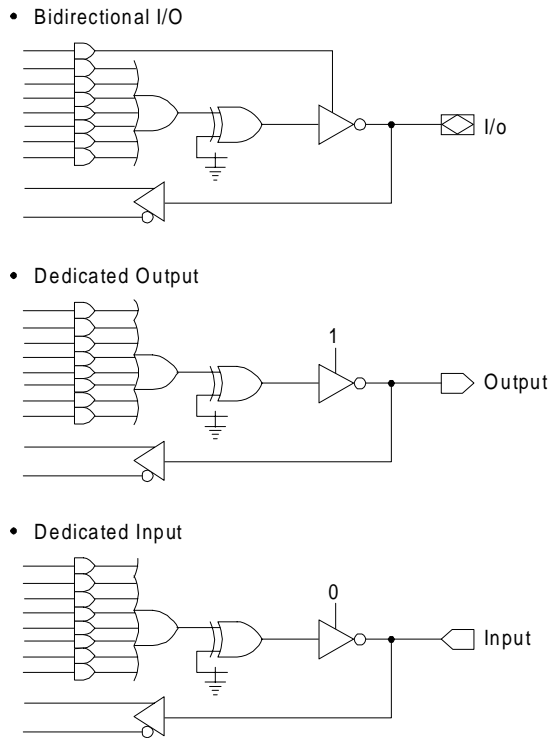
### 5.3.2 Generic Array Logic (GAL)

Generic Array Logic (GAL) is a programmable logic device that can be configured to emulate many earlier PLDs including those with internal flip-flops. GAL 16V8C and 20V8C are examples of Generic Array Logic. The only difference between the two is that the 16V8 is a 20-pin chip and the 20V8 is a 24-pin chip, which uses the extra pins for inputs. The characteristics of these devices are:

- 10 input (14 for 20V8)
- Programmable (one AND term) 3-state outputs
- Seven or eight product terms per output
- Programmable output polarity

- Realize either true or complemented output signal
- Realize either POS or SOP directly

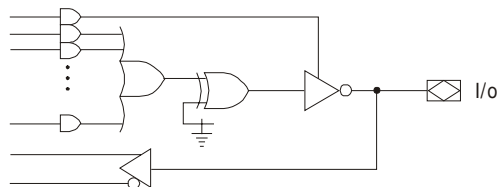
When using GAL as a combinational device, All outputs can be programmed to one of the following three configurations except that the two end outputs have some minor limitations as illustrated by Fig. 5.11.



**Fig. 5.11**

For example GAL 22V10C is a 24-pin chip having 12 input terminals and 10 input/output terminals. Among outputs, two of the outputs can have up to 8 product terms, two have 10, two have 12, two have 14 and two have 16, except the output buffer control.

The Combinational Configurations for GAL 22V10C is

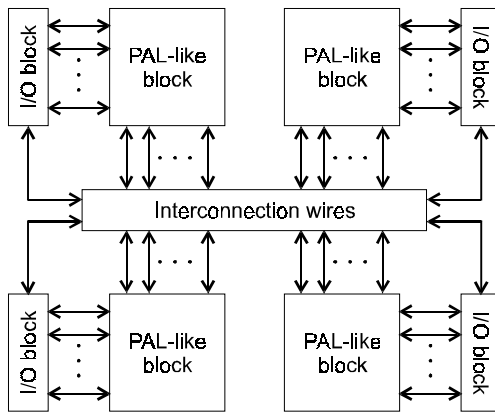


### 5.3.3 Applications of PLDs

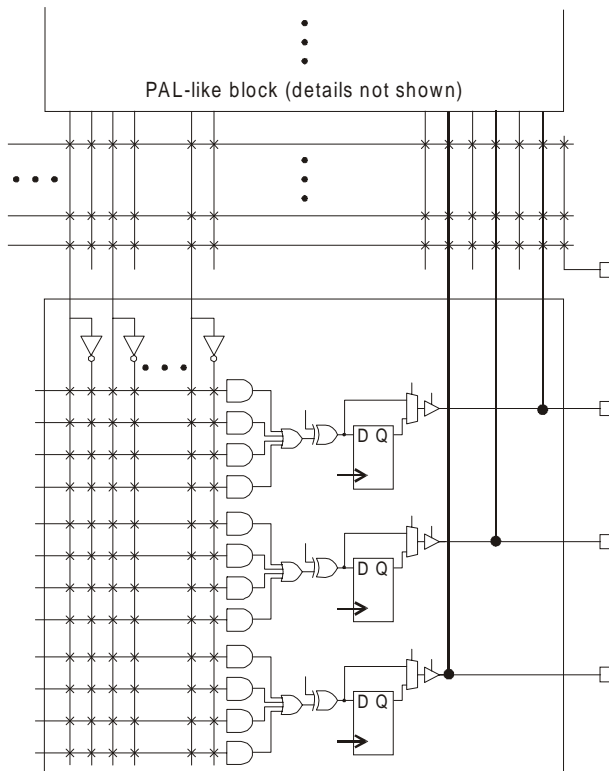
PLDs are often used for address decoding, where they have several clear advantages over the 7400-series TTL parts that they replaced. First, of course, is that one chip requires less board area, power, and wiring than several do. Another advantage is that the design inside the chip is flexible, so a change in the logic doesn't require any rewiring of the board. Rather, the decoding logic can be altered by simply replacing that one PLD with another part that has been programmed with the new design.

### 5.4 COMPLEX PROGRAMMABLE LOGIC DEVICES (CPLD)

As technology has advanced, it has become possible to produce devices with higher capacity than SPLDs (PALs). The difficulty with increasing capacity of a strict SPLD architecture is that the structure of the programmable logic-planes grows too quickly in size as the number of inputs is increased. It also significantly slows the chip down due to long rows of AND gates. The only feasible way to provide large capacity devices based on SPLD architectures is then to integrate multiple SPLDs onto a single chip, and are referred to as Complex PLDs (CPLDs) as shown in Fig. 5.12.



**Fig. 5.12 (a)**



**Fig. 5.12 (b)**

### 5.4.1 Applications of CPLDs

Because CPLDs offer high speeds and a range of capacities, they are useful for a very wide range of applications, from implementing random glue logic to prototyping small gate arrays. One of the most common uses in industry at this time, and a strong reason for the large growth of the CPLD market, is the conversion of designs that consist of multiple SPLDs into a smaller number of CPLDs.

CPLDs can realize reasonably complex designs, such as graphics controller, LAN controllers, UARTs, cache control, and many others. As a general rule-of-thumb, circuits that can exploit wide AND/OR gates, and do not need a very large number of flip-flops are good candidates for implementation in CPLDs. A significant advantage of CPLDs is that they provide simple design changes through re-programming (all commercial CPLD products are re-programmable). With programmable CPLDs it is even possible to re-configure hardware (an example might be to change a protocol for a communications circuit) without power-down.

Designs often partition naturally into the SPLD-like blocks in a CPLD. The result is more predictable speed-performance than would be the case if a design were split into many small pieces and then those pieces were mapped into different areas of the chip. Predictability of circuit implementation is one of the strongest advantages of CPLD architectures.

### 5.5 FIELD-PROGRAMMABLE GATE ARRAYS (FPGA)

Field Programmable Gate Arrays (FPGAs) are flexible, programmable devices with a broad range of capabilities. Their basic structure consists of an array of universal, programmable logic cells embedded in a configurable connection matrix. There are three key parts of FPGA structure: logic blocks, interconnect, and I/O blocks. The I/O blocks form a ring around the outer edge of the part. Each of these provides individually selectable input, output, or bi-directional access to one of the general-purpose I/O pins on the exterior of the FPGA package. Inside the ring of I/O blocks lies a rectangular array of logic blocks. And connecting logic blocks to logic blocks and I/O blocks to logic blocks is the programmable interconnect wiring.

In FPGAs, CPLD's PLDs are replaced with a much smaller *logic block*. The logic blocks in an FPGA are generally nothing more than a couple of logic gates or a look-up table and a flip-flop. The FPGAs use a more flexible and faster interconnection structure than the CPLDs. In the FPGAs, the logic blocks are embedded in a mesh or wires that have programmable interconnect points that can be used to connect two wires together or a wire to a logic block as shown in Fig. 5.13.

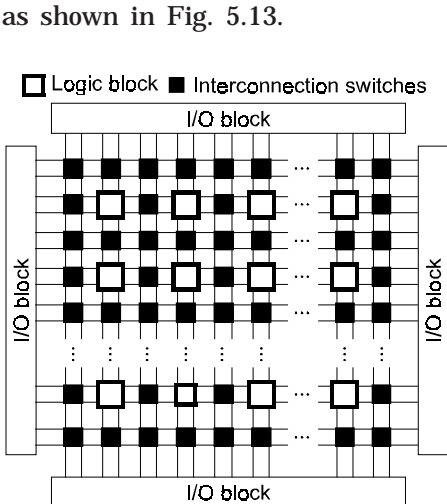


Fig. 5.13

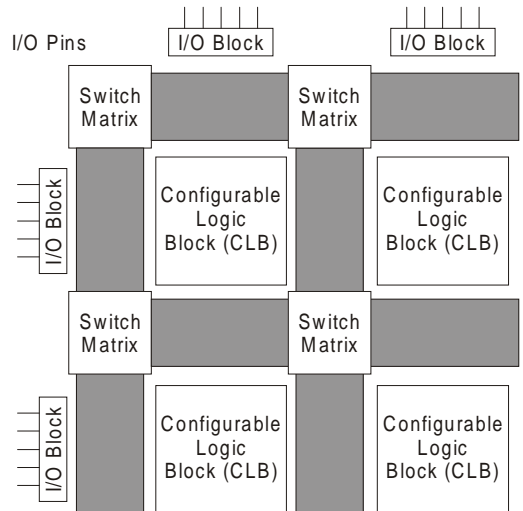
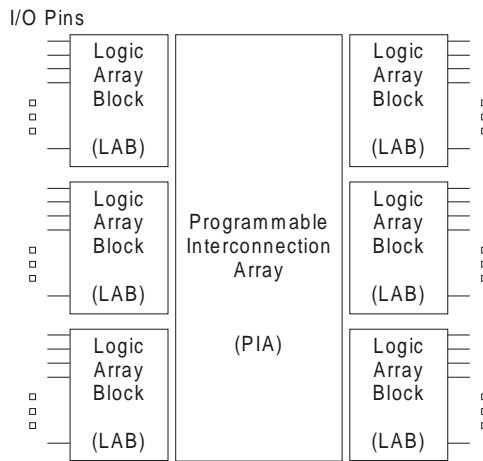


Fig. 5.14

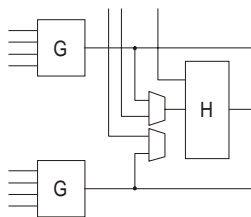
There are several architectures for FPGAs available but the two popular architectures are that, used by Xilinx and Altera. The Xilinx chips utilize an “island-type” architecture, where logic functions are broken up into small islands of 4–6 term arbitrary functions, and connections between these islands are computed. Fig. 5.14 illustrates the basic structure of the Xilinx FPGA. Altera’s architecture ties the chip inputs and outputs more closely to the logic blocks, as shown in Fig. 5.15. This architecture places the logic blocks around one central, highly connected routing array.

The circuits used to implement combinational logic in logic blocks are called lookup tables (LUT). For example the LUT in the Xilinx XC4000 uses three ROMs to realize the LUTs and generate the following classes of logical functions:



**Fig. 5.15**

- Any two different functions of 4 variables each plus any other function of 3 variables.
- Any function of 5 variables. How?
- Any function of 4 variables, plus some (but not all) functions of 6 variables.
- Some (but not all) functions of up to 9 variables.



**Fig. 5.16**

The Fig. 5.16 shows the LUT’s structure in Xilinx XC4000. The boxes G and H are ROMs with 16 1-bit words and H is a ROM with 8 1-bit words. While the structure of LUT’s in Altera FPGAs are as shown in Fig. 5.17.

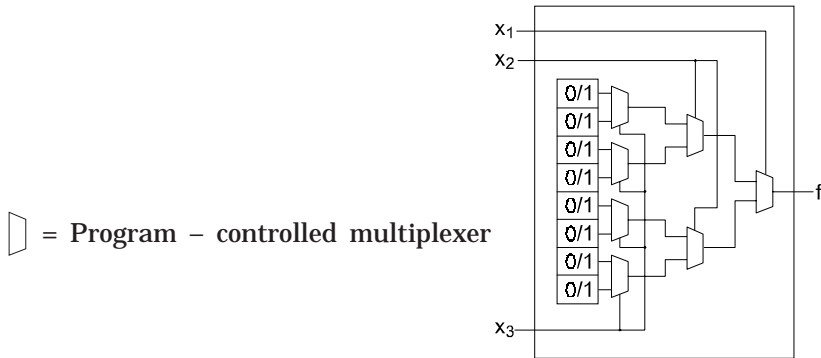
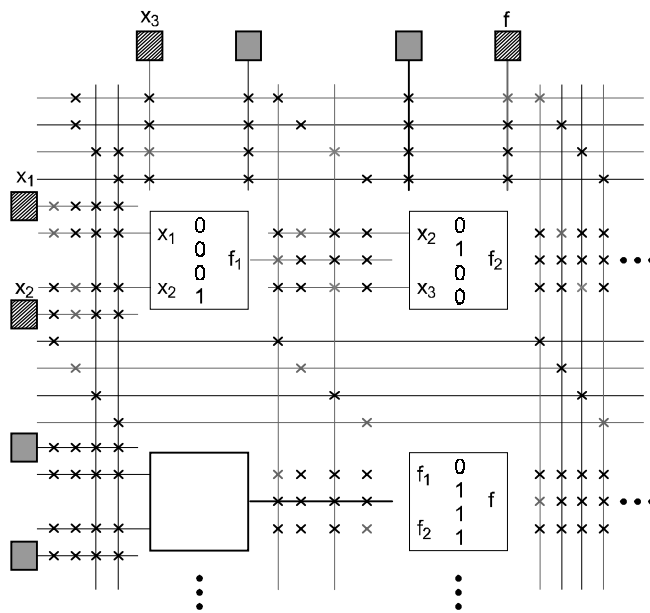


Fig. 5.17

- Example of programmed FPGA



### 5.5.1 Applications of FPGAs

FPGAs have gained rapid acceptance and growth over the past decade because they can be applied to a very wide range of applications. A list of typical applications includes: random logic, integrating multiple SPLDs, device controllers, communication encoding and filtering, small to medium sized systems with SRAM blocks, and many more.

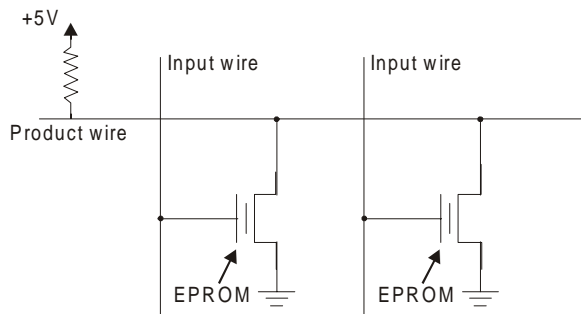
Other interesting applications of FPGAs are prototyping of designs later to be implemented in gate arrays, and also emulation of entire large hardware systems. The former of these applications might be possible using only a single large FPGA (which corresponds to a small Gate Array in terms of capacity), and the latter would involve many FPGAs connected by some sort of interconnect.

Another promising area for FPGA application, which is only beginning to be developed, is the usage of FPGAs as custom computing machines. This involves using the programmable parts to “execute” software, rather than compiling the software for execution on a regular CPU.

### 5.6 USER-PROGRAMMABLE SWITCH TECHNOLOGIES

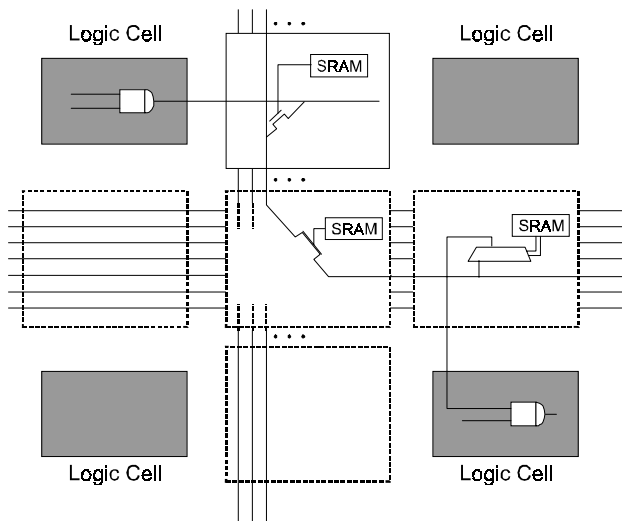
The first type of user-programmable switch developed was the *fuse* used in PLAs. Although fuses are still used in some smaller devices, but for higher density devices, where CMOS dominates the IC industry, different approaches to implementing programmable switches have been developed. For CPLDs the main switch technologies (in commercial products) are floating gate transistors like those used in EPROM and EEPROM, and for FPGAs they are SRAM and antifuse. Each of these are briefly discussed below.

An EEPROM or EPROM transistor is used as a programmable switch for CPLDs (and also for many SPLDs) by placing the transistor between two wires in a way that facilitates implementation of wired-AND functions. This is illustrated in Fig. 5.18, which shows EPROM transistors as they might be connected in an AND-plane of a CPLD. An input to the AND-plane can drive a product wire to logic level '0' through an EPROM transistor; if that input is part of the corresponding product term. For inputs that are not involved for a product term, the appropriate EPROM transistors are programmed to be permanently turned off. A diagram for an EEPROM based device would look similar.



**Fig. 5.18** EPROM Programmable Switches.

Although there is no technical reason why EPROM or EEPROM could not be applied to FPGAs, current commercial FPGA products are based either on SRAM or antifuse technologies, as discussed below.



**Fig. 5.19** SRAM-controlled Programmable Switches

An example of usage of SRAM-controlled switches is illustrated in Fig. 5.19, showing two applications of SRAM cells: for controlling the gate nodes of pass-transistor switches and to control the select lines of multiplexers that drive logic block inputs.

The figures gives an example of the connection of one logic block (represented by the AND-gate in the upper left corner) to another through two pass-transistor switches, and then a multiplexer, all controlled by SRAM cells. Whether an FPGA uses pass-transistors or multiplexers or both depends on the particular product.

The other type of programmable switch used in FPGAs is the antifuse. Antifuses are originally open-circuits and take on low resistance only when programmed. Antifuses are suitable for FPGAs because they can be built using modified CMOS technology. As an example, Actel's antifuse structure, known as PLICE, is depicted in Fig. 5.20. The figure shows that an antifuse is positioned between two interconnect wires and physically consists of three sandwiched layers: the top and bottom layers are conductors, and the middle layer is an insulator. When unprogrammed, the insulator isolates the top and bottom layers, but when programmed the insulator changes to become a low-resistance link. PLICE uses Poly-Si and n+ diffusion as conductors and ONO as an insulator, but other antifuses rely on metal for conductors, with amorphous silicon as the middle layer.

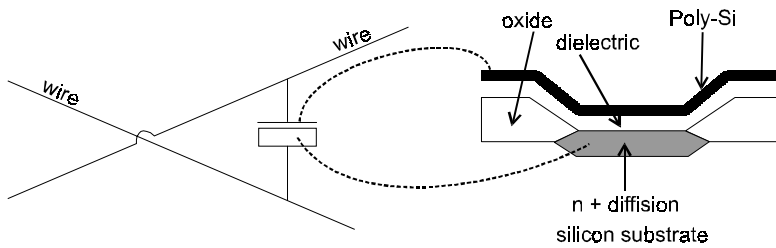


Fig. 5.20 Actel Antifuse Structure

Table lists the most important characteristics of the programming technologies discussed in this section. The left-most column of the table indicates whether the programmable switches are one-time programmable (OTP), or can be re-programmed (RP). The next column lists whether the switches are volatile, and the last column names the underlying transistor technology.

Table: Summary of Programming Technologies

Name	Re-programmable	Volatile	Technology
Fuse	no	no	Bipolar
EPROM	Yes (out of circuit)	no	UVC MOS
EEPROM	Yes (in circuit)	no	EECMOS
SRAM	Yes (in circuit)	yes	CMOS
Antifuse	no	no	CMOS+

5.7 EXERCISES

1. Realize the following functions using PLA

$$f_1(A, B, C) = \Sigma(0, 2, 4, 5)$$

$$f_2(A, B, C) = \Sigma(1, 5, 6, 7)$$



2. Realize the following functions using PLA

$$f_1 = \Sigma (1, 2, 3, 5)$$

$$f_2 = \Sigma (2, 5, 6, 7)$$

$$f_3 = \Sigma (0, 4, 6)$$

3. What is a PLA ? Describe its uses.  
 4. What is ROM ? Describe using block diagram. What size ROM would it take to implement a binary multiplier that multiplies two 4 bit-numbers.  
 5. Implement the combinational circuit specified by the truth table given

<i>Inputs</i>		<i>Outputs</i>	
$A_1$	$A_0$	$F_1$	$F_2$
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	0

6. Derive the PLA program table for a combinational circuit that squares a 3-bit number. Minimize the number of product terms.  
 7. Implement the problem 6 with the ROM.  
 8. List the PLA program table for the BCD-to-excess-3 code converter.  
 9. Write short notes on user programmable switch technologies.  
 10. Write short notes on following:
- |           |           |
|-----------|-----------|
| (i) ROM   | (ii) PLA  |
| (iii) PAL | (iv) GAL  |
| (v) CPLD  | (vi) FPGA |

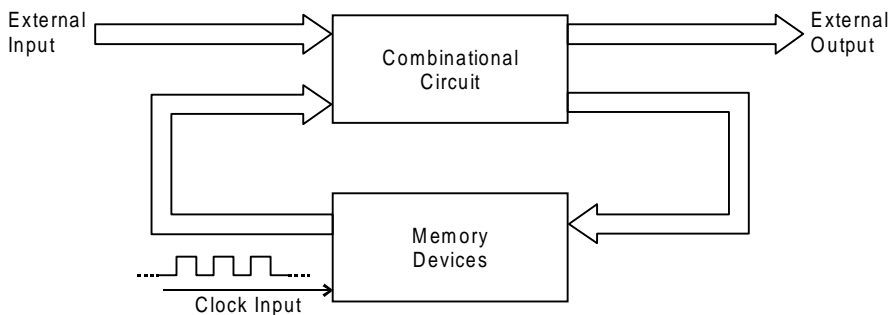
# 6

CHAPTER

## SYNCHRONOUS (CLOCKED) SEQUENTIAL CIRCUITS

### 6.0 INTRODUCTION

In the earlier chapters, we studied the digital circuits whose output at any instant of time are entirely dependent on the input present at that time. Such circuits are called as combinational circuits on the other hand **sequential circuits** are those in which the output at any instant of time is determined by the applied input and past history of these inputs (i.e. present state). Alternately, sequential circuits are those in which output at any given time is not only dependent on the input, present at that time but also on previous outputs. Naturally, such circuits must record the previous outputs. This gives rise to memory. Often, there are requirements of digital circuits whose output remain unchanged, once set, even if the inputs are removed. Such devices are referred as “memory elements”, each of which can hold 1-bit of information. These binary bits can be retained in the memory indefinitely (as long as power is delivered) or untill new information is feeded to the circuit.



**Fig. 6.1** Block Diagram of a Sequential Circuit

A block diagram of a sequential circuit is shown in Fig. 6.1. A **Sequential circuit** can be regarded as a collection of memory elements and combinational circuit, as shown in Fig. 6.1. A feedback path is formed by using memory elements, input to which is the output of combinational circuit. The binary information stored in memory element at any given time is defined as the **state** of sequential circuit at that time. Present contents of memory elements is referred as the **present state**. The combinational circuit receive the signals from external input and from the memory output and determines the external output. They also determine the condition and binary values to change the state of memory. The new contents of the memory elements are referred as **next state** and depend upon the external

input and present state. Hence a sequential circuit can be completely specified by a time sequence of inputs, outputs and the internal states. In general, clock is used to control the operation. The clock frequency determines the speed of operation of a sequential circuit.

There exist two main category of sequential circuits, namely synchronous and asynchronous sequential circuits.

A sequential circuit whose behaviour depends upon the sequence in which the inputs are applied, are called **Asynchronous Sequential Circuits**. In these circuits, outputs are affected whenever a change in inputs are detected. Memory elements used in asynchronous circuits mostly, are time delay devices. The memory capability of time delay devices are due to the propagation delay of the devices. Propagation delay produced by the logic gates are sufficient for this purpose. Hence “An Synchronous sequential circuit can be regarded as a combinational circuit with feedback”. However feedback among logic gates make the asynchronous sequential circuits, often susceptible to instability. As a result they may become unstable. This makes the design of asynchronous circuits very tedious and difficult.

A **Synchronous Sequential Circuit** may be defined as a sequential circuit, whose state can be affected only at the discrete instants of time. The synchronization is achieved by using a timing device, termed as **System Clock Generator**, which generates a periodic train of clock pulses. The clock pulses are feeded to entire system in such a way that internal states (i.e. memory contents) are affected only when the clock pulses hit the circuit. A synchronous sequential circuit that uses clock at the input of memory elements are referred as **Clocked Sequential circuit**.

The clocked sequential circuits use a memory element known as **Flip-Flop**. A flip-flop is an electronic circuit used to store 1-bit of information, and thus forms a 1-bit memory cell. These circuits have two outputs, one giving the value of binary bit stored in it and the other gives the complemented value. In this chapter it is our prime concern to discuss the characteristics of most common types of flip-flops used in digital systems.

The real difference among various flip-flops are the number of inputs and the manner in which binary information can be entered into it. In the next section we examine the most general flip-flops used in digital systems.

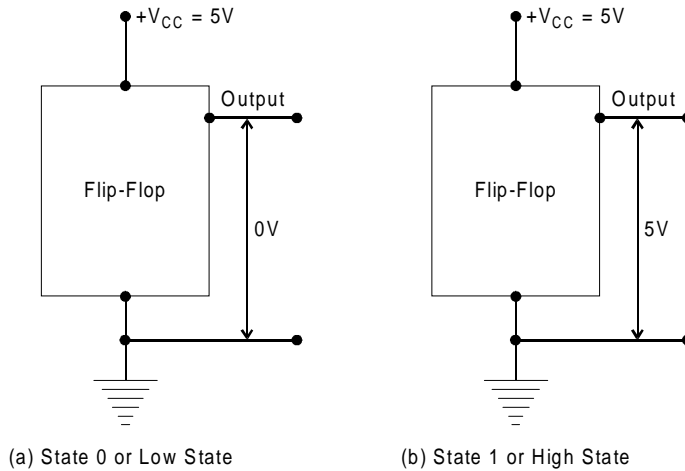
## 6.1 FLIP-FLOPS

We have earlier indicated that flip-flops are 1-bit memory cells, that can maintain the stored bit for desired period of time.

A **Bistable** device is one in which two well defined states exist, and at any time the device could assume either of the stable states. A **stable state** is a state, once reached by a device does not changes untill and unless something is done to change it. A toggle switch has two stable states, and can be regarded as a bistable device. When it is closed, it remains closed (A stable state) untill some one opens it. When it is open, it remains open (2nd stable state) untill some one closes it i.e. make it to return to its first stable state. So it is evident that the switch may be viewed as 1-bit memory cell, since it maintains its state (either open or close). Infact any bistable device may be referred as 1-bit memory cell.

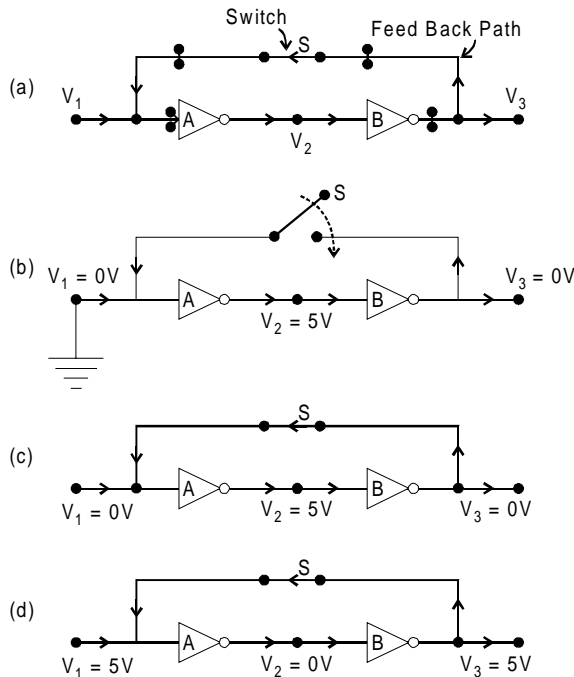
A **Flip-Flop** may also be defined as a bistable electronics device whose two stable states are 0V and + 5V corresponding to Logic 0 and Logic 1 respectively. The two stable states and flip-flop as a memory element is illustrated in Fig. 6.2. Fig. 6.2 (a) shows that the flip-flop is in ‘State 0’ as output is 0V. This can be regarded as storing Logic 0. Similarly flip-flop is said to be in ‘State 1’, see Fig. 6.2 (b), when the output is 5 V. This can be regarded

as storing logic 1. Since at any given time flip-flop is in either of two states the flip-flop may also be regarded as Bistable Multivibrator. Since the state once reached is maintained until it is deliberately changed, the flip-flop is viewed as memory element.



**Fig. 6.2** Flip-Flop as Bistable Device

The basic memory circuit or flip-flop can be easily obtained by connecting two inverters (Not gates) in series and then connecting the output of second inverter to the input of first inverter through a feedback path, as shown in Fig. 6.3(a).



**Fig. 6.3** Basic Flip-Flop or Latch 'Logic 0' = 0V, and 'Logic 1' = 5 V

It is evident from figure that  $V_1$  and  $V_3$  will always be same, due to very nature of inverters.

Let us define Logic 0 = 0V and Logic 1 = 5 V. Now open the switch 'S' to remove the feedback and connect  $V_1$  to ground, as shown in Fig. 6.3 (b). Thus input to inverter A is Logic 0 and its output would be Logic 1 which is given to the input of inverter B. Since input to inverter B is Logic 1, its output would be Logic 0. Hence input of inverter A and output of inverter B are same. Now if we close the switch S feedback path is reconnected, then ground can be removed from  $V_1$  and  $V_3$  can still be at 0V i.e. Logic 0. This is shown in Fig. 6.3 (c). This is possible because once the  $V_1$  is given 0V (i.e. Logic 0) the  $V_3$  will also be at 0V and then it can be used to hold the input to inverter A at 0V, through the feedback path. This is first stable state.

In the simpler way if we connect the  $V_1$  to 5 V and repeat the whole process, we reach to second stable state because  $V_3 = 5V$ . Essentially the  $V_3$  holds the input to inverter A (i.e.  $V_1$ ), allowing + 5V supply to be removed, as shown in Fig. 6.3 (d). Thus  $V_3 = 5 V$  can be maintained until desired period time.

A simple observation of the flip-flop shown in Fig. 6.3 (a) reveals that  $V_2$  and  $V_3$  are always complementary, i.e.  $V_2 = \overline{V_3}$  or  $V_3 = \overline{V_2}$ . This does mean that "at any point of time, irrespective of the value of  $V_1$ , both the stable states are available", see Fig. 6.3 (c) 6.3 (d). This is fundamental condition to 9 Flip-Flop.

Since the information present at the input (i.e. at  $V_1$ ) is locked or latched in the circuit, it is also referred or **Latch**.

When the output is in low state (i.e.  $V_3 = 0 V$ ), it is frequently referred as **Reset State**. Where as when the output is in high state (i.e.  $V_3 = 5 V$ ), it is conveniently called as **Set State**. Fig. 6.3 (c) and 6.3 (d) shows the reset and set states, respectively.

### 6.1.1 RS Flip-Flop

Although the basic latch shown by the Fig. 6.3 (a) was successful to memorize (or store) 1-bit information, it does not provide any convenient mean to enter the required binary bit. Thus to provide a way to enter the data circuit of Fig. 6.3 (a) can be modified by replacing the two inverters by two 2-input NOR gate or NAND gates, discussed in following articles.

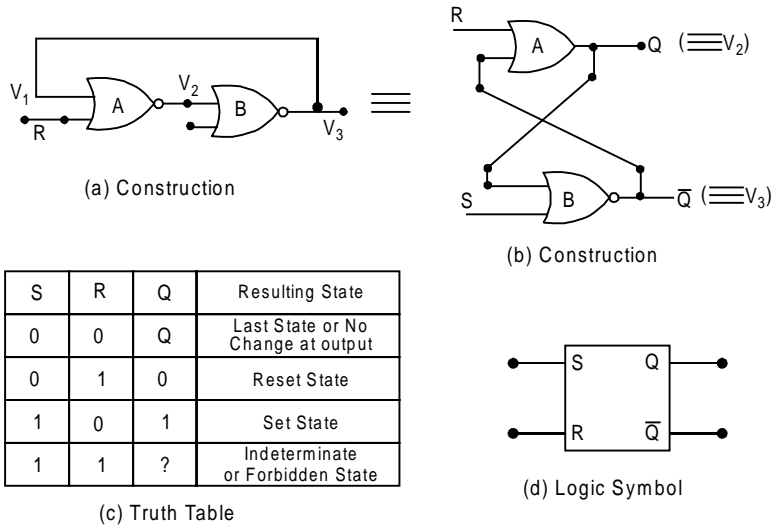
**The NOR LATCH:** The NOR latch is shown by Fig. 6.4 (a) and 6.4 (b). Notice that if we connect the inputs, labelled as R and S, to logic 0 the circuit will be same as the circuit shown in Fig. 6.3 (a) and thus behave exactly same as the NOT gate latch of Fig. 6.3 (a).

The voltage  $V_2$  and  $V_3$  are now labelled as Q and  $\overline{Q}$  and are declared as output. Regardless of the value of Q, its complement is  $\overline{Q}$  (as  $V_3 = \overline{V_2}$ ). The two inputs to this flip-flop are R and S, stand for RESET and SET inputs respectively. A '1' on input R switches the flip-flop in reset state i.e. Q = '0' and  $\overline{Q}$  = '1'. A '1' on inputs (SET input) will bring the latch into set state i.e. Q = '1' and  $\overline{Q}$  = '0'. Due to this action it is often called **set-reset latch**. The operation and behaviour is summarized in the truth table shown by Fig. 6.4 (c). Fig. 6.4 (d) displays the logic symbol of RS (or SR) flip-flop.

To understand the operation of this flip-flop, recall that a '1' at any input of a NOR gate forces its output to '0' where as '0' at an input does not affect the output of NOR gate.

When inputs are S = R = 0, first row of truth tables it does not affect the output. As a result the Latch maintains its state. For example if before application of the inputs S = R = 0, the output was Q = 1, then it remains 1 after S = R = 0 are applied. Thus when both

the inputs are low the flip-flop maintain its last state. That's why the truth table has entry Q in first row.



**Fig. 6.4** Basic NOR Gate Latch or RS (or SR) Flip-Flop

Now if  $S = 0$  and  $R = 1$ , output of gate-A goes low i.e.  $Q = 0$ . The  $Q$  is connected to input of gate-B along with  $S$  input. Thus with  $Q = 0$  both the inputs to NOR gate B are LOW. As a result  $\bar{Q} = 1$ . This  $Q$  and  $\bar{Q}$  are complementary. Since  $Q = 0$  the flip-flop is said to be in “reset state”. This is indicated by the second row of the truth table.

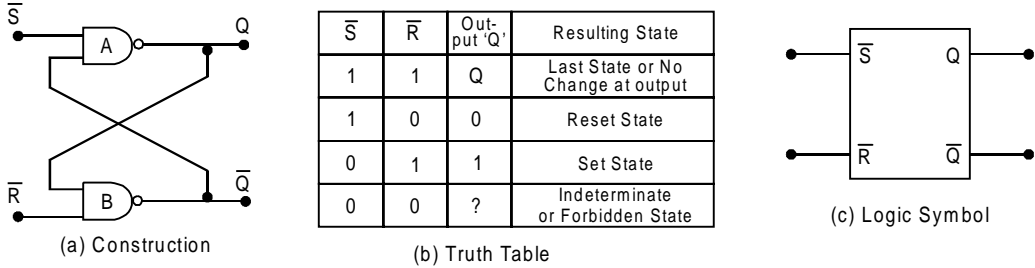
Now if  $S = 1$  and  $R = 0$  output of gate-B is LOW, making both the inputs of gate-A LOW, consequently  $Q = 1$ . This is “set state”. As  $Q = 1$  and  $\bar{Q} = 0$ , the two outputs are complementary. This is shown in third row of truth table.

When  $S = 1$  and  $R = 1$ , output of both the gates are forced to Logic 0. This conflicts with the definition that both  $Q$  and  $\bar{Q}$  must be complementary. Hence this condition must not be applied to SR flip-flop. But if due to some reasons  $S = 1$  and  $R = 1$  is applied, then it is not possible to predict the output and flip-flop state is said to be indeterminate. This is shown by the last row of truth table.

It is worth to devote some time to investigate why  $S = R = 1$  results indeterminate state while we said earlier that output of both the gates go LOW for this input. This is true due to the logic function of NOR gate that if any of the input is HIGH output is LOW. In the circuit of Fig. 6.4 (b) both  $Q$  and  $\bar{Q}$  are LOW as long as  $S$  and  $R$  are High. The problem occurs when inputs  $S$  and  $R$  goto LOW from High. The two gates can not have exactly same propagation delay. Now the gate having smaller delay will change its state to HIGH earlier than the other gate. And since this output (i.e. Logic 1) is feeded to the second gate, the output of second gate is forced to stay at Logic 0. Thus depending upon the propagation delays of two gates, the flip-flop attains either of the stable states (i.e. either  $Q = 1$  or  $Q = 0$ ). Therefore it is not possible to predict the state of flip-flop after the inputs  $S = R = 1$  are applied. That's why the fourth row of truth table contains a question mark (?). For the above reasons the input condition  $S = R = 1$  is forbidden.

**The NAND Gate Flip-Flop**

The NOT gate latch shown by Fig. 6.5 (b) may also be modified by replacing each inverter by a 2-input NAND gate as shown in Fig. 6.5 (a). This is a slightly different latch



**Fig. 6.5** NAND Gate Latch or  $\bar{S}$ ,  $\bar{R}$  Flip-Flop

from the NOR latch. We call it  $\bar{S}\bar{R}$  latch. The truth table (Fig. 6.5(b)) summarizes the operation and Fig. 6.5(c) shows the logic symbol for  $\bar{S}\bar{R}$  latch.

The name  $\bar{S}\bar{R}$  is given to this latch to indicate that intended operation is achieved on asserting logic '0' to the inputs. This is complementary to the NOR latch in which operation is performed when input is logic '1'.

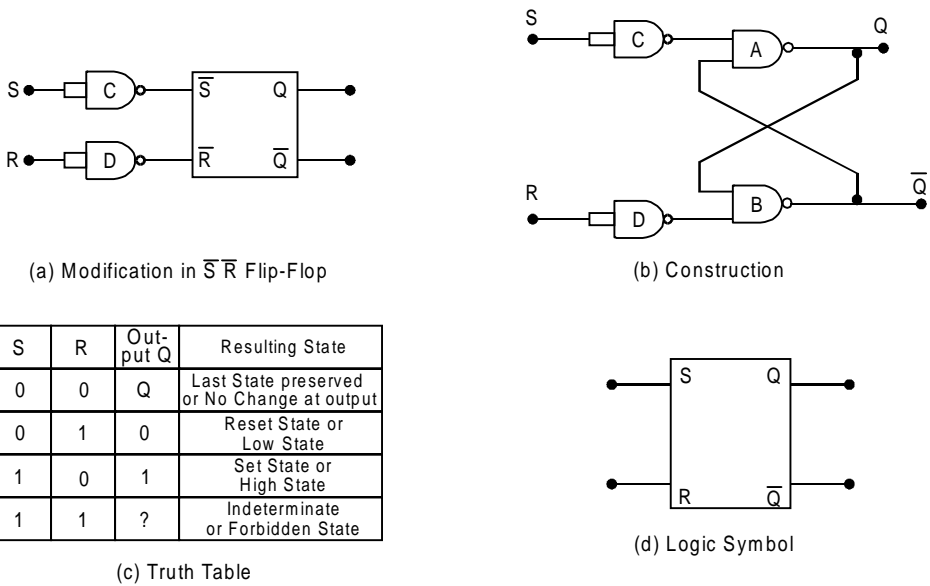
The explanation of operation of  $\bar{S}\bar{R}$  flip-flop lies in the statement that, If any input of NAND gate goes LOW the output is HIGH, whereas a '1' at any NAND input does not affect the output. Moreover, the output will be LOW only and only when all the inputs to NAND gate are HIGH.

When both  $\bar{S}$  and  $\bar{R}$  are HIGH i.e.  $\bar{S} = \bar{R} = 1$ , then the NAND output are not affected. Thus last state is maintained. When  $\bar{S} = 1$  and  $\bar{R} = 0$  then output of gate-B goes HIGH making both the inputs to NAND-A as HIGH. Consequently  $Q = 0$  which is reset state. In the similar way  $\bar{S} = 0$  and  $\bar{R} = 1$  bring the circuit to set state i.e.  $Q = 1$ . When both the inputs are LOW i.e.  $\bar{S} = \bar{R} = 0$  both  $Q$  and  $\bar{Q}$  are forced to stay HIGH which inturns lead to indeterminate state for the similar reasons given for NOR latch.

The  $\bar{S}\bar{R}$  flip-flop can be modified further by using two additional NAND gates.

These two gates, labelled as C and D, are connected at  $\bar{S}$  and  $\bar{R}$  Fig. 6.5 (a) inputs to act as NOT gate, as shown in Fig. 6.6 (a). This converts  $\bar{S}\bar{R}$  latch into a latch that behaves exactly same as the NOR gate flip-flop (i.e. NOR latch), shown in Fig. 6.4 (b). Hence this latch also is referred as SR flip-flop. The truth table and logic symbol will be same as that of NOR latch. The truth table may also be obtained by inverting all the input bits in  $\bar{S}\bar{R}$  truth table shown in Fig. 6.5 (b) as input to  $\bar{S}\bar{R}$  latch is complemented. To understand the operation of this latch, consider the Fig. 6.6 (a).

When both S and R inputs are LOW outputs of NAND gates C and D are HIGH. This is applied to the inputs of  $\bar{S}\bar{R}$  latch which maintains the last state in response to  $\bar{S} = \bar{R} = 1$  (see Fig. 6.5 (b)). In the same way, application of  $S = 0$  and  $R = 1$  results  $\bar{S} = 1$  and  $\bar{R} = 0$ , and consequently the latch attains "Reset State". Similarly the other input combination in truth table can be verified.



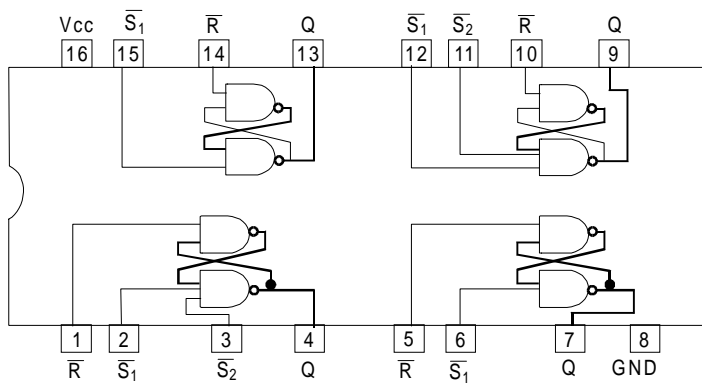
**Fig. 6.6** NAND Gate Latch or SR Flip-Flop

At this point we advise readers to verify the truth table of SR latch through the NAND gate construction of this latch, shown in Fig. 6.6 (b).

For the beginners it is worth to go back to the beginning of this article (i.e. start of 6.1.1). Infact the SR (or RS) flip-flop gives the basic building block to study and analyze various flip-flops, and their application in the design of clocked sequential circuits.

**Example 6.1.** Draw the internal block diagram alongwith pinout for IC 74LS279, a quad set reset latch. Explain its operation in brief with the help of truth table.

**Sol.** Fig. 6.7 shows the required block diagram and pinout.



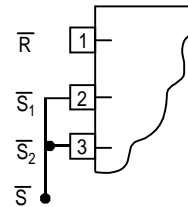
**Fig. 6.7** IC 74LS279, A Quad Set Reset Latch

From the figure it is evident that the IC contains 4  $\overline{S}\overline{R}$  latch shown earlier in Fig. 6.5. Two flip-flops have two inputs, named  $\overline{S}_1, \overline{R}$  are exact reproduction of  $\overline{S}\overline{R}$  latch shown in Fig. 6.5. Remaining two flip-flops have three inputs labelled  $\overline{S}_1, \overline{S}_2, \overline{R}$ , in which instead of single  $\overline{S}$  input we get two set inputs  $\overline{S}_1$  and  $\overline{S}_2$ . Since the latches are constructed by NAND gates a LOW either on  $\overline{S}_1$  or on  $\overline{S}_2$  will set the latch. Truth table summarizing its operation



is shown in Fig. 6.8. Not that making  $\bar{R} = 0$  and either of  $\bar{S}_1$  and  $\bar{S}_2$  LOW, leads to indeterminate state.

$\bar{S}_1$	$\bar{S}_2$	$\bar{R}$	Q	Resulting State
0	x	1	1	Set State
x	0	1	1	Set State
1	1	0	0	Reset state
1	1	1	Q	Last State
0	x	0	?	Indeterminate
x	0	0	?	Indeterminate



**Fig. 6.8** Truth Table; 'X' → don't care      **Fig. 6.9** Converting  $\bar{S}_1$  and  $\bar{S}_2$  into  $\bar{S}$

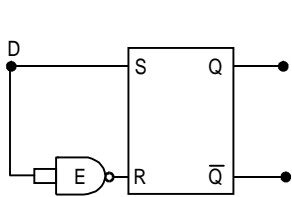
Also if  $\bar{S}_1$  and  $\bar{S}_2$  are shorted (or tied) together, as shown in Fig. 6.9, the two set inputs can be converted into single set input  $\bar{S}$ . When  $\bar{S}_1$  and  $\bar{S}_2$  are shorted together, the latch exactly becomes SR flip-flop of Fig. 6.5.

### 6.1.2 D Flip-Flop

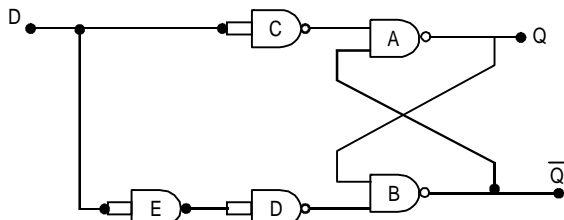
The SR latch, we discussed earlier, has two inputs S and R. At any time to store a bit, we must activate both the inputs simultaneously. This may be troubling in some applications. Use of only one data line is convenient in such applications.

Moreover the forbidden input combination  $S = R = 1$  may occur unintentionally, thus leading the flip-flop to indeterminate state.

In order to deal such issues, SR flip-flop is further modified as shown in Fig. 6.10. The resultant latch is referred as D flip-flop or D latch. The latch has only one input labelled D (called as Data input). An external NAND gate (connected as inverter) is used to ensure that S and R inputs are always complement to each other. Thus to store information in this latch, only one signal has to be generated.



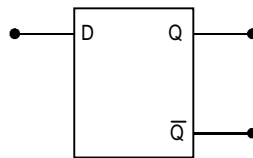
(a) Modification in S R Flip-Flop



(b) Construction

D	Q	Resulting State
0	0	Reset State or Low State
1	1	Set State or High State

(c) Truth Table



(d) Logic Symbol

**Fig. 6.10** D Flip-flop or D latch

Operation of this flip-flop is straight forward. At any instant of time the output  $Q$  is same as  $D$  (i.e.  $Q = D$ ). Since output is exactly same as the input, the latch may be viewed as a delay unit. The flip-flop always takes some time to produce output, after the input is applied. This is called propagation delay. Thus it is said that the information present at point  $D$  (i.e. at input) will take a time equal to the propagation delay to reach to  $Q$ . Hence the information is delayed. For this reason it is often called as **Delay (D) Flip-Flop**. To understand the operation of this latch, consider Fig. 6.10 (a).

As shown in figure, the  $D$  input goes directly to  $S$  and its complement is applied to  $R$  input. When data input is LOW i.e.  $D = 0$ , we get  $S = 0$  and  $R = 1$ . So flip-flop reaches to RESET State where  $Q = 0$ . When  $D = 1$  the  $S$  input receives 1 and  $R = 0$ . Thus the flip-flop goes to SET state, where  $Q = 1$ . This operation is summarized in truth table, shown in Fig. 6.10 (c). It is interesting to note that the next state of  $D$  flip-flop is independent of present state. It means that if input  $D = 1$  the next state will be SET state, whether presently it is in SET or RESET state.

Furthermore, by Fig. 6.10 (a) it is clear that the external inverter ensures that the forbidden condition  $S = R = 1$  will never arrive. The  $D$  flip-flops are popularly used as the delay devices and/or latches. In general, simply saying latch means a  $D$  flip-flop.

**Example 6.2.** A logic circuit having a single input labelled  $X$ , and two outputs  $Y_1$  and  $Y_2$  is shown in fig 6.11. Investigate the circuit and find out does this circuit represents a latch? If yes, then name the latch and draw the truth table for this circuit.

**Sol.** To investigate the circuit, we find out values of outputs  $Y_1$  and  $Y_2$  for each and every possibility of input  $X$ .

A careful inspection of circuit reveals that the portion of the circuit which consist of two NAND gates  $A$  and  $B$

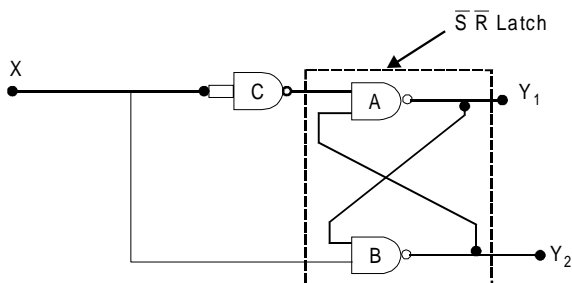


Fig. 6.11 Logic Circuit for Example 6.2

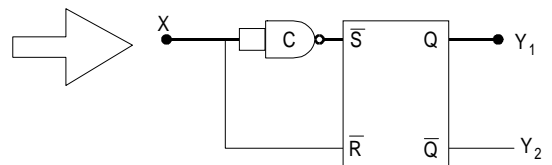


Fig. 6.12 Simplified circuit for Fig. 6.11

represent  $\overline{S}\overline{R}$  flip-flop. This portion of the circuit is surrounded by dotted lines in Fig. 6.11. The circuit is redrawn in Fig. 6.12 for simplicity. This simplification shows that input to  $\overline{S}$  is  $\overline{X}$  and input to  $\overline{R}$  is  $X$  or  $\overline{S} = \overline{X}$  and  $\overline{R} = X$ . The outputs  $Y_1$  and  $Y_2$  are nothing but the  $Q$  and  $\overline{Q}$  outputs  $\overline{S}\overline{R}$  latch i.e.  $Y_1 = Q$  and  $Y_2 = \overline{Q}$ . Thus the outputs  $Y_1$  and  $Y_2$  are always complementary.

Hence when the input  $X$  is LOW i.e.  $X = 0$ , it results in  $\overline{S} = 1$  and  $\overline{R} = 0$ . The latch is forced to reset state, in which case  $Q = 0$  and  $\overline{Q} = 1$ , consequently  $Y_1 = 0$  and  $Y_2 = 1$ . Thus for  $X = 0$  we get  $Y_1 = 0$  and  $Y_2 = 1$ . In the similar way when  $X = 1$ ,  $\overline{S} = 0$  and  $\overline{R} = 1$  making  $Y_1 = 1$  and  $Y_2 = 0$  which is set state. We now summarize these results in a truth table shown

in Fig. 6.13. From the truth table it is clear that  $Y_1 = X$  and  $Y_2 = \bar{X}$ . Thus the given circuit represents a D Latch which gives  $Q = D$  and  $\bar{Q} = \bar{D}$ . In the Figs. 6.11 and 6.12 the input D is renamed as X and the outputs Q and  $\bar{Q}$  are named as  $Y_1$  and  $Y_2$  respectively.

X	$\bar{S}$	$\bar{R}$	Q	$\bar{Q}$	$Y_1 = Q$	$Y_2 = \bar{Q}$
0	1	0	0	1	0	1
1	0	1	1	0	1	0

→

X	$Y_1$	$Y_2$
0	0	1
1	1	0

(a)
(b)

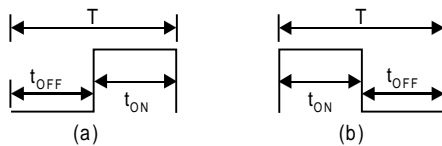
**Fig. 6.13** Truth Table for Fig. 6.12

In Fig. 6.12, if the output of gate C is connected to  $\bar{R}$  and input X is directly connected to  $\bar{S}$  then  $Y_1 (= Q) = \bar{X}$  and  $Y_2 (= \bar{Q}) = X$ . Therefore the circuit may be referred as inverted D latch.

### 6.1.3 Clocked Flip-Flops

All the flip-flops discussed earlier are said to be **transparent**, because any change in input is immediately accepted and the output changes accordingly. Since they consist of logic gates along with feedback they are also regarded as **asynchronous flip-flops**.

However, often there are requirements to change the state of flip-flop in synchronism with a train of pulses, called as **Clock**. In fact we need a control signal through which a flip-flop can be instructed to respond to input or not. Use of clock can serve this purpose.



**Fig. 6.14** Representation of Pulse

A clock signal can be defined as a train of pulses. Essentially each pulse must have two states, ON state and OFF state. Fig. 6.14 shows two alternate representation of a pulse, and Fig. 6.15 shows a clock signal. The clock pulses are characterized by the **duty cycle**, which is representative of ON time in the total time period of pulse, and is given as:

$$\text{Duty Cycle} = D = \frac{t_{ON}}{t_{ON} + t_{OFF}} \text{ or } D = \frac{t_{ON}}{T}$$

In the digital systems we need a clock with duty cycle  $D \leq 50\%$ . The OFF time of a pulse is also referred as **bit-time**. This is the time in which flip-flop remains unaffected in either of two stable states. The state of latch during this time is due to the input signals present during ON time of pulse.

State  $Q_{n+1}$  is due to the inputs present during ON time of  $(n + 1)$ th pulse i.e at  $t = nT$ . In the analysis and discussion we adopt the designation  $Q_n$  to represent “**present state**” which is the state before the ON time of  $(n + 1)$ th pulse or state just before the time  $t = nT$  in Fig. 6.15, and  $Q_{n+1}$  as “**next state**” i.e. the state just after the ON time of

$(n + 1)$ th clock pulse. Thus  $Q_n$  represents the state (or output) in bit time  $n$  and  $Q_{n + 1}$  represents the output  $Q$  in bit time  $n + 1$ , as shown in Fig. 6.15.

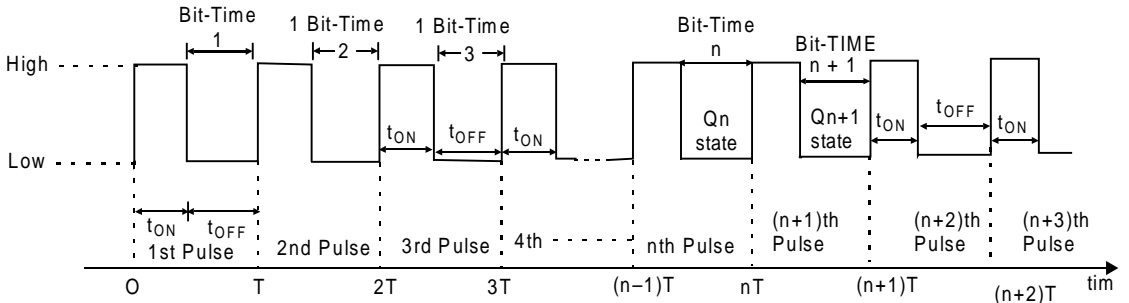


Fig. 6.15 Clock Signal-Pulse Train of Shape shown in Fig. 6.14 (b).

As we said earlier, the clock pulse can be used as a control signal. It allows the flip-flop to respond to inputs during  $t_{ON}$  period of clock, it is called **enabling** the flip-flop. Where as the flip-flop is instructed, not to respond to inputs during  $t_{OFF}$  period of clock, i.e. flip-flop maintains its output irrespective of changes in input. This is called **disabling** the flip-flop.

In this way it is possible to **strobe** or **clock** the flip-flop in order to store the information at any time said alternately clocking allow us to selectively enable or disable the flip-flop which is a necessary requirement in large digital systems.

**Clocked SR Flip-Flop:** A simple way to get a clocked SR flip-flop is to AND the inputs signals with clock and then apply them to S and R inputs of flip-flop as shown in fig. 6.16 (a). For the simplicity SET and RESET inputs of unlocked SR latch are labelled  $S_1$  and  $R_1$  respectively. Where as external inputs are labelled S and R.

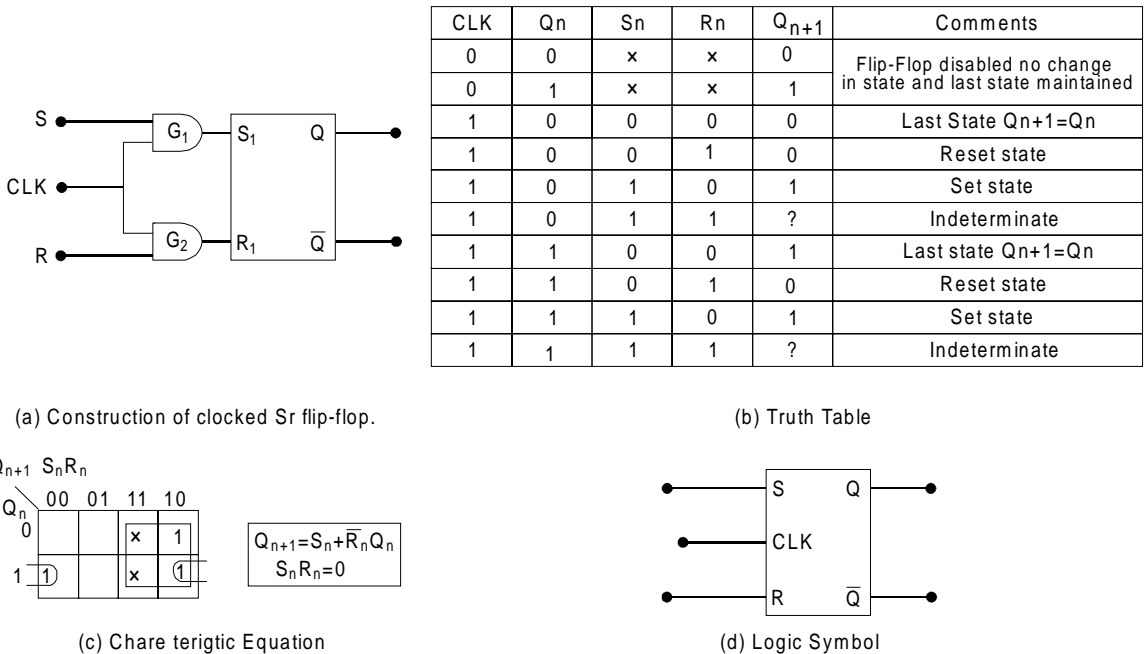


Fig. 6.16 Clocked RS (or SR) Flip-Flop

When clock (abbreviated as CLK) is LOW, outputs of gates  $G_1$  and  $G_2$  are forced to 0, which is applied to flip-flop inputs. Thus when  $CLK = 0$ ,  $S_1 = 0$  and  $R_1 = 0$ . Since both the inputs are LOW, flip-flop remain in its previous state. Alternatively if  $Q_n = 0$ ,  $Q_{n+1} = 0$  and if  $Q_n = 1$  we get  $Q_{n+1} = 1$ . Thus during  $t_{OFF}$  period inputs have no effect on the circuit. This is shown in first two rows of truth table given in Fig. 6.16 (b).

When clock is HIGH, gates  $G_1$  and  $G_2$  are transparent and signals S and R can reach to flip-flop inputs  $S_1$  and  $R_1$ . The next state will now be determined by the values of S and R. Thus during  $t_{ON}$  point  $CLK = 1$  causing  $S_1 = S$  and  $R_1 = R$  and the circuit behaves exactly same as the normal flip-flop discussed in subsection 6.11, as shown by rest of the rows of truth table.

Note that in truth table, inputs are labelled  $S_n, R_n$ . They represent the value of inputs during bit-time 'n' at the  $t_{ON}$  time of  $(n + 1)$ th pulse or at  $t = nT$  in Fig. 6.15. The output also is  $Q_{n+1}$  not simply Q. Because presence of clock pulses force us to consider two different instants of time : the time before application of pulse,  $Q_n$  (i.e. present state) and the time after the application of pulse,  $Q_{n+1}$  (i.e. next state).

Characteristic equation for this flip-flop is obtained from K-map shown in Fig. 6.16 (c), which is an algebraic expressions for the binary information of the truth table. This expression gives the value of next state as a function of present state and present inputs. The indeterminate conditions are marked "X"-don't care in the map because, depending upon the propagation delay of logic gates, state can be either 1 or 0. Inclusion of relation  $S_n R_n = 0$  as a part of characteristics equation is due to the requirement that both  $S_n$  and  $R_n$  must not be made 1 simultaneously.

Finally, the logic symbol of clocked SR flip-flop is shown in Fig. 6.16 (d), which now has three inputs named S, R and CLK.

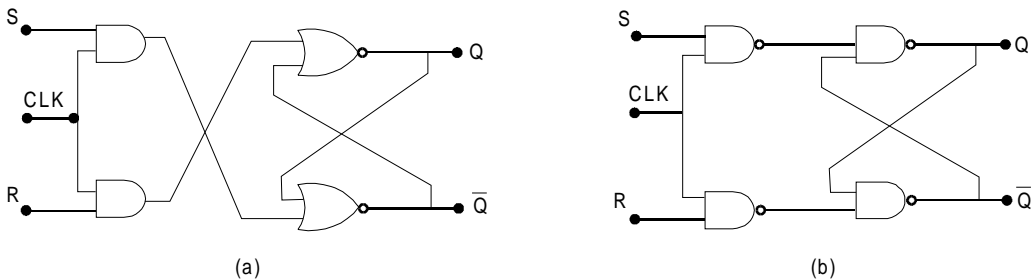


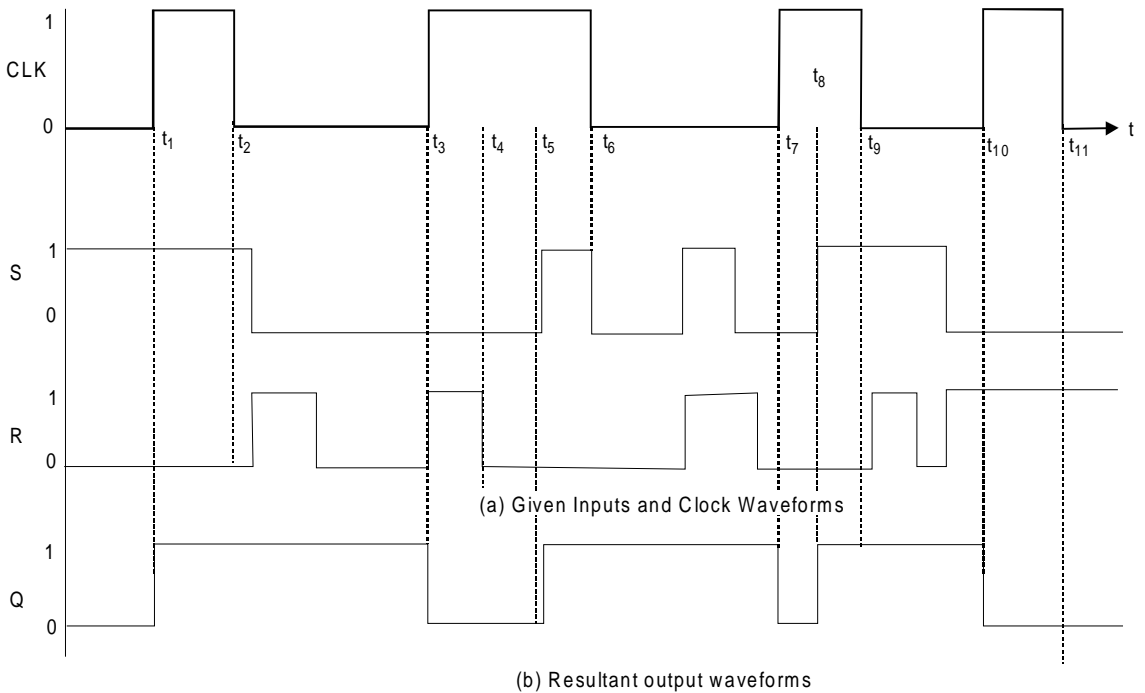
Fig. 6.17 Two Different Realizations of Clocked SR Flip-Flop.

Figure 6.17 shows two alternate realizations of clocked SR flip-flop. Both the realizations are popularly used in MSI and LSI (Medium and Large Scale Integrated) circuits. In many texts the signal CLOCK is also labelled ENABLE or EN.

**Example 6.3.** Fig. 6.18 (a) shows the input waveforms S, R and CLK, applied to clocked SR flip-flop. Obtain the output waveform Q and explain it in brief. Assume flip-flop is reset initially.

**Sol.** The resulting waveform at the output Q is shown in Fig. 6.18 (b). To understand the output waveform, recall that the inputs affect the flip-flop only when the clock = 1 otherwise flip-flop maintains its previous output irrespective of present input.

Initially at  $t = 0$  flip-flop is reset i.e.  $Q = 0$ . At this time  $S = 1$  and  $R = 0$ , but flip-flop remains unaffected since  $CLK = 0$ .



**Fig. 6.18** Waveforms for Example 6.3

At  $t_1$  clock goes HIGH and output also goes HIGH i.e.  $Q = 1$  since  $S = 1$  and  $R = 0$  at this time. At time  $t_2$   $CLK = 0$  and flip-flop remain SET until  $t_3$ , irrespective of changes in  $S$  and  $R$ .

At time  $t = t_3$   $CLK = 1$  and because  $S = 0$  and  $R = 1$  at this instant, we get  $Q = 0$ . At  $t_4$  both inputs are LOW while clock is still HIGH. Since  $S = R = 0$  at this instant flip-flop remains reset until just after time  $t_5$ .

Just after  $t_5$   $S$  goes HIGH making  $Q = 1$ . At time  $t_6$  clock switches to LOW state. Just before this HIGH to LOW transition of clock  $S = 1$  and  $R = 0$  and  $Q = 1$ . Thus flip-flop remains set until  $t_7$ . Changes in  $R$  and  $S$  does not affect flip-flop during  $t_6$  to  $t_7$  as  $CLK = 0$ .

At  $t = t_7$  clock goes HIGH, at which time  $R = 1$  and  $S = 0$ . So flip-flop enters in reset state.  $Q = 0$  is retained until  $t_8$  where  $R$  switches to 0 and  $S$  switches to 1. Therefore  $Q = 1$  at this time.

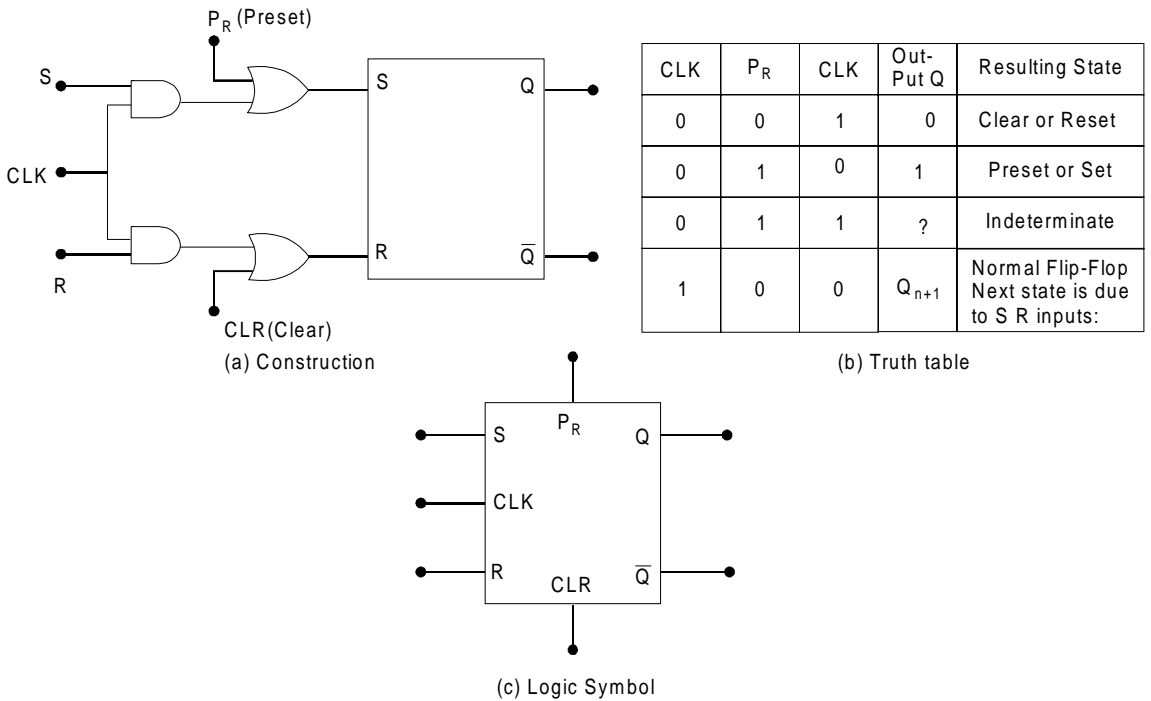
Clock goes LOW at  $t_9$  and since  $Q = 1$  just before clock goes LOW, flip-flop remains set until  $t_{10}$  where clock goes HIGH again. The inputs  $S$  and  $R$  changes during time between  $t_9$  to  $t_{10}$ , but can not affect the output since  $CLK = 0$ .

At  $t = t_{10}$  clock goes HIGH and since  $R = 1$  and  $S = 0$  at  $t_{10}$ , the flip-flop attains reset state. At the time  $t = t_{11}$  clock goes LOW and still  $R = 1$  and  $S = 0$  was maintained at the input, the flip-flop remains in LOW state beyond  $t_{11}$ .

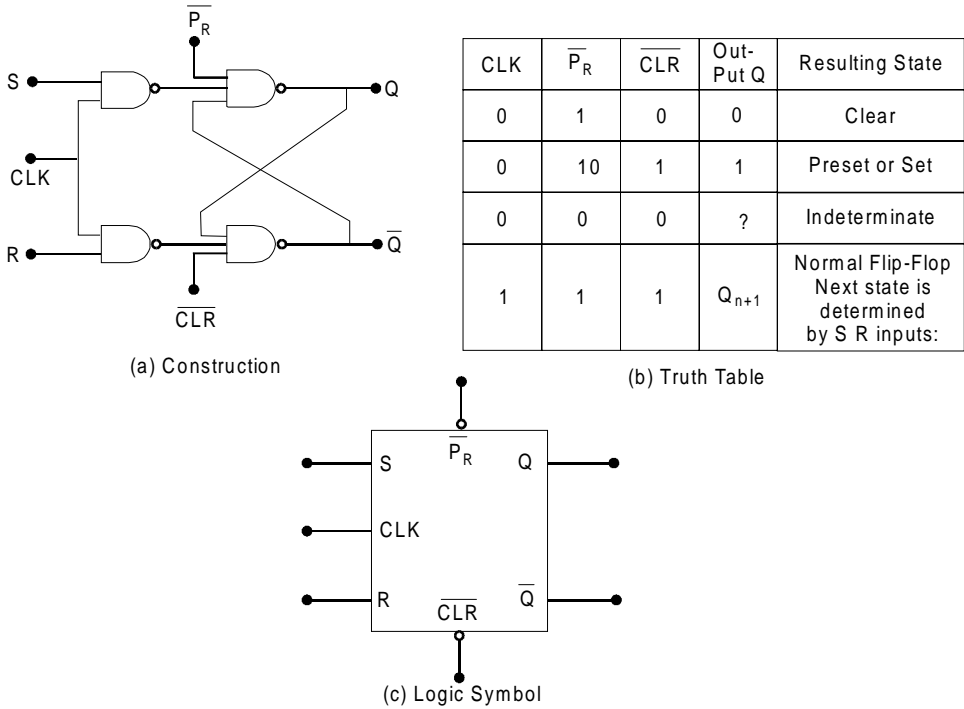
### Clocked SR Flip-flop with Clear and Preset

When the power is first applied to the flip-flop, it come up in random state i.e. state of circuit is uncertain. It may be in SET state or in RESET state. This is highly undesired in majority of application. There are requirements that the flip-flop must be in a particular state before the actual operation begins. In practice it may be required to preset ( $Q = 1$ ) or

clear ( $Q = 0$ ) the flip-flop to start the operation. In flip-flops such provisions can easily be provided for this purpose.



**Fig. 6.19** SR Flip-Flop with 'CLEAR' and 'PRESET'



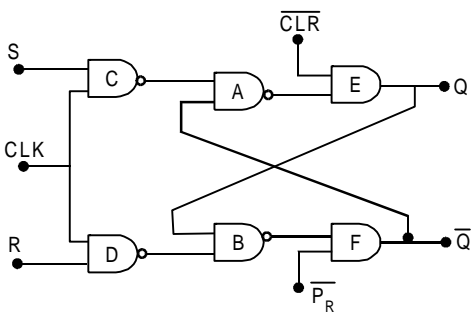
**Fig. 6.20** Alternate Realization of 'CLEAR' and 'PRESET' with SR Flip-Flop

Two different realizations to accommodate a preset (abbreviated as  $P_R$ ) and a clear (abbreviated as CLR) inputs are shown in Figs. 6.19 and 6.20. The  $P_R$  and CLR are direct inputs and are called **asynchronous inputs**; as they don't need the clock to operate. Both of the above circuits require that  $P_R$  and CLR inputs should be applied only in absence of clock otherwise unexpected behaviour can be observed. More over both  $P_R$  and CLR should not asserted at same time as it leads to indeterminate state. Logic values to be applied to preset and clear are accommodated in the truth tables. Before starting normal clocked operation the two direct inputs must be connected to a fix value as shown by the last entries of corresponding truth tables. The logic symbols for the two circuits are also shown in the figure.

In Fig. 6.19, due to the construction of circuit the flip-flop can be preset or clear on the application of Logic 1 at  $P_R$  or CLR respectively. In contrast, realization shown in fig. 6.20 demands a Logic 0 to be applied at the particular asynchronous input, in order to perform the intended operation. For normal operation these inputs must be connected to Logic 1, as indicated by the last row of truth table shown in Fig. 6.20 (b). Similarly in fig. 6.19 the  $P_R$  and CLR must be connected to Logic 0 to obtain normal flip-flop operation.

The circuits proposed in Figs. 6.19 and 6.20 requires that the  $P_R$  and CLR should only be applied when clock is LOW. This imposes a restriction to use these two signals. An improvement may be considered to remove this restriction.

Fig. 6.21 shows a clocked SR flip-flop with preset and clear ( $\overline{P_R}$  and  $\overline{CLR}$ ) inputs that can override the clock.  $\overline{P_R}$  and  $\overline{CLR}$  can be safely applied at any instant of time weather clock is present or not.



(a) Construction

CLK	$\overline{P_R}$	$\overline{CLR}$	Out- Put Q	Resulting State
x	1	0	0	Clear
x	0	1	1	Preset
x	0	0	?	Indeterminate
1	1	1	$Q_{n+1}$	Normal Flip-Flop Next state is determined by S and R Inputs.

(b) Truth Table

**Fig. 6.21** SR Flip-Flop with Clock Override 'Clear' and 'Preset'

As shown in figure two AND gates, E and F, are used to accommodate preset ( $\overline{P_R}$ ) and Clear ( $\overline{CLR}$ ) inputs. The truth table summaries the effect of these asynchronous inputs in the circuit. Output Q is provided through gate E whereas  $\overline{Q}$  is output of gate F. Both  $\overline{P_R}$  and  $\overline{CLR}$  are active LOW signals i.e. asserting Logic '0' at these inputs perform the intended operation.

According to first row of truth table when  $\overline{CLR} = 0$  and  $\overline{P_R} = 1$  is applied then irrespective of the value of S, R and CLK, output of gate E is 0. Thus  $Q = 0$  and it is feeded to the input of NAND gate B. So output of NAND-B is 1 which is applied to the input of gate F. Since we applied  $\overline{P_R} = 1$ , both the inputs to AND gate F are HIGH. Consequently the output of gate



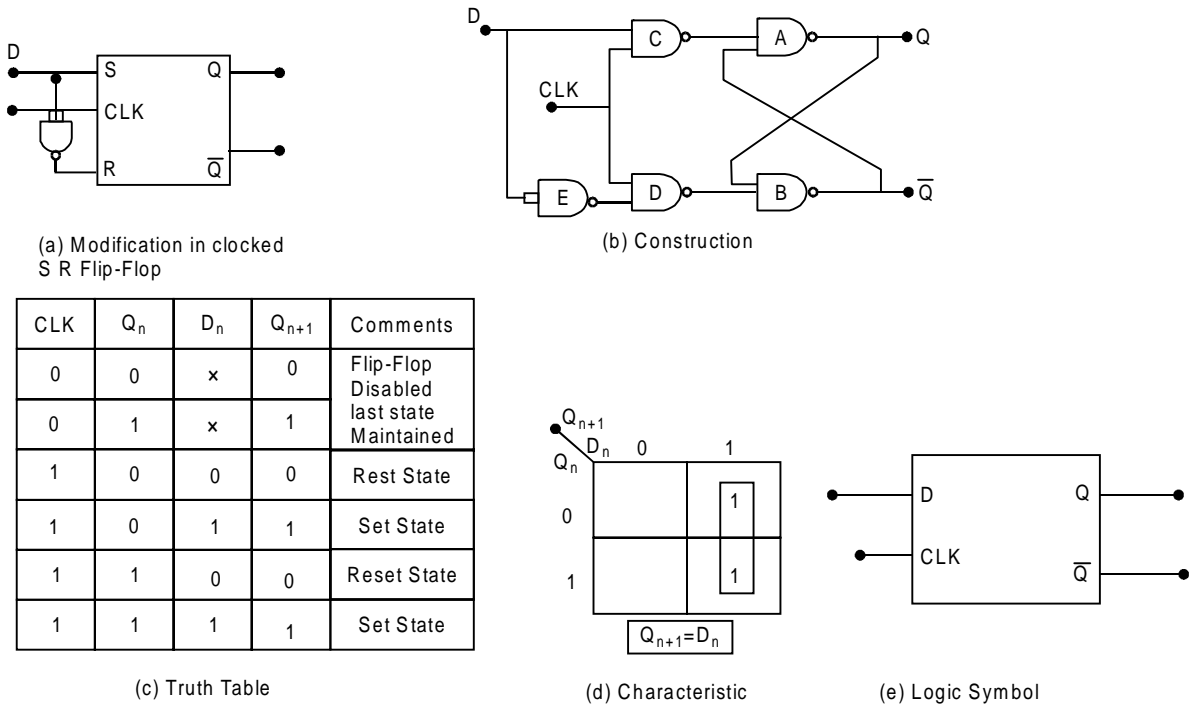
F goes HIGH, which is available at  $\bar{Q}$  output of flip-flop. Hence when  $\overline{PR} = 1$  and  $\overline{CLR} = 0$  is applied, we get  $Q = 0$ , and  $\bar{Q} = 1$ . This is frequently called as **clearing** the flip-flop.

Similarly when  $\overline{PR} = 0$  and  $\overline{CLR} = 1$  is applied, output of gate F, which is  $\bar{Q}$ , goes LOW i.e.  $\bar{Q} = 0$ . This forces the gate-A to give HIGH output. Since  $\overline{CLR} = 1$  is already present, output of gate E, which is  $Q$ , goes HIGH. Thus when  $\overline{PR} = 0$  and  $\overline{CLR} = 1$  is applied we get  $Q = 1$  and  $\bar{Q} = 0$ , which is required state. This is referred as presetting.

Since both  $\overline{PR}$  and  $\overline{CLR}$  are active LOW inputs, they must be connected to Logic 1, in order to obtain normal flip-flop operation as shown by fourth row of truth table. Also making both  $\overline{PR}$  and  $\overline{CLR}$  LOW is forbidden as it results in indeterminate state, for the similar reasons explained earlier.

### Clocked D Flip-Flop

In subsection 6.1.2 we obtained a D flip-flop by using an external inverter present at the input of SR latch as shown in Fig. 6.10 (a). In the similar way a clocked D flip-flop is obtained by using an external inverter at the input of clocked SR flip-flop. The clocked D flip-flop is shown below in Fig. 6.22. Note that unclocked RS latch of Fig. 6.10 (a) is replaced by a clocked RS flip-flop shown in Fig. 6.16 (d).



**Fig. 6.22** Clocked D Flip-Flop Equation

The characteristics equation is derived from K-map shown in Fig. 6.22 (d), which specifies that irrespective of previous state next state will be same as the data input. In truth table  $D_n$  represents the data input at the  $t_{ON}$  time of  $n$ th pulse.

The operation of clocked D flip-flop is same as explained in subsection 6.1.2, when  $CLK = 1$ . When  $CLK = 0$ , the D input has no effect on the flip-flop and the present state is maintained. This is evident by the first two rows of truth table. The Logic symbol of clocked D flip-flop is shown in Fig. 6.22 (e).

Similar to clocked SR flip-flops, the clocked D flip-flop may also be accommodated with the asynchronous inputs “preset” and “clear”. One particular realization is shown in Fig. 6.23. An alternative arrangement to obtain a D flip-flop, directly from SR flip-flop with clear and preset is shown in Fig. 6.24.

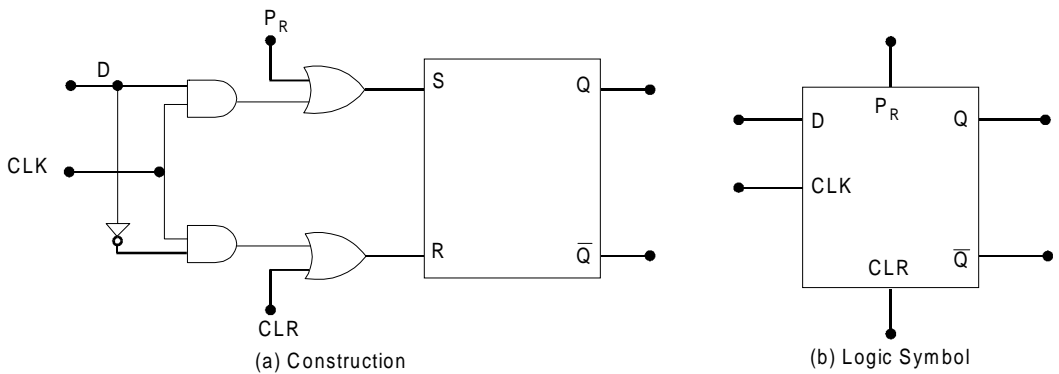


Fig. 6.23 D Flip-Flop with Clear and Preset

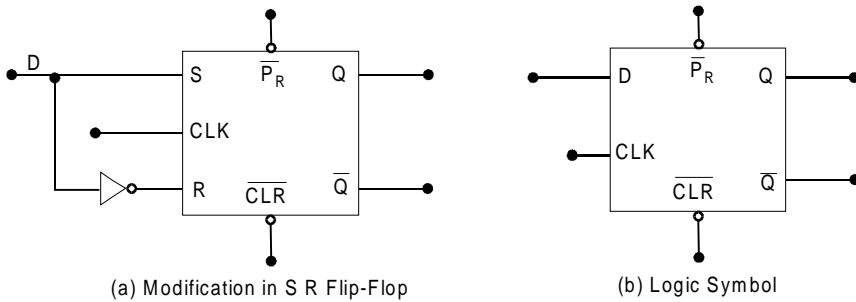


Fig. 6.24 Alternate Realization of Clear and Preset in D Flip-Flop

On comparing Fig. 6.23 (a) with Fig. 6.19 (a), we find that both the circuits are same except that, in two external inputs are connected together through an inverter placed between them. Thus both the circuits behave similarly and explanation of Fig. 6.19 (a) is equally valid in this case. Similarly, the arrangement shown in Fig. 6.24 (a) is built upon the clock SR flip-flop shown in Fig. 6.20 (c).

**Example 6.4.** In a certain digital application it is required to connect an SR flip-flop as toggle switch, which changes its state every time when clock pulse hits the system. Show the arrangement and explain in brief how it works as a toggle switch.

**Sol.** SR flip-flop can be connected as a toggle switch as shown in Fig. 6.25. On the arrival of CLOCK pulse this arrangement forces the flip-flop either to go to SET state if currently it is RESET or to RESET state if currently it is SET.

As shown, a feedback path connects the output Q to R input while another feedback path connects the  $\bar{Q}$  to input S. Recall that the state of flip-flop can be changed only when the  $CLK = 1$  and the last state reached, is maintained while  $CLK = 0$ .

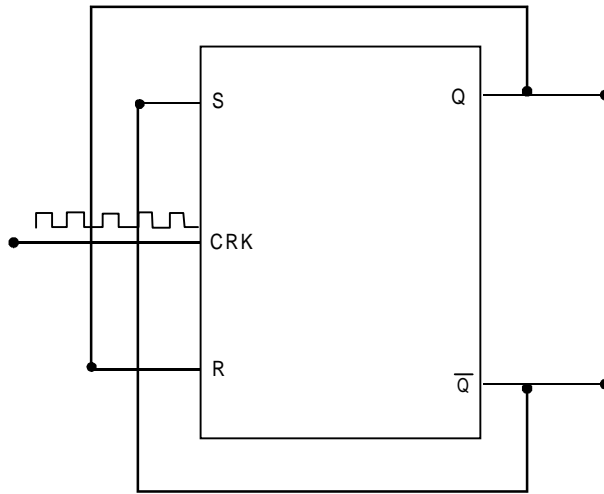


Fig. 6.25 SR Flip-Flop Connected as Toggle Switch

To start let the flip-flop is initially reset, i.e.  $Q = 0$  and  $\bar{Q} = 1$ . Same time due to the feedback, applied inputs are  $S = 1$  and  $R = 0$  because  $S = \bar{Q}$  and  $R = Q$ .

As soon as the clock goes HIGH flip-flop enters into SET state i.e.  $Q = 1$  and  $\bar{Q} = 0$ . Thus the inputs would be  $S = \bar{Q} = 0$  and  $R = Q = 1$  because of feedback path and remain unchanged until the next clock pulse arrives.

The moment clock goes HIGH again, flip-flop changes its state and attains RESET state where  $Q = 0$  and  $\bar{Q} = 1$ . Again through the feedback inputs become  $R = 0$  and  $S = 1$ . Note that this is the initial state we assumed in beginning. Thus after the arrival of two successive clock pulses, the switch has returned to its initial state  $Q = 0$  and  $\bar{Q} = 1$ .

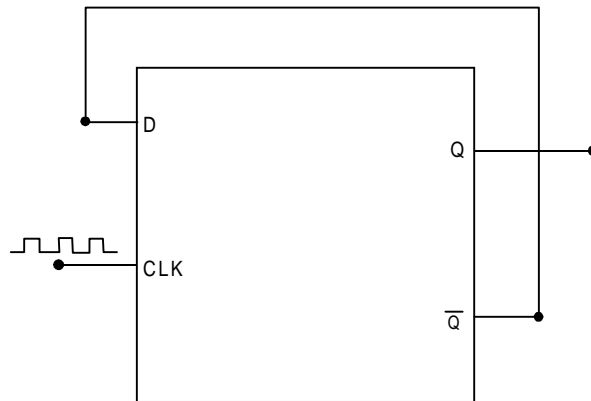
Therefore, the switch shown in Fig. 6.25 toggle between the two states with clock. Moreover the switch reproduces its state (either  $Q = 0$  or  $Q = 1$ ) after two successive clock pulses has arrived. This does mean that it really does not matter whether the initially assumed state is  $Q = 0$  or  $Q = 1$ .

In practice the feedback paths in Fig. 6.25 may lead to uncertainty of the state. In the above paragraphs we assumed that the inputs available at S and R do not change during the  $t_{ON}$  period of clock. Thus the change in state can occur only once in a single clock pulse, which is not true. If the propagation delay ( $t_p$ ) of the flip-flop is smaller than the  $t_{ON}$  time, multiple transitions (or state changes more than once) can be observed in a single clock pulse. Hence at the end of clock pulse the state of flip-flop is uncertain. This situation is referred as **race-around condition**.

Race around has resulted because the flip-flop remains transparent as long as  $CLK = 1$  (or for entire  $t_{ON}$  time). At this point we refrain ourselves from discussing it further. We address the complete discussion on it, only after examining and identifying the similar situations in various flip-flops.

**Example 6.5.** Realize a toggle switch, that changes its state on the arrival of clock, by using a D flip-flop. Explain its operation briefly.

**Sol.** Fig. 6.26 shows a D flip-flop configured to work as toggle switch when clock is applied.



**Fig. 6.26** D Flip-Flop as Toggle Switch

As shown the output  $\bar{Q}$  is connected to D via a feedback path, so that  $D = \bar{Q}$  always. To understand the operation recall that “in a D flip-flop output at any instant of time, provided  $CLK = 1$ , is same as input.” So  $Q = D$  always, if  $CLK = 1$ . Furthermore output  $\bar{Q} = \bar{D}$ , so through the feedback path complement of data D is now feeded to input. Thus if initially the flip-flop was reset ( $Q = 0$ ) the  $\bar{Q} = 1$  is applied to input through the feedback path. Consequently  $D = 1$  will be retained until next clock pulse arrive. As soon as  $CLK = 1$ ,  $D = 1$  affects the circuit. This results in change in state of flip-flop giving  $Q = D = 1$  and  $\bar{Q} = 0$  at the output. But at the same time  $\bar{Q}$  is feeded to D input due to which input changes to  $D = \bar{Q} = 0$ . On the arrival of next clock pulse the circuit toggles again and change its state in similar way.

It is evident again from Fig. 6.26 that even this switch also, suffers from the “race around” problem, explained in example 6.4. The problem is a consequence of the feedback path present between  $\bar{Q}$  and D. The output of this switch races for the same reasons as was given for SR flip-flop in example 6.4.

#### 6.1.4 Triggering of Flip Flops

By a momentarily change in the input signal the state of a flip-flop is switched. (0-1-0). This momentarily change in the input signal is called a trigger. There are two types by which flip-flops can be triggered.

Edge trigger

Level (pulse) trigger

An edge-triggered flip-flop responds only during a clock pulse transition i.e. clock pulses switches from one level (reference voltage) to another. In this type of flip-flops, output transitions occur at a specific level of the clock pulse. When the pulse input level exceeds this reference level, the inputs are locked out and flip-flop is therefore unresponsive to further changes in inputs until the clock pulse returns to 0 and another clock pulse occurs. An edge-triggered flip-flop changes states either at the positive edge (rising edge) or at the negative edge (falling edge) of the clock pulse on the control input.

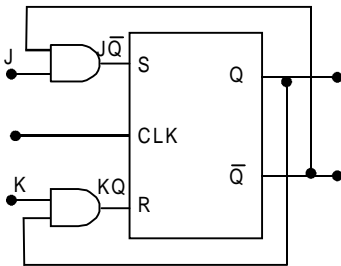
When the triggering occurs on the positive going edge of the clock, it is called positive edge triggering and when the triggering occurs at the trailing edge of the clock this is called as negative edge triggering.

The term pulse-triggered or level triggered means that data are entered into flip-flop on the rising edge of the clock pulse, but the output does not reflect the input state until the falling edge of the clock pulse. As this kind of flip-flops are sensitive to any change of the input levels during the clock pulse is still HIGH, the inputs must be set up prior to the clock pulse's rising edge and must not be changed before the falling edge. Otherwise, ambiguous results will happen.

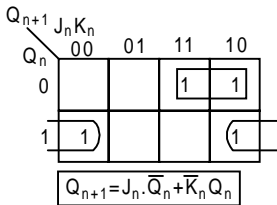
6.1.5 JK and T Flip-Flops

Problem of SR flip-flop to lead to indeterminate state when  $S = R = 1$ , is eliminated in JK flip-flops. A simple realization of JK flip-flop by modifying the SR type is shown in Fig. 6.27 (a). In JK the indeterminate state of SR flip-flop is now modified and is defined as TOGGLED STATE (i.e. its own complement state) when both the inputs are HIGH. Table shown in Fig. 6.27 (b), summarizes the operation of JK flip-flop for all types of input possibilities. Characteristic equation in Fig. 6.27 (c) is derived from K-Map, filled by the data provided by truth table.

Truth table shows the input applied from external world (J and K). It also shows the flip-flop inputs S and R and the whose values are due to the modification and in accordance with the values of J and K. Input signal J is for Set and K is for RESET. Both J and K are ANDED with  $\bar{Q}$  and Q respectively to generate appropriate signal for S and R. Since Q and  $\bar{Q}$  are always complementary, only one of the AND gates (Fig. 6.27 (a)) is enable at a time. So either only J or only K can reach to one of S and R inputs, thus any one of inputs will receive the data to be stored. While the other AND gate is disabled i.e. its output is 0, thus second input of SR flip-flop will always be 0. Thus indeterminate state never occurs even when both J and K inputs are HIGH.



(a) Realization of JK Flip-Flop from SR Flip-Flop

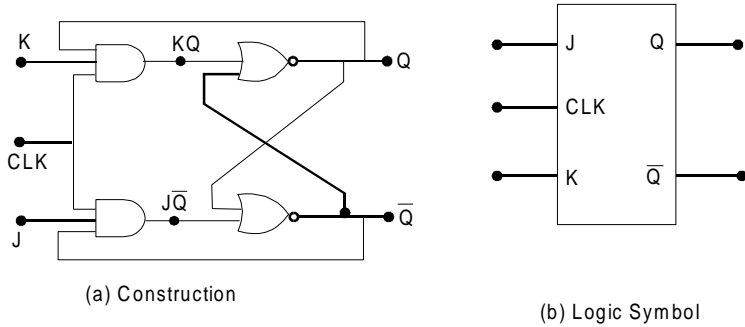


(c) Characteristic Equation

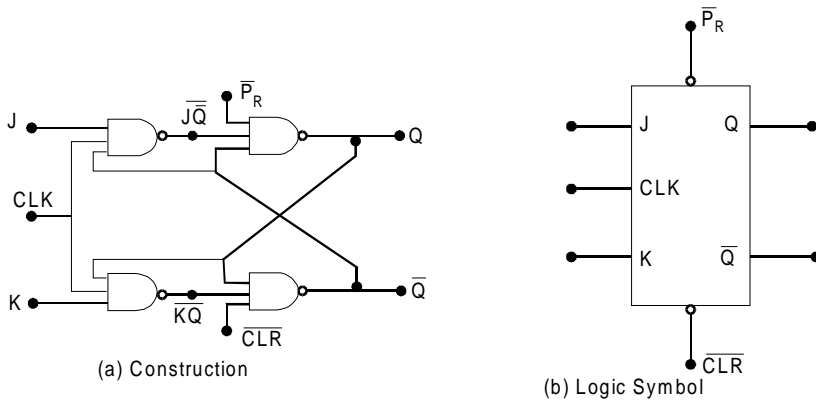
CLO CK	Present State		External Inputs		Flip-Flop Inputs		Output or Next state		Resulting state
	Q <sub>n</sub>	$\bar{Q}_n$	J <sub>n</sub>	K <sub>n</sub>	S <sub>n</sub>	R <sub>n</sub>	Q <sub>n+1</sub>		
0	0	1	x	x	0	0	0	Q <sub>n</sub>	Flip-Flop Disabled for CLK=0, no change
0	1	0	x	x	0	0	1		
1	0	1	0	0	0	0	0	Q <sub>n</sub>	No change at output. Present state becomes next state
1	1	0	0	0	0	0	1		
1	0	1	0	1	0	0	0	0	Reset state
1	1	0	0	1	0	1	0		
1	0	1	1	0	1	0	1	1	Set State
1	1	0	1	0	0	0	1		
1	0	1	1	1	1	0	1	$\bar{Q}_n$	Toggled state next state is complement of present state
1	1	0	1	1	0	1	0		

(b) Detailed truth table

Fig. 6.27 JK Flip-Flop Through SR Flip-Flop



**Fig. 6.28** Clocked JK Flip-Flop



**Fig. 6.29** JK Flip-Flop with Active LOW Preset and Clear

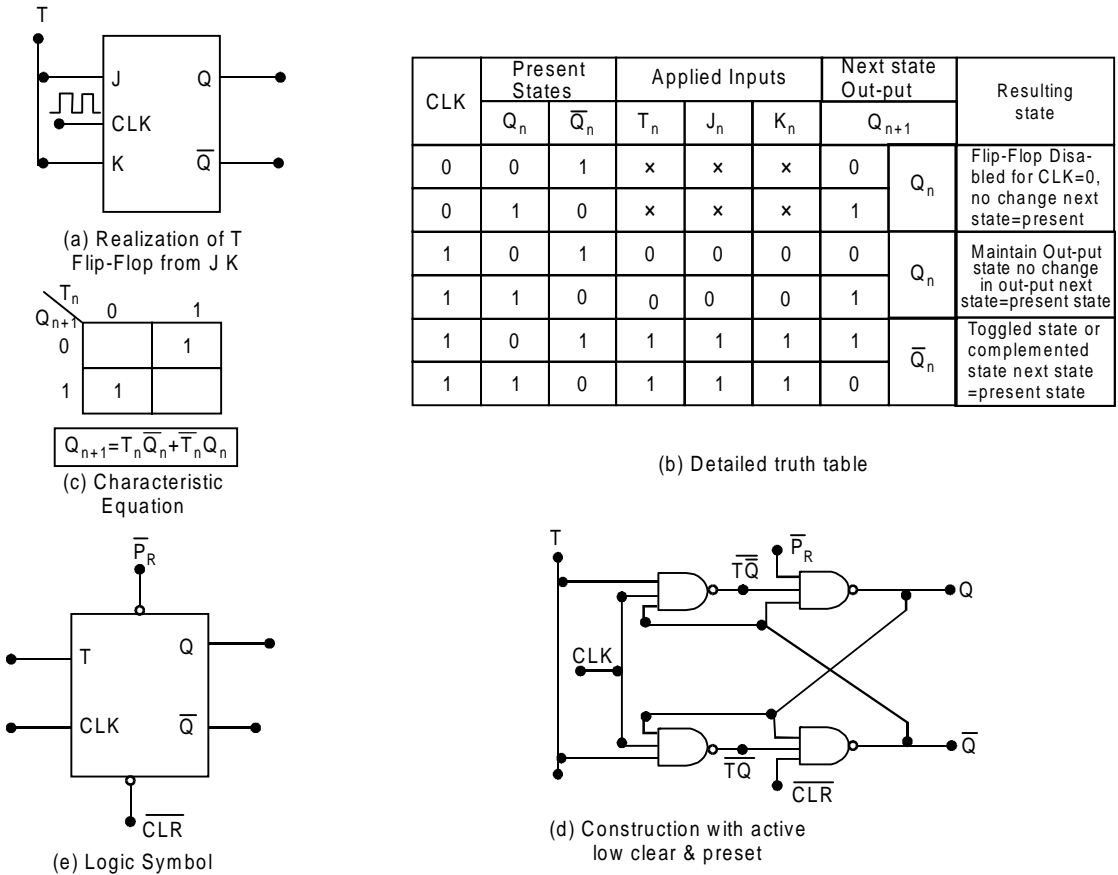
As an example assume that initially  $Q_n = 0$ ,  $\overline{Q}_n = 1$  and inputs applied are  $J = 1$  and  $K = 0$ . So we get  $S_n = J \cdot \overline{Q}_n = 1$  and  $R_n = K \cdot Q_n = 0$ . Application of  $S = 1$  and  $R = 0$ , when clock arrives, switches the state of flip-flop HIGH. Now if we assume  $Q_n = 1$ ,  $\overline{Q}_n = 0$  with same  $J$  and  $K$ , inputs result in  $S_n = J \cdot \overline{Q}_n = 0$  and  $R_n = K \cdot Q_n = 0$  applied to SR Flip-flop. When both the  $S$  and  $R$  inputs are LOW flip-flop does not under go any change of state. So  $Q_{n+1} = Q_n = 1$ . These two inputs are shown in 7th and 8th row of truth table. In the similar way entries in other rows of truth table can be verified.

Note that when both inputs are HIGH (i.e.  $J = 1$  and  $K = 1$ ) then, in Fig. 6.27 (a), we find that now  $S = J = Q$  and  $R = K = \overline{Q}$ . Thus it is evident that the two external AND gates become full transparent and circuit may be viewed as if  $\overline{Q}$  is connected to  $S$  input and  $Q$  connected to  $R$  input. Thus at  $J = K = 1$  flip-flop behaves as an SR toggle switch shown in Fig. 6.25.

**Note :** Asynchronous inputs  $\overline{P}_R$  and  $\overline{CLR}$  must not be activate when clock is present, other wise unexpected behaviour may be observed. Hence all the discussions presented in example 6.4 for toggle switch is equally valid for JK flip-flop when both  $J$  and  $K$  are HIGH.

Furthermore, when  $J = K = 1$  then due to the two feedback paths the output may start racing around the inputs causing multiple transitions. Thus “Race Around” condition may occur in JK flip-flop when both inputs are HIGH, for the similar reasons given in example 6.4.

Two gate level constructions for the JK flip-flop is shown in Fig. 6.28 (a) and 6.29 (a) along with their logic symbols shown in figure (b) in each figures.



**Fig. 6.30** Clocked T (Toggles) Flip-Flop with Active LOW Asynchronous Inputs

The JK flip-flops are very popular as indeterminate state (as present in SR type) does not exist. Furthermore, due to the toggling capability, when both inputs HIGH, on each arrival of pulse, it forms the basic element for counters. For this purpose JK flip-flop is further modified to provide a T flip-flop as shown in Fig. 6.30.

T flip-flop is a single input version of JK flip-flop, in which the inputs J and K are connected together, as shown, and is provided as a single input labelled as T. The operation is straight forward and easy, and summarized in truth-table given in Fig. 6.30 (b), while characteristic equation is derived in Fig. 6.30 (c).

When the clock is absent, the flip-flop is disable as usual and previously latched output is maintained at output. When the clock is present and  $T = 0$ , even though flip-flop is enabled the output does not switch its state. It happens so because for  $T = 0$  we get  $J = K = 0$  and thus next state is same as present state. Thus if either  $CLK = 0$  or  $T = 0$ , state does not change next state is always same as present state.

When  $T = 1$  during  $CLK = 1$ , it causes  $J = K = 1$  and as earlier discussed it will toggle the output state. Thus when input T is HIGH, flip-flop toggles its output on the arrival of clock, and for this reason input T is called as the **Toggle Input**. Essentially the T flip-flop also, suffer from race around condition, (when input is HIGH) and thus causing multiple transition at output due to same reasons given in example 6.4.

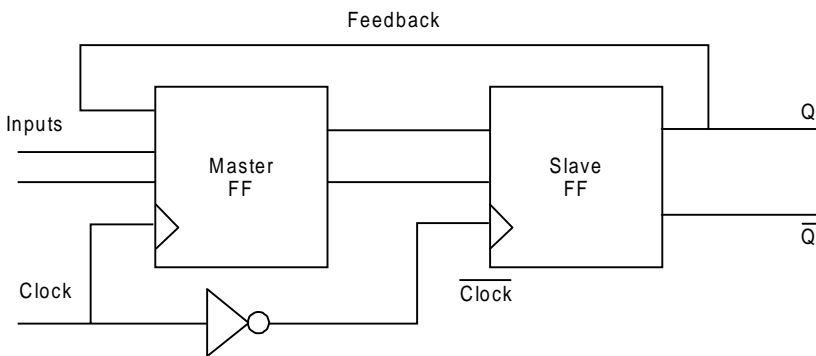
If the race around problem is somehow eliminated then T flip-flop can be used as frequency divider circuit. To obtain such operation input T is permanently made HIGH and the frequency to be divided is applied CLK inputs. At the outputs Q and  $\bar{Q}$  we receive a square wave whose time period is now doubled due to which frequency reduces to half of the input frequency. Note that Q and  $\bar{Q}$  generate square waves complementary to each other.

### 6.1.6 Race Around Condition and Solution

Whenever the width of the trigger pulse is greater than the propagation time of the flip-flop, then flip-flop continues to toggle 1-0-1-0 until the pulse turns 0. When the pulse turns 0, unpredictable output may result i.e. we don't know in what state the output is whether 0 or 1. This is called race around condition.

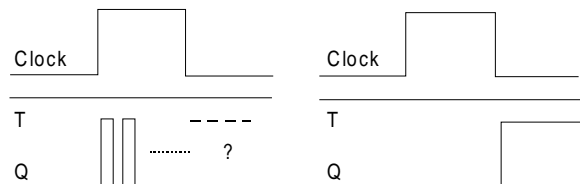
In level-triggered flip-flop circuits, the circuit is always active when the clock signal is high, and consequently unpredictable output may result. For example, during this active clock period, the output of a T-FF may toggle continuously. The output at the end of the active period is therefore unpredictable. To overcome this problem, *edge-triggered* circuits can be used whose output is determined by the edge, instead of the level, of the clock signal, for example, the rising (or trailing) edge.

Another way to resolve the problem is the Master-Slave circuit shown below:



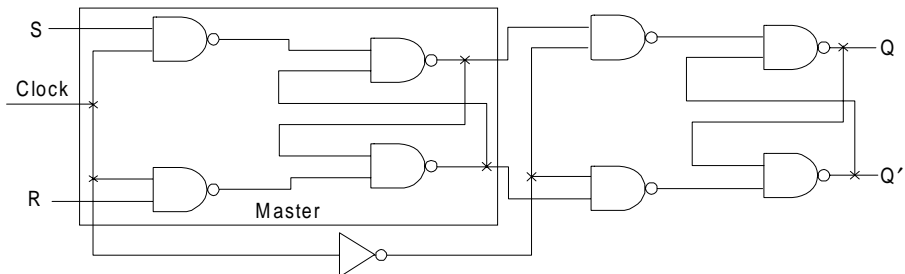
The operation of a Master-Slave FF has two phases:

- During the high period of the clock, the master FF is active, taking both inputs and feedback from the slave FF. The slave FF is de-activated during this period by the negation of the clock so that the new output of the master FF won't effect it.
- During the low period of the clock, the master FF is deactivated while the slave FF is active. The output of the master FF can now trigger the slave FF to properly set its output. However, this output will not effect the master FF through the feedback as it is not active.



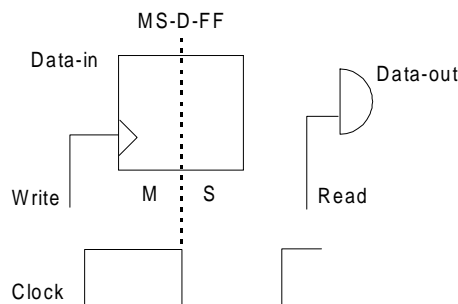


It is seen that the trailing edge of the clock signal will trigger the change of the output of the Master-Slave FF. The logic diagram for a basic master-slave S-R flip-flop is shown below.



Flip-flops are generally used for storing binary information. One bit of information can be written into a flip-flop, and later read out from it. If a master-slave FF is used, both read and write operations can take place during the same clock cycle under the control of two control signals **read** and **write**,

- During the first half of clock cycle : **clock = read = write = 1**, the old content in slave-FF is read out, while the new content is being written into master-FF at the same time.,
- During the second half of clock cycle : **clock = read = write = 0**, the new content in master-FF is written into slave-FF.



### 6.1.7 Operating Characteristics of Flip-flops

The operation characteristics specify the performance, operating requirements, and operating limitations of the circuits. The operation characteristics mentions here apply to all flip-flops regardless of the particular form of the circuit.

*Propagation Delay Time*—is the interval of time required after an input signal has been applied for the resulting output change to occur.

*Set-up Time*—is the minimum interval required for the logic levels to be maintained constantly on the inputs (J and K, or S and R, or D) prior to the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop.

*Hold Time*—is the minimum interval required for the logic levels to remain on the inputs after the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop.

*Maximum Clock Frequency*—is the highest rate that a flip-flop can be reliably triggered.

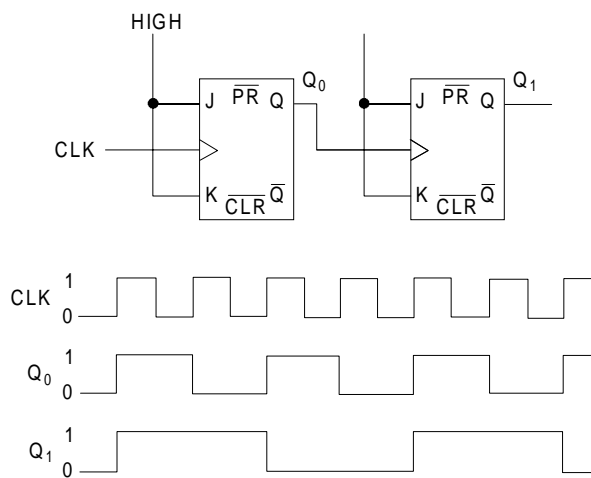
*Power Dissipation*—is the total power consumption of the device.

*Pulse Widths*—are the minimum pulse widths specified by the manufacturer for the Clock, SET and CLEAR inputs.

### 6.1.8 Flip-Flop Applications

#### Frequency Division

When a pulse waveform is applied to the clock input of a J-K flip-flop that is connected to toggle, the Q output is a square wave with half the frequency of the clock input. If more flip-flops are connected together as shown in the figure below, further division of the clock frequency can be achieved.

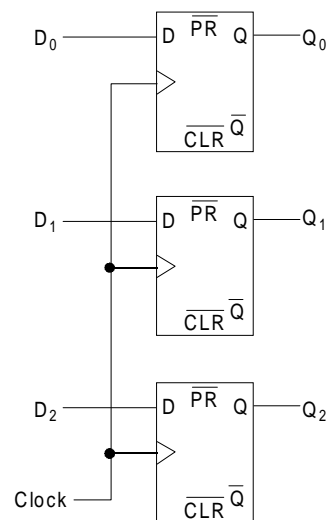


The Q output of the second flip-flop is one-fourth the frequency of the original clock input. This is because the frequency of the clock is divided by 2 by the first flip-flop, then divided by 2 again by the second flip-flop. If more flip-flops are connected this way, the frequency division would be 2 to the power  $n$ , where  $n$  is the number of flip-flops.

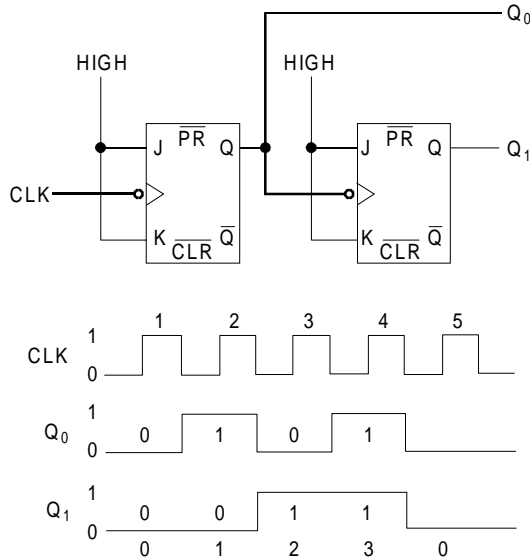
#### Parallel Data Storage

In digital systems, data are normally stored in groups of bits that represent numbers, codes, or other information. So, it is common to take several bits of data on parallel lines and store them simultaneously in a group of flip-flops. This operation is illustrated in the figure below.

Each of the three parallel data lines is connected to the D input of a flip-flop. Since all the clock inputs are connected to the same clock, the data on the D inputs are stored simultaneously by the flip-flops on the positive edge of the clock.



Another very important application of flip-flops is in digital counters, which are covered in detail in the chapter 7. A counter that counts from 0 to 2 is illustrated in the timing diagram given below. The two-bit binary sequence repeats every four clock pulses. When it counts to 3, it recycles back to 0 to begin the sequence again.



### 6.2 FLIP FLOP EXCITATION TABLE

The characteristic table is useful during the analysis of sequential circuits when the value of flip-flop inputs are known and we want to find the value of the flip-flop output Q after the rising edge of the clock signal. As with any other truth table, we can use the map method to derive the characteristic equation for each flip-flop.

During the design process we usually know the transition from present state to the next state and wish to find the flip-flop input conditions that will cause the required transition. For this reason we will need a table that lists the required inputs for a given change of state. Such a list is called the *excitation table*. There are four possible transitions from present state to the next state. The required input conditions are derived from the information available in the characteristic table. The symbol X in the table represents a “don’t care” condition, that is, it does not matter whether the input is 1 or 0.

The different types of flip flops (RS, JK, D, T) can also be described by their excitation, table as shown below. The left side shows the desired transition from  $Q_n$  to  $Q_{n+1}$ , the right side gives the triggering signals of various types of FFs needed for the transitions.

Desired transition		Triggering signal needed					
$Q_n$	$Q_{n+1}$	S	R	J	K	D	T
0	0	0	x	0	x	0	0
0	1	1	0	1	x	1	1
1	0	0	1	x	1	0	1
1	1	x	0	x	0	1	0

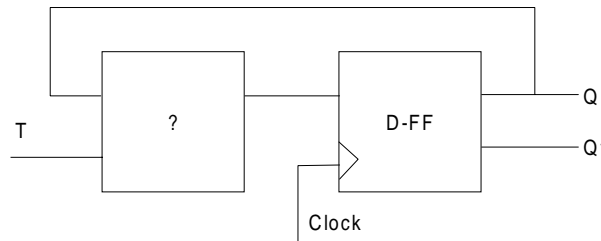
### 6.3 FLIP-FLOP CONVERSIONS

This section shows how to convert a given type A FF to a desired type B FF using some conversion logic.

The key here is to use the excitation table, which shows the necessary triggering signal (S, R, J, K, D and T) for a desired flip flop state transition  $Q_n \rightarrow Q_{n+1}$ .

$Q_n$	$Q_{n+1}$	S	R	J	K	D	T
0	0	0	x	0	x	0	0
0	1	1	0	1	x	1	1
1	0	0	1	x	1	0	1
1	1	x	0	x	0	1	0

**Example 1.** Convert a D-FF to a T-FF :



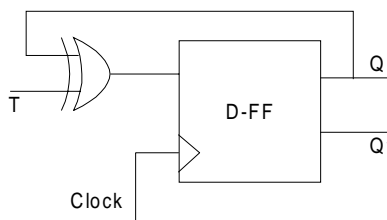
We need to design the circuit to generate the triggering signal D as a function of T and Q :  $D = f(T, Q)$

Consider the excitation table :

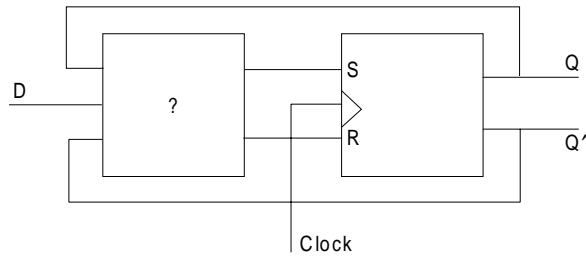
$Q_n$	$Q_{n+1}$	T	D
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

Treating D as a function of T and current FF state  $Q$   $Q_n$  we have

$$D = T\bar{Q} + TQ = T \oplus Q$$



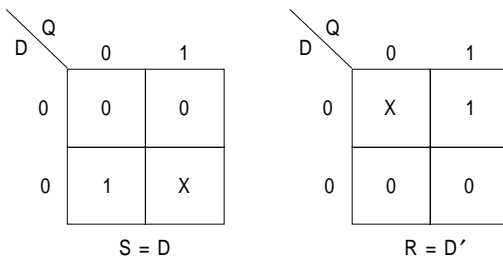
**Example 2.** Convert a RS-FF to a D-FF :



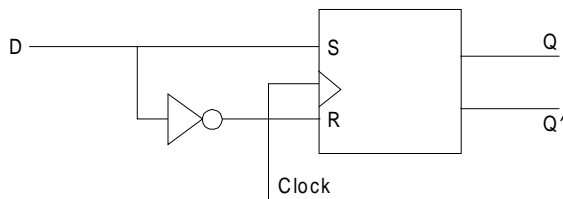
We need to design the circuit to generate the triggering signals S and R as functions of D and Q. Consider the excitation table :

$Q_n$	$Q_{n+1}$	$D$	$S$	$R$
0	0	0	0	X
0	1	1	1	0
1	0	0	0	1
1	1	1	X	0

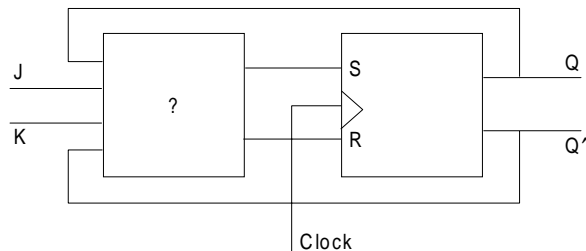
The desired signal S and R can be obtained as functions of T and current FF state Q from the Karnaugh maps :



$S = D, R = D'$



**Example 3.** Convert a RS-FF to a JK-FF.



We need to design the circuit to generate the triggering signals S and R as functions of J, K and Q. Consider the excitation table.

$Q_n$	$Q_{n+1}$	J	K	S	R
0	0	0	x	0	x
0	1	1	x	1	0
1	0	x	1	0	1
1	1	x	0	x	0

The desired signals S and R as function J, K and current FF state Q can be obtained from the Karnaugh maps:

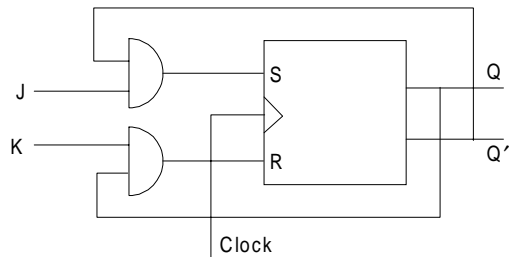
		QJ			
		00	01	11	10
K	0	0	1	X	X
	1	1	1	0	0

$S = Q'J$

		QJ			
		00	01	11	10
K	0	X	0	0	0
	1	X	0	1	1

$R = QK$

$$S = Q'J, \quad R = QK$$

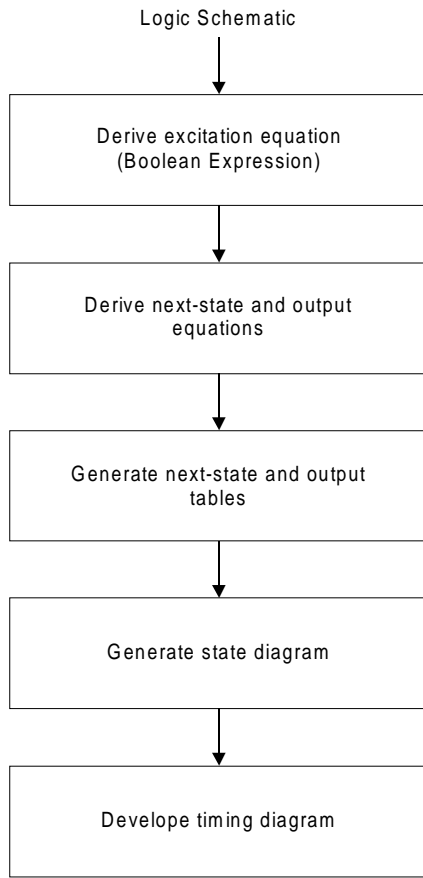


## 6.4 ANALYSIS OF CLOCKED SEQUENTIAL CIRCUITS

The behaviour of a sequential circuit is determined from the inputs, the outputs and the states of its flip-flops. Both the output and the next state are a function of the inputs and the present state.

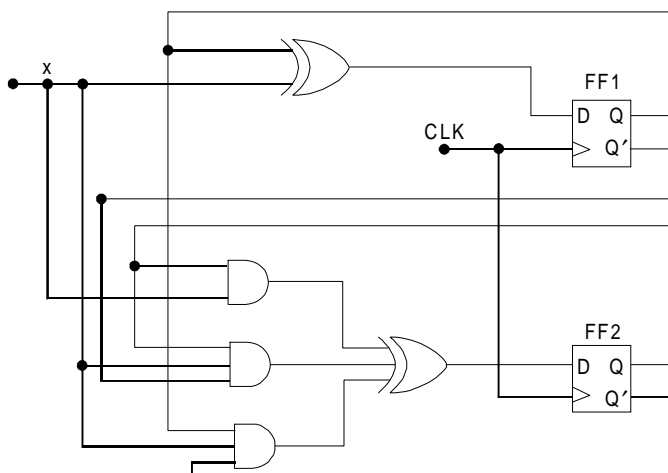
The suggested analysis procedure of a sequential circuit is set out below.

We start with the logic schematic from which we can derive excitation equations for each flip-flop input. Then, to obtain next-state equations, we insert the excitation equations into the characteristic equations. The output equations can be derived from the schematic, and once we have our output and next-state equations, we can generate the next-state and output tables as well as state diagrams. When we reach this stage, we use either the table or the state diagram to develop a timing diagram which can be verified through simulation.



**Example 1.** *Modulo-4 counter*

*Derive the state table and state diagram for the sequential circuit shown in Figure A.*



**Figure:** A Logic schematic of a sequential circuit.

**Solution.**

**Step 1 :** First we derive the Boolean expressions for the inputs of each flip-flops in the schematic, in terms of external input  $X$  and the flip-flop outputs  $Q_1$  and  $Q_0$ . Since there are two D flip-flops in this example, we derive two expressions for  $D_1$  and  $D_0$  :

$$D_0 = x \oplus Q_0 = x'Q_0 + xQ_0'$$

$$D_1 = x'Q_1 + xQ_1'Q_0 + xQ_1Q_0'$$

These Boolean expressions are called excitation equations since they represent the inputs to the flip-flops of the sequential circuit in the next clock cycle.

**Step 2 :** Derive the next-state equations by converting these excitation equations into flip-flop characteristic equations. In the case of D flip-flops,  $Q(\text{next}) = D$ . Therefore the next state equal the excitation equations.

$$Q_0(\text{next}) = D_0 = x'Q_0 + xQ_0'$$

$$Q_1(\text{next}) = D_1 = x'Q_1 + xQ_1'Q_0'$$

**Step 3 :** Now convert these next-state equations into tabular form called the next-state table.

Present State $Q_1Q_0$	Next State	
	$x = 0$	$x = 1$
0 0	0 0	0 1
	0 1	0 1
0 1	1 0	1 0
	1 0	1 1
1 0	1 1	1 1
	0 0	0 0

Each row is corresponding to a state of the sequential circuit and each column represents one set of input values. Since we have two flip-flops, the number of possible states is four - that is,  $Q_1Q_0$  can be equal to 00, 01, 10, or 11. These are present states as shown in the table.

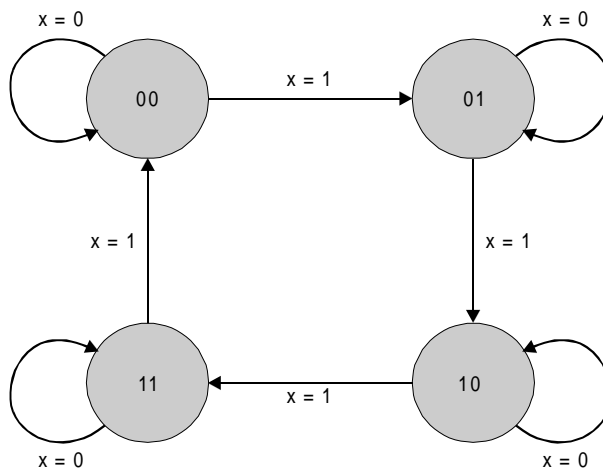
For the next state part of the table, each entry defines the value of the sequential circuit in the next clock cycle after the rising edge of the CLK. Since this value depends on the present state and the value of the input signals, the next state table will contain one column for each assignment of binary values to the input signals. In this example, since there is only one input signal, Cnt, the next-state table shown has only two columns, corresponding to  $x = 0$  and  $x = 1$ .



Note that each entry in the next-state table indicates the values of the flip-flops in the next state if their value in the present state is in the row header and the input values in the column header.

Each of these next-state values has been computed from the next-state equations in STEP 2.

**Step 4 :** The state diagram is generated directly from the next-state table, shown in Fig. B.



**Fig. B :** State diagram.

Each arc is labelled with the values of the input signals that cause the transition from the present state (the source of the arc) to the next state (the destination of the arc).

In general, the number of states in a next-state table or a state diagram will equal  $2^m$  where  $m$  is the number of flip-flops. Similarly, the number of arcs will equal  $2^m \times 2^k$ , where  $k$  is the number of binary input signals. Therefore, in the state diagram, there must be four states and eight transitions. Following these transition arcs, we can see that as long as  $x = 1$ , the sequential circuit goes through the states in the following sequence : 0, 1, 2, 3, 0, 1, 2, ..... On the other hand, when  $x = 0$ , the circuit stays in its present state until  $x$  changes to 1, at which the counting continues.

Since this sequence is characteristic of modulo-4 counting, we can conclude that the sequential circuit in Figure A is a modulo-4 counter with one control signal,  $x$ , which enables counting when  $X = 1$  and disables it when  $x = 0$ .

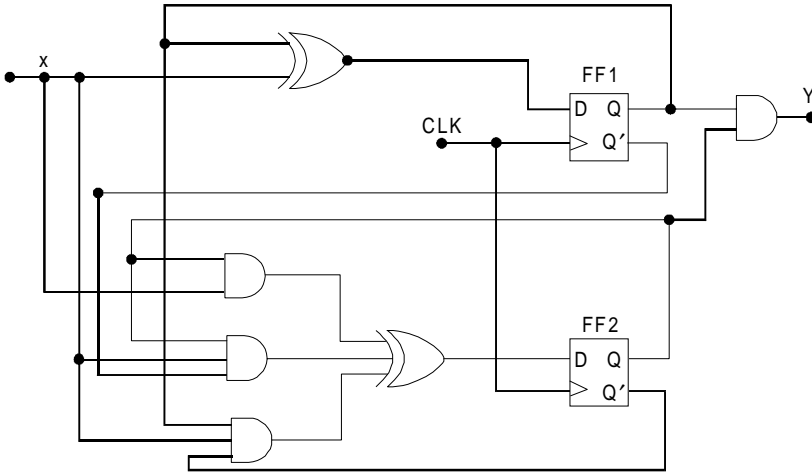
**Example 2.** Derive the next state, the output table and the state diagram for the sequential circuit shown in Fig. B.

**Solution.** The input combinational logic in Figure B is the same as in Example 1, so the excitation and the next-state equations will be same as in Example 1.

Excitation equations :

$$D_0 = x \oplus Q_0 + x'Q_0 + xQ_0'$$

$$D_1 = x'Q_1 + xQ_1'Q_0 + xQ_1Q_0'$$



**Figure B :** Logic schematic of a sequential circuit.

Next-state equations :

$$Q_0(\text{next}) = D_0 = x'Q_0 + xQ_0'$$

$$Q_1(\text{next}) = D_1 = x'Q_1 + xQ_1'Q_0 + xQ_1Q_0$$

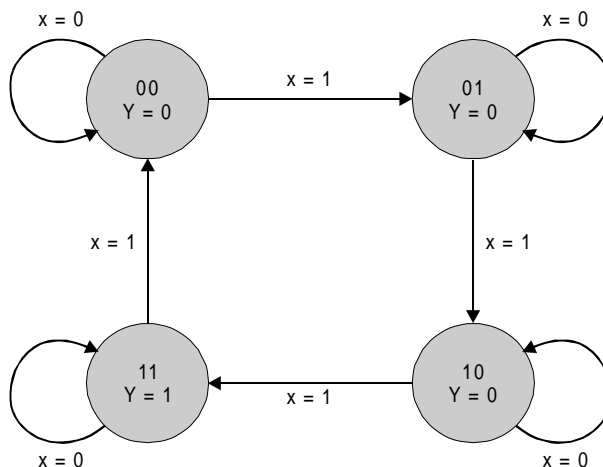
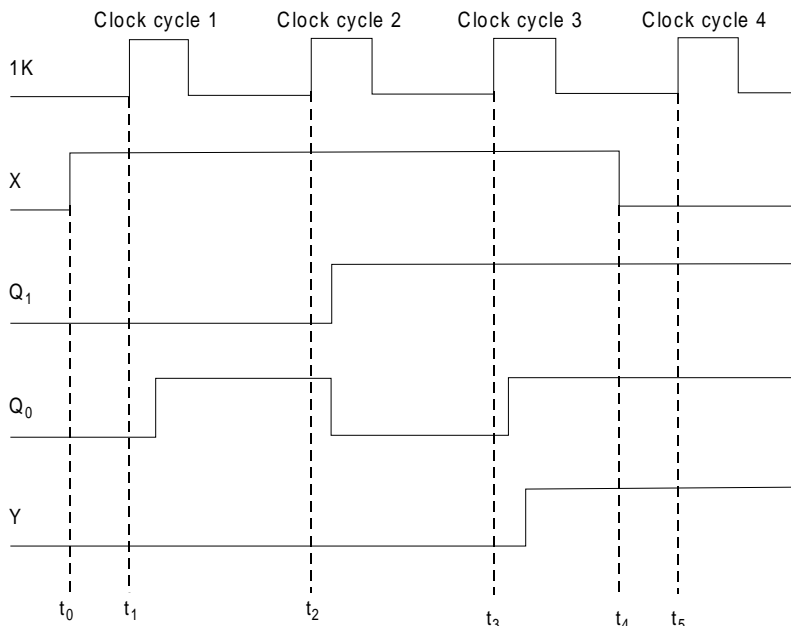
In addition, however, we have computed the output equation.

Output equation :  $Y = Q_1 Q_0$

As this equation shows, the output Y will equal to 1 when the counter is in state  $Q_1Q_0 = 11$ , and it will stay 1 as long as the counter stays in that state.

Next-state and output table :

Present State $Q_1Q_0$	Next State $x = 0$	Output $z$
00	00	0
	01	
01	01	0
	10	
10	10	0
	11	
11	11	1
	00	



**Fig.** State diagram of sequential circuit in Figure B.

**Timing diagram**

Figure C. Timing diagram of sequential circuit in Figure A.

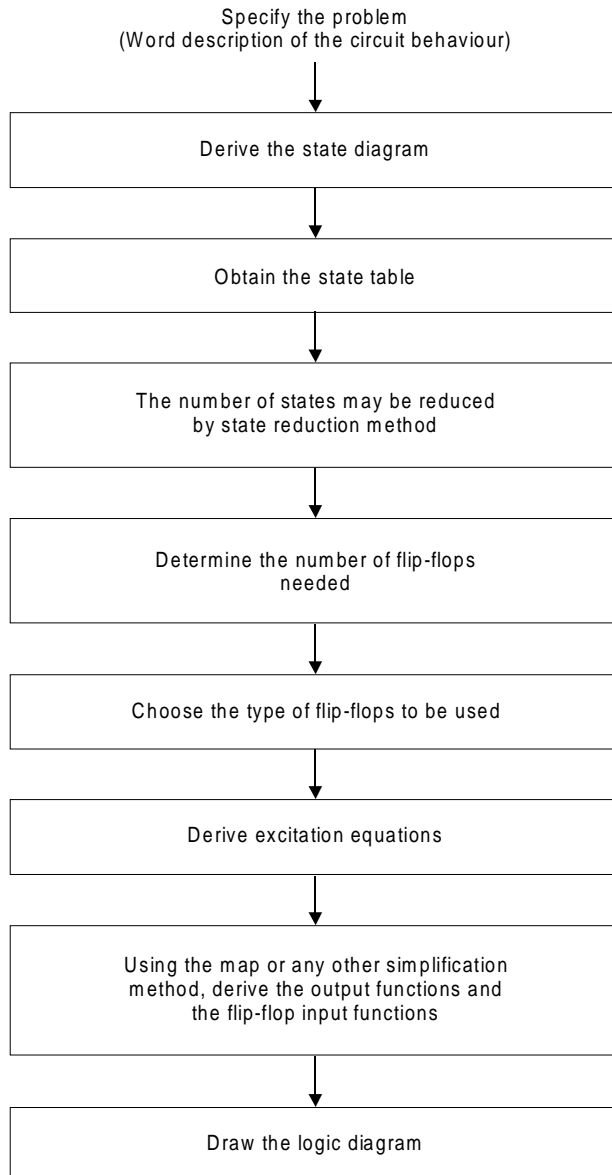
Note that the counter will reach the state  $Q_1Q_0 = 11$  only in the third clock cycle, so the output Y will equal 1 after  $Q_0$  changes to 1. Since counting is disabled in the third clock cycle, the counter will stay in the state  $Q_1Q_0 = 11$  and Y will stay asserted in all succeeding clock cycles until counting is enabled again.

**6.5 DESIGN OF CLOCKED SEQUENTIAL CIRCUITS**

The design of a synchronous sequential circuit starts from a set of specifications and culminates in a logic diagram or a list of Boolean functions from which a logic diagram can

be obtained. In contrast to a combinational logic, which is fully specified by a truth table, a sequential circuit requires a state table for its specification. The first step in the design of sequential circuits is to obtain a state table or an equivalence representation, such as a state diagram.

The recommended steps for the design of sequential circuits are set out below.



A synchronous sequential circuit is made up of flip-flops and combinational gates. The design of the circuit consists of choosing the flip-flops and then finding the combinational structure which, together with the flip-flops, produces a circuit that fulfils the required specifications. The number of flip-flops is determined from the number of states needed in the circuit.

### State Table

The state table representation of a sequential circuit consists of three sections labelled *present state*, *next state* and *output*. The present state designates the state of flip-flops before the occurrence of a clock pulse. The next state shows the states of flip-flops after the clock pulse, and the output section lists the value of the output variables during the present state.

### State Diagram

In addition to graphical symbols, tables or equations, flip-flops can also be represented graphically by a state diagram. In this diagram, a state is represented by a circle, and the transition between states is indicated by directed lines (or arcs) connecting the circles.

### State Diagrams of Various Flip-Flops

Table below shows the state diagrams of the four types of flip-flops.

Name	State Diagram
SR	<p>State diagram for SR flip-flop showing two states: <math>Q = 0</math> and <math>Q = 1</math>. Transitions are labeled with <math>S, R</math> values: <math>S, R = 0, 0</math> (self-loops), <math>S, R = 1, 0</math> (<math>Q = 0 \rightarrow Q = 1</math>), and <math>S, R = 0, 1</math> (<math>Q = 1 \rightarrow Q = 0</math>).</p>
JK	<p>State diagram for JK flip-flop showing two states: <math>Q = 0</math> and <math>Q = 1</math>. Transitions are labeled with <math>J, K</math> values: <math>J, K = 0, 0</math> (self-loops), <math>J, K = 1, 0</math> or <math>0, 1</math> (<math>Q = 0 \rightarrow Q = 1</math>), and <math>J, K = 0, 1</math> or <math>1, 1</math> (<math>Q = 1 \rightarrow Q = 0</math>).</p>
D	<p>State diagram for D flip-flop showing two states: <math>Q = 0</math> and <math>Q = 1</math>. Transitions are labeled with <math>D</math> values: <math>D = 1</math> (self-loops and <math>Q = 0 \rightarrow Q = 1</math>), and <math>D = 0</math> (<math>Q = 1 \rightarrow Q = 0</math>).</p>
T	<p>State diagram for T flip-flop showing two states: <math>Q = 0</math> and <math>Q = 1</math>. Transitions are labeled with <math>T</math> values: <math>T = 0</math> (self-loops), and <math>T = 1</math> (<math>Q = 0 \rightarrow Q = 1</math> and <math>Q = 1 \rightarrow Q = 0</math>).</p>

One can see from the table that all four flip-flops have the same number of states and transitions. Each flip-flop is in the set state when  $Q = 1$  and in the reset state when

$Q = 0$ . Also, each flip-flop can move from one state to another, or it can re-enter the same state. The only difference between the four types lies in the values of input signals that cause these transitions.

A state diagram is a very convenient way to visualize the operation of a flip-flop or even of large sequential components.

### State Reduction

Any design process must consider the problem of minimizing the cost of the final circuit. The two most obvious cost reductions are reductions in the number of flip-flops and the number of gates.

The number of states in a sequential circuit is closely related to the complexity of the resulting circuit. It is therefore desirable to know when two or more states are equivalent in all aspects. The process of eliminating the equivalent or redundant states from a state table/diagram is known as *state reduction*.

**Example.** Let us consider the state table of a sequential circuit shown in Table A.

**Table A. State table**

<i>Present State</i>	<i>Next State</i>		<i>Output</i>	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
A	B	C	1	0
	F	F	0	0
B	D	D	1	1
	E	E	1	1
C	F	F	1	0
	F	F	0	0
D	E	E	1	1
	A	A	0	0
E	D	D	0	0
	B	B	1	1
F	C	C	0	0
	C	C	0	0

It can be seen from the table that the present state A and F both have the same next states, B (when  $x = 0$ ) and C (when  $x = 1$ ). They also produce the same output 1 (when  $x = 0$ ) and 0 (when  $x = 1$ ). Therefore states A and F are equivalent. Thus one of the states, A or F can be removed from the state table. For example, if we remove row F from the table and replace all F's by A's in the columns, the state table is modified as shown in Table B.

**Table B. State F removed**

<i>Present State</i>	<i>Next State</i>		<i>Output</i>	
	<i>x = 0</i>	<i>x = 1</i>	<i>x = 0</i>	<i>x = 1</i>
A	B		1	
	C		0	
B	A		0	
	D		0	
C	D		1	
	E		1	
D	A		0	
	E		1	
E	A		0	
	D		0	

It is apparent that states B and E are equivalent. Removing E and replacing E's by B's results in the reduce table shown in Table C.

**Table C. Reduced state table**

<i>Present State</i>	<i>Next State</i>		<i>Output</i>	
	<i>x = 0</i>	<i>x = 1</i>	<i>x = 0</i>	<i>x = 1</i>
A	B		1	
	C		0	
B	A		0	
	D		0	
C	D		1	
	B		1	
D	A		0	
	B		1	

The removal of equivalent states has reduced the number of states in the circuit from six to four. Two states are considered to be *equivalent* if and only if for every input sequence the circuit produces the same output sequence irrespective of which one of the two states is the starting state.

**6.6 FINITE STATE MACHINE (FSM)**

**Definition:** A typical sequential system composed of

- Inputs from outside world;

- Internal *state* of the system stored in the memory elements;
- *Outputs* to outside world;
- *Next state decoder*;
- *Output decoder*.

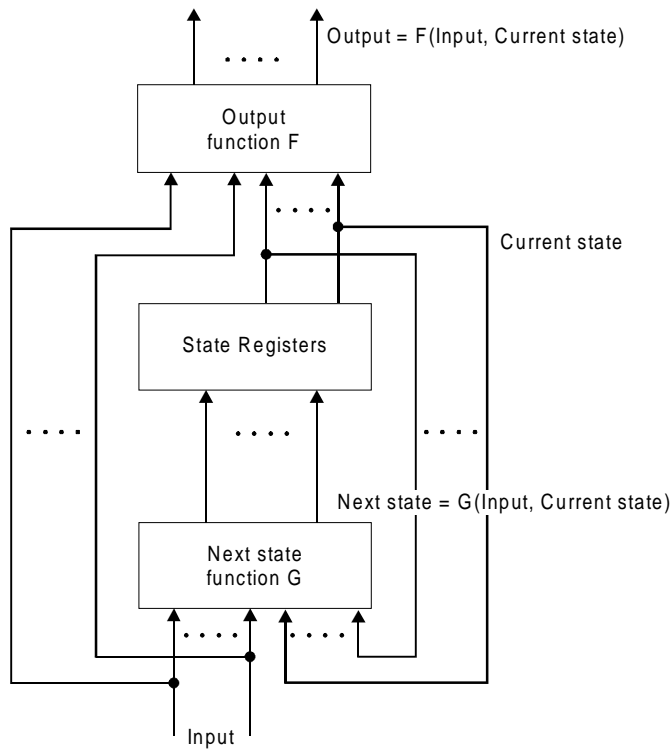
Both the next state and the output are functions of the input and the current state :

$$\text{Next State} = G(\text{Input}, \text{Current State})$$

$$\text{Output} = F(\text{Input}, \text{Current State})$$

Such a system is also called a *Mealy Machine*, or Class A machine. Finite state machine has different variations.

### A General Modal of FSM—Mealy (Class A) Machine



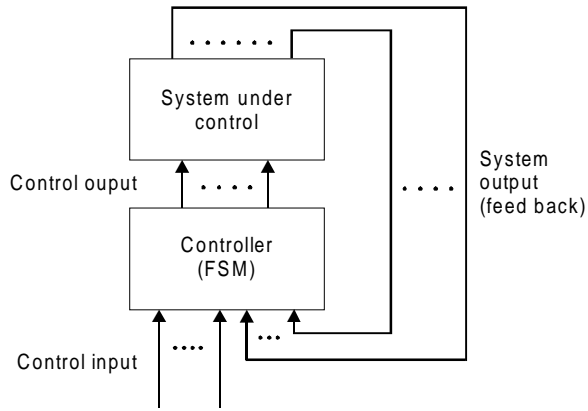
Finite state machine can be designed to control processes of *digital* nature (discrete in time, binary in variable values) which can be described by Boolean algebra. This is comparable with but different from the PID controllers which are used to control processes of *analog* nature (continuous in both time and variable values) described by differential equations.

### Finite State Machine (FSM) Used as a Controller

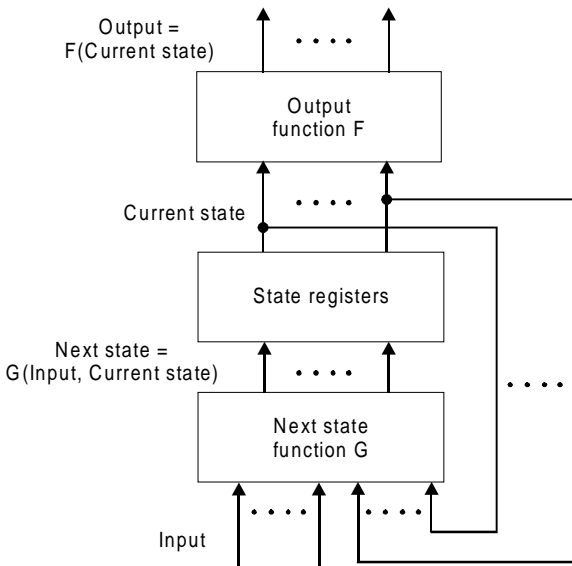
With the help of following steps one can design an FSM for solving a given problem :

1. Understand the problem and determine the **number of states** needed. If there are  $N$  states, then at least  $\log_2 N$  flip-flop's are needed to represent these states. This is the most important step!

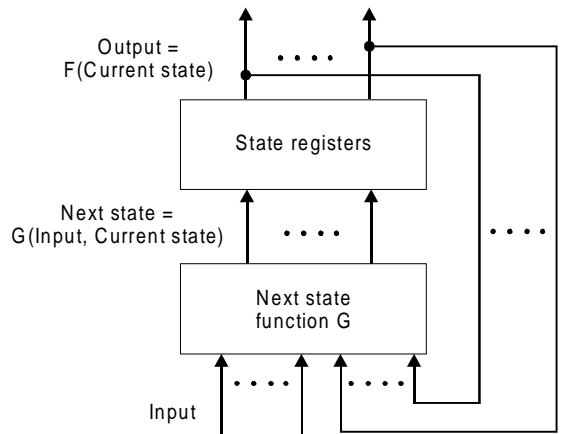




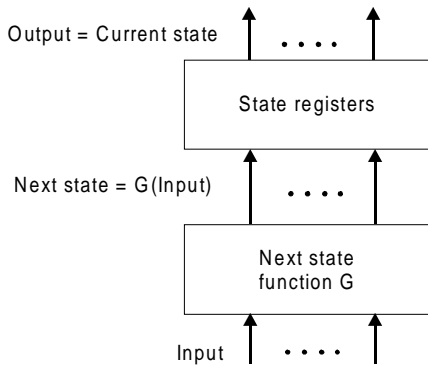
**Moore (Class B) Machine**



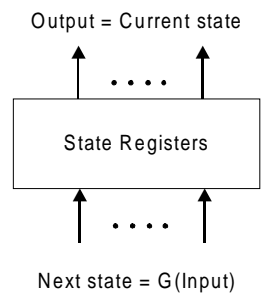
**Class C Machine**



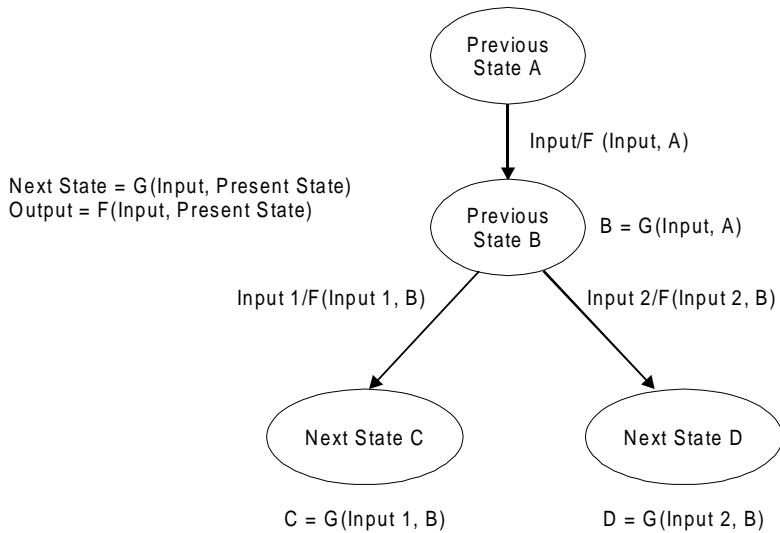
**Class D Machine**



**Class E Machine**



2. Draw the **state diagram**. At each state, find out where to go as the next state and what output to generate each combination of inputs.
3. Make the **state table** based on the state diagram. The current state (represented by the  $Q(t)$ 's of the FFs) and inputs are listed on the left as arguments, while the corresponding next state (represented by  $Q(t+1)$ 's of the FFs) and outputs are listed on the right as the functions.
4. Design the **next state decoder** and the **output decoder** using the state table as the truth table. If D-FFs are used, then  $Q(t+1)$  of the  $i$ th FF can be used directly as the signal  $D_i$  to set the FF. However, when other types of FFs are to be used, the excitation table is helpful to figure out the signals (S, R, J, K, or T) needed to realize the desired state transition :  $Q(t) \rightarrow Q(t+1)$  for each of the FFs.
5. **Simplify** the functions by K-map, and **implement** the next state and output decoders at logic gate level.

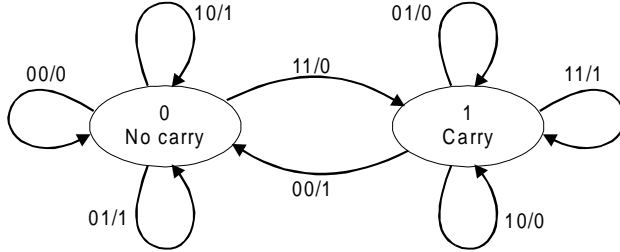


**Note:** Many of the problems of interest to us only require More or class B machine (the outputs are functions of the state only) or class C machine (the outputs are the same as the state). In these cases, the outputs can be generated as functions of the new state after the transition is completed.

**Example 1.** A serial adder receives two operands  $A = a_{n-1}, \dots, a_1 \dots a_0$   $B = b_{n-1}, \dots, b_1 \dots b_0$  as two sequences of bits ( $i = 0, 1, \dots, n - 1$ ) and adds them one bit at a time to generate the sequence of bits  $S_i$  of the sum as the output. Implement this serial adder as a finite state machine.

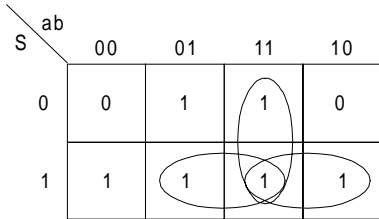
- \* Inputs :  $a_i$  and  $b_i$
- \* Output :  $S_i$
- \* Two states : carry  $S = 1$ , or no carry  $S = 0$

\* State diagram:

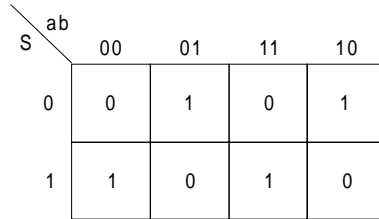


\* State table:

Present State $S$	Inputs		Next State $S'$	Output $S_i$
	$a_i$	$b_i$		
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



$$S' = ab + aS + bS$$

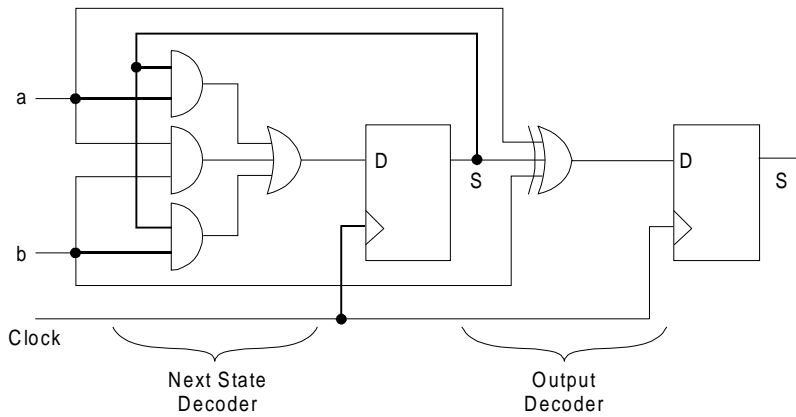


$$S_i = a \oplus b \oplus S$$

- Next state decoder:  $S' = G(a_i, b_i, S) = a_i b_i + a_i S + b_i S$
- Output decoder:  $S_i = F(a_i, b_i, S) = a_i \oplus b_i \oplus S$

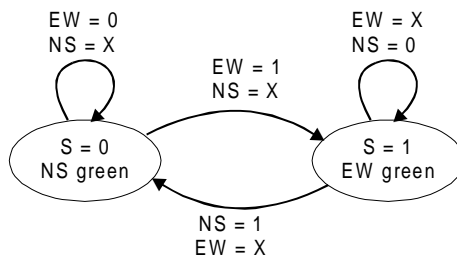
The FSM implementation of the serial adder contains three pieces of hardware: (i) a D-FF for keeping the state (whether or not there is a carry from the  $i$ th bit to the  $(i+1)$ th bit), (ii) the next state decoder  $S = a_i b_i + a_i S + b_i S$  that sets the D-FF, and (iii) the output decoder that generates the sum bit  $s_i = a_i \oplus b_i \oplus S$ . Note that a MS-FF is used for the output so that the output is a function of the current state and input, and it will stay unchanged after the state transition (from current to next state).

**Example 2.** Design the FSM controller for the traffic lights at an intersection (North/South (NS) vs. East/West (EW) with green and red lights only. The rule : (a) if no car detected, stay the same state, (b) if cars are detected in the direction with red light (independent of whether cars are detected in the direction with green light), switch state.



- States :
- S = 0: NS green (EW red);
- S = 1: EW green (NS red).
- Inputs :
- NS = 1/0: NS car detected/not detected
- EW = 1/0: EW car detected/not detected
- output: same as states (a class C FSM).

The state diagram :



The state table:

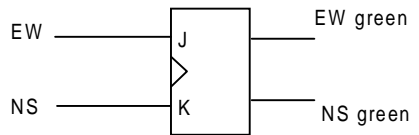
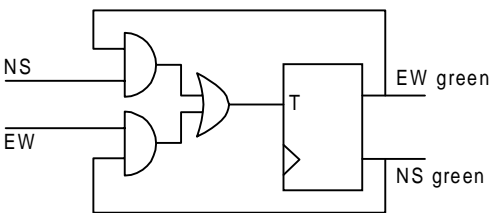
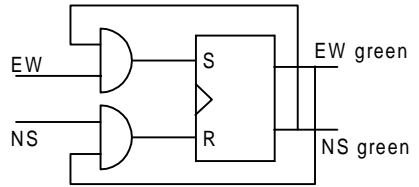
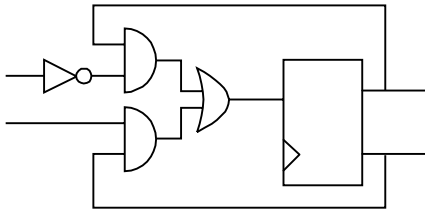
Present State (PS)	Inputs'		Next State	Signals to trigger the FF					
	NS	EW		D	S	R	J	K	T
0	x	0	0	0	0	x	0	0	0
0	x	1	1	1	1	0	1	x	1
1	0	x	1	1	x	0	x	0	0
1	1	x	0	0	0	1	x	1	1

The next state decoder can be implemented in any of the four types of flip flops. Given the desired state transition (from present state to next state), the signals needed to trigger the chosen FF can be obtained by the excitation table (also shown in the state table), to be generated by the next state decoder. Note that if D-FF is used, the triggering signal is the same as the desired next state.

- D-FF :  $D = \overline{PS}.EW + PS.\overline{NS}$
- RS-FF :  $S = \overline{PS}.EW, R = PS.NS$

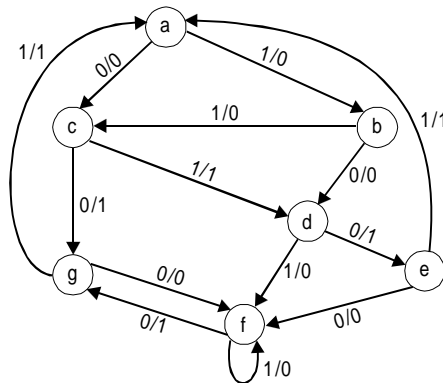
• JK-FF :  $J = EW, K = NS$

• T-FF :  $T = \overline{PS}.EW + PS.NS$



6.7 SOLVED EXAMPLES

**Example 1.** For the state diagram shown in Fig. 6. Write state table & reduced state table.



**Solution.** From the state diagram, a state table is prepared as shown in table 6.

Present State	Next state		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	c	b	0	0
b	d	c	0	0
c	g	d	1	1
d	e	f	1	0
e	f	a	0	1
f	g	f	1	0
g	f	a	0	1

It has been shown from the table 6 that the present state  $e$  and  $g$  both have the same next states  $f$  (when  $x = 0$ ) and  $a$  (when  $x = 1$ ). They also produce the same output 0 (when  $x = 0$ ) and 1 (when  $x = 1$ ). Therefore states  $e$  and  $g$  are equivalent. Thus one of the states,  $e$  or  $g$  can be removed from the state table. For example, if we remove row  $g$  from the table and replace all  $g$ 's by  $e$ 's in the columns, the state table is modified as shown in table 6.

**Table 6. State  $g$  removed**

Present State	Next state		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	c	b	0	0
b	d	c	0	0
c	e	d	1	1
d	e	f	1	0
e	f	a	0	1
f	e	f	1	0

Similarly state  $d$  and  $f$  are also equivalent, therefore one of them say  $f$ , can be eliminated. After replacing all  $f$ 's by  $d$ 's in the columns, the reduced state table is given in table 6.

**Table 6. Reduced state table**

Present State	Next state		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	c	b	0	0
b	d	c	0	0
c	e	d	1	1
d	e	d	1	0
e	d	a	0	1

**Example 2.** A sequential circuit has two flip-flops say  $A$  and  $B$ , two inputs say  $X$  and  $Y$ , and an output say  $Z$ . The flip-flop input functions and the circuit output function are as follows :

$$JA = xB + y'B'$$

$$JB = xA'$$

$$Z = xy A + x'yB$$

$$KA = xy'B'$$

$$KB = xy' + A$$

Obtain state table, state diagram and state equations.

**Solution.**

State table for the problem is shown in table 6.1.

Present state		Next state				Output z			
A	B	xy = 00	xy = 01	xy = 10	xy = 11	xy = 00	xy = 01	xy = 10	xy = 11
		A B	A B	A B	A B				
0	0	1 0	0 0	1 1	0 1	0	0	0	0
0	1	0 1	0 1	1 0	1 1	1	0	0	0
1	0	1 0	1 0	0 0	1 0	0	0	0	1
1	1	1 0	1 0	1 0	1 0	1	0	0	1

Fig. 6 State table

With the help of state table we can draw the state diagram as shown in figure 6.

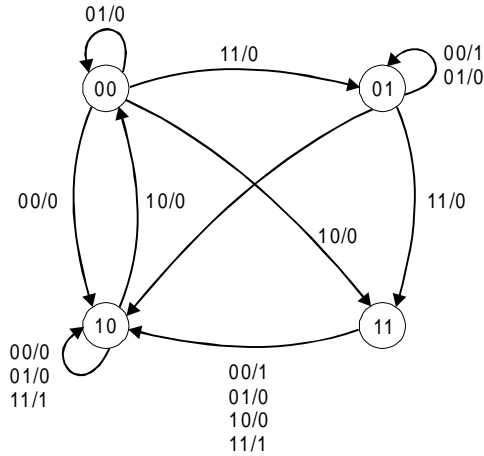


Fig. 6. State diagram

The state equation will be

$$A(t+1) = xB + Y' + Y A + x'A$$

$$B(t+1) = x A B' + x'AB + Y A B$$

**Example 3.** A clocked sequential circuit has three states, A, B and C and one input X. As long as the input X is 0, the circuit alternates between the states A and B. If the input X becomes 1 (either in state A or in state B), the circuit goes to state C and remains in the state C as long as X continues to be 1. The circuit returns to state A if the input becomes 0 once again and from then one repeats its behaviour. Assume that the state assignments are A = 00, B = 01 and C = 10.

- (a) Draw the state diagram.
- (b) Give the state table for the circuit.

**Solution.** (a) First draw circles for 3 states A, B, C and write state assignments.

The directed line indicate the transition and input on the directed line are those causes the change on the line. The figure 6. Shows the state diagram.

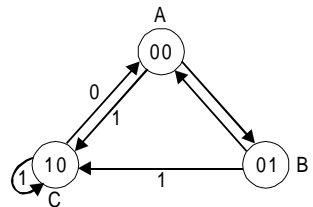


Fig. 6

(b) From the state diagram, a state table is drawn as shown in table 6.

**Table 6. State table**

Present State	Next state		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
A	B	C	0	1
B	A	C	0	1
C	A	C	0	1

Given the state assignments  $A = 00, B = 01, C = 10$ .

**Example 4.** A new clocked  $x$ - $Y$  flip-flop is defined with two inputs  $X$  and  $Y$  in addition to the clock input. The flip-flop functions as follows :

If  $XY = 00$ , the flip-flop changes state with each clock pulse.

If  $XY = 01$ , the flip flop state  $Q$  becomes 1 with the next clock pulse.

If  $XY = 10$ , the flip flop state  $Q$  become 0 with the next clock pulse.

If  $XY = 11$ , no change of state occurs with the clock pulse.

(a) Write the truth table for the  $X$ - $Y$  flip-flop.

(b) Write the Excitation table for the  $X$ - $Y$  flip flop.

(c) Draw a circuit to implement the  $X$ - $Y$  flip-flop using a  $J$ - $K$  flip-flop.

**Solution.** (a) Truth table for the clocked  $X$ - $Y$  flip flop is shown in table 6.

Inputs		Next state
$X$	$Y$	$Q_{n+1}$
0	0	$Q_n$
0	1	1
1	0	0
1	1	$Q_n$

(b) The Excitation table for the  $X$ - $Y$  flip flop is shown in table 6.

$Q_n$	$Q_{n+1}$	$X$	$Y$
0	0	1	x
0	1	0	x
1	0	x	0
1	1	x	1

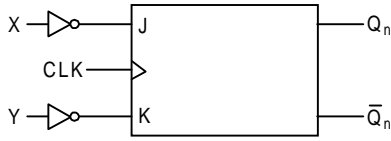
$x = \text{distance}$

(c) On comparing Excitation table of  $X$ - $Y$  flip-flop with  $J$ - $K$  flip-flop

$$X = \bar{J} : Y = \bar{K}$$

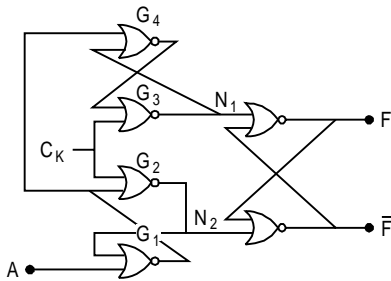
Therefore the  $X$ - $Y$  flip flop can be implemented using  $J$ - $K$  flip-flop as shown in figure 6.





**Example 5.** For the digital circuit shown in the figure 6. Explain what happens at the nodes  $N_1$ ,  $N_2$ ,  $F$  and  $\bar{F}$ , when

- (I)  $C_K = 1$  and 'A' changes from '0' to '1'.
- (II)  $A = 1$  and ' $C_K$ ' changes from '1' to '0'.
- (III)  $C_K = 0$  and 'A' changes from '1' to '0'.
- (IV) Initially,  $C_K = 0$  and 'A' changes from 0 to 1, and then  $C_K$  goes to 1.
- (V) What is the circuit performing.



**Solution.**

(I)

$N_{1(n)}$	$N_{2(n)}$	$C_K$	A	$G_1$	$G_2$	$G_3$	$G_4$	$N_{2(n+1)}$	$N_{1(n+1)}$
0	0	1	1	0	0	0	1	0	0
1	0	1	1	0	0	0	0	0	0
0	1	1	1	0	0	0	1	0	0
1	1	1	1	0	0	0	0	0	0

Initially if  $N_1, N_2$  are 0, it remains at 0. If at 1, they go to 0.

As  $N_1, N_2$  are at 0, in initially, if  $F$  is 1,  $F$  is 0. or initially if  $F = 0, \bar{F}$  is 1.

That is  $F, \bar{F}$  do not change their initial states.

(II)

$N_{1(n)}$	$N_{2(n)}$	$C_K$	A	$G_1$	$G_2$	$N_{2(n+1)}$	$G_3$	$G_4$	$N_{1(n+1)}$
0	0	0	1	0	1	1	0	1	0
1	0	0	1	0	1	1	1	0	1
0	1	0	1	0	1	1	0	1	0
1	1	0	1	0	1	1	1	0	1

$N_2$  continues to be 1 whatever be its initial state.  $N_1$  remains at 0 if initially 0, and 1 if initially 1.

If  $F = 0, \bar{F} = 0$

If  $N_1 = 0$ , F will become 1 and  $\bar{F} = 0$

If  $N_1 = 1$ , F will be 0 and  $\bar{F}$  also 0, which is prohibited state.

(III)

$N_{1(n)}$	$N_{2(n)}$	$C_K$	A	$G_1$	$G_2$	$N_{2(n+1)}$	$G_3$	$G_4$	$N_{1(n+1)}$
0	0	0	0	1	0	0	1	0	1
1	0	0	0	1	0	0	1	0	1
0	1	0	0	0	1	1	0	1	0
1	1	0	0	0	1	1	1	0	1

$N_2 = 0, F = 1, \bar{F} = 0, N_1 = 0, F = 1, N_1 = 1, F = 0, \bar{F} = 1$

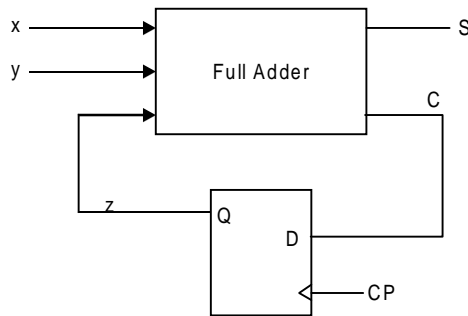
$N_2 = 1, F = 1, F = 0, N_1 = 0, F = 1, N_1 = 1, F = 0, \bar{F} = 1$

(IV) The change of state is similar to (II),  $C_K = 0, A = 1$  initially and finally as at (I),  $C_K = 1, A = 1$ .

(V) The circuit is functioning as a SR latch.

**Example 6.** The full adder given in figure 6 receives two external inputs  $x$  &  $y$ ; the third input  $Z$  comes from the output of a D flip-flop. The carry output is transferred to the flip-flop, every clock pulse. The external  $S$  output gives the sum of  $x, y$  and  $z$ . Obtain the state table and state diagram of the sequential circuit.

**Solution.**



The state table for the given circuit is shown in figure 6.

Present state $z$	Inputs		Next state $z$	Output $S$
	$x$	$y$		
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Fig. 6. State table

The state diagram corresponding to state-table of figure 6.

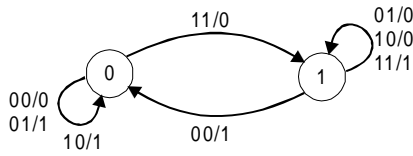
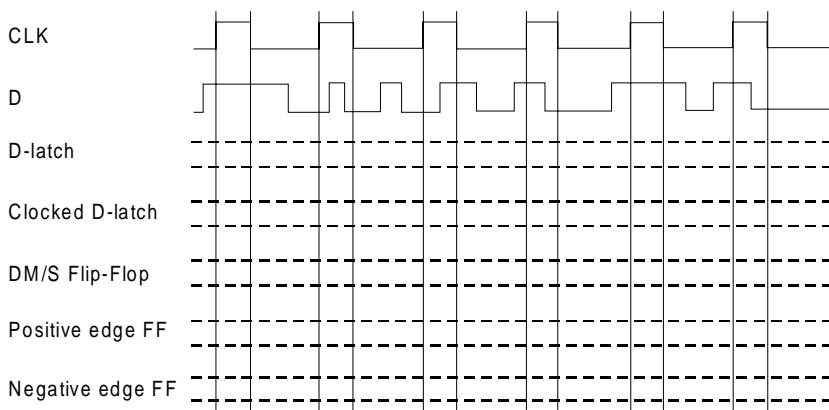


Fig. 6. State diagram

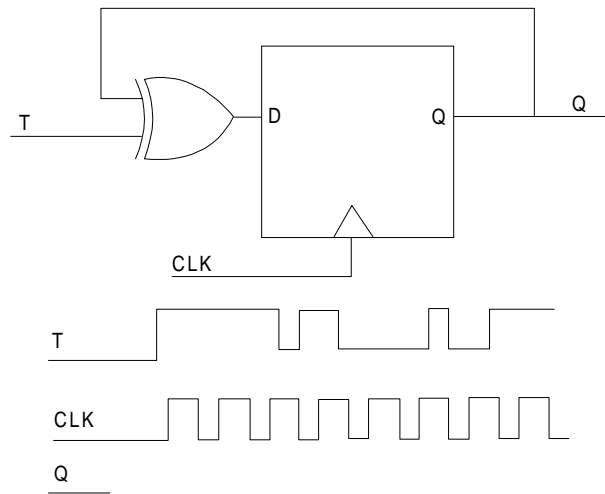
6.8 EXERCISES

1. An R-S latch can be based on cross-coupled NOR gates. It is also possible to construct an R'-S' latch using cross-coupled NAND gates.
  - (a) Draw the R'-S' latch, labelling the R' and S' inputs and the Q and Q' outputs.
  - (b) Show the timing behaviour across the four configurations of R' and S'. Indicate on your timing diagram the behaviour in entering and leaving the forbidden state when R'=S'=0.
  - (c) Draw the state diagram that shows the complete input/output and state transition behaviour of the R'-S' latch.
  - (d) What is the characteristic equation of the R'-S' latch.
  - (e) Draw a simple schematic for a gated R-S latch with an extra enable input, using NAND gates only.
2. Consider a D-type storage element implemented in five different ways :
  - (a) D-latch (i.e., D wired to the S-input and D' wired to the R-input of an R-S latch);
  - (b) Clock Enabled D-latch;
  - (c) Master-Slave Clock Enabled D-Flip-flop;
  - (d) Positive Edge-triggered Flip-flop;
  - (e) Negative Edge-triggered Flip-flop.

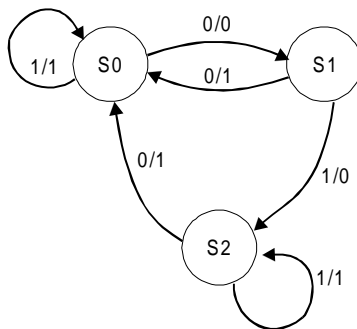
Complete the following timing charts indicating the behaviour of these alternative storage elements. Ignore set-up and hold time limitations (assume all constraints are meant):



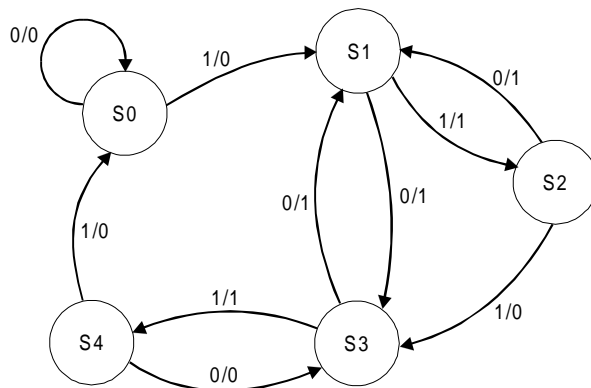
3. Complete the timing diagram for this circuit.



4. Design a circuit that implements the state diagram



5. Design a circuit that implements the state diagram

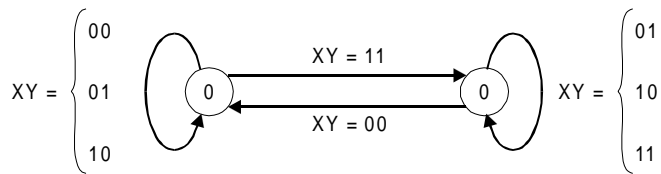


6. A sequential network has one input  $X$  and two outputs  $S$  and  $V$ .  $X$  represent a four bit binary number  $N$ , which is input least significant bit first.  $S$  represents a four bit binary number equal to  $N + 2$ , which is output least significant bit first. At the time the fourth input is sampled,  $V = 1$ , in  $N + 2$  is too large to be represented by four bits; otherwise  $V = 0$ . Derive a Mealy state graph and table with a minimum number of states.
7. A sequential network has one input  $X$  and two outputs  $S$  and  $V$ .  $X$  represent a four bit binary number  $N$ , which is input least significant bit first.  $S$  represents a four bit binary number

equal to  $N - 2$ , which is output least significant bit first. At the time the fourth input is sampled,  $V = 1$ , in  $N - 2$  is too small to be represented by four bits; otherwise  $V = 0$ .

Derive a Mealy state graph and table with a minimum number of states

8. Design a synchronous circuit using negative edge-triggered D flip-flops that provides an output signal Z which has one-fifth the frequency of the clock signal. Draw a timing diagram to indicate the exact relationship between the clock signal and the output signal Z. To ensure illegal state recovery, force all unused or illegal states to go to 0.
9. Consider the design of a sequence detector finite state machine that will assert a 1 when the current input equals the just previously seen input.
  - (a) Draw as simple state diagrams for a MEALY MACHINE and a MOORE MACHINE implementation as you can (minimization is not necessary). The MEALY MACHINE should have fewer states. Briefly explain why.
  - (b) If the Mealy Machine is implemented as a SYNCHRONOUS MEALY MACHINE, draw the timing diagram for the example input/output sequence described above.
  - (c) If the timing behaviours are different for the MOORE, MEALY, and SYNCHRONOUS MEALY machines, explain the reason why.
10. A sequential circuit is specified by the following flip-flop input functions. Draw the logic diagram of the circuit.  
 $JA = Bx'$      $KA = Bx$   
 $JB = x$        $KB = A \oplus x$
11. Design the circuit and draw the logic diagram of the sequential circuit specified by the following state diagram. Use an RS flip-flop.

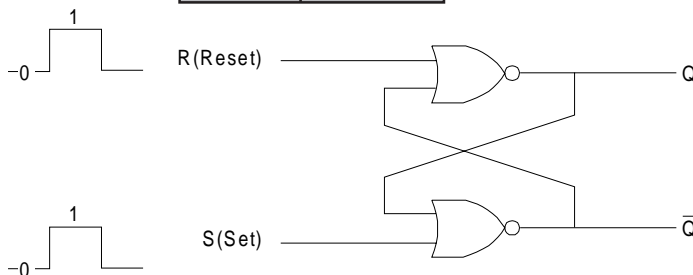


12. Complete the truth table for the latch constructed from 2 NOR gates.

S	R	Q	Q'
1	0		
0	0		
0	1		
0	0		
1	1		

(after  $S = 1, R = 0$ )

(after  $S = 0, R = 1$ )



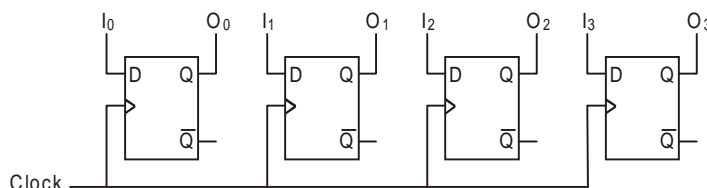
13. Construct a logic diagram of a clocked D flip-flop using AND and NOR gates.
14. Explain the master-slave flip-flop constructed from two R-S flip-flop.
15. Draw the logic diagram of a master-slave D flip-flop using NAND gates.

# 7 CHAPTER

## SHIFT REGISTERS AND COUNTERS

### 7.0 INTRODUCTION

Registers are the group of flip-flops (single bit storage element). The simplest type of register is a data register, which is used for the temporary storage of data. In its simplest form, it consists of a set of  $N$  D flip-flops, all sharing a common clock. All of the digits in the  $N$  bit data word are connected to the data register by an  $N$  line “data bus”. Fig. 7.0 shows a four bit data register, implemented with four D flip flops.



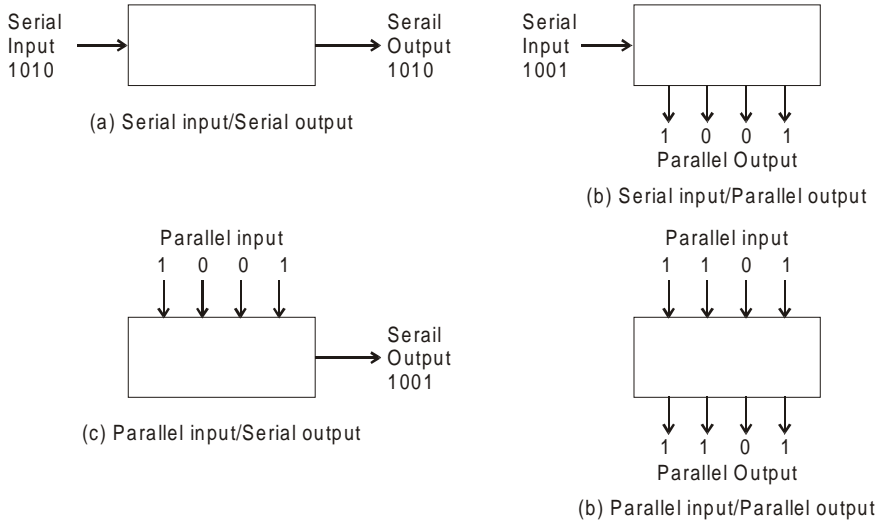
**Fig. 7.0** 4-bit D register

The data register is said to be a synchronous device, because all the flip flops change state at the same time.

### 7.1 SHIFT REGISTERS

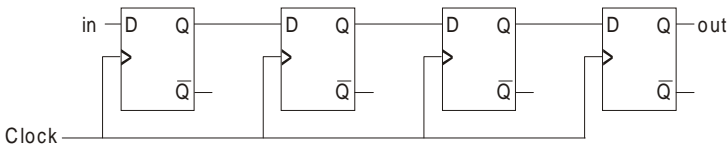
A common form of register used in many types of logic circuits is a shift register. Registers like counters, are sequential circuits and as they employ flip flops they possess memory; but memory is not the only requirement of a shift register. The function of storage of binary data can be very well performed by a simple register. Shift registers are required to do much more than that. They are required to store binary data momentarily until it is utilized for instance, by a computer, microprocessor, etc. Sometimes data is required to be presented to a device in a manner which may be different from the way in which it is fed to a shift register. For instance, shift register can present data to a device in a serial or parallel form, irrespective of the manner in which it is fed to a shift register. Data can also be manipulated within the shift register, so that it is presented to a device in the required form. These devices can also shift left or right and it is this capability which gives them the name of shift register. Fig. 7.1 show the many ways in which data can be fed into a shift register and presented by it to a device.

Shift registers have found considerable application in arithmetic operations. Since moving a binary number one bit to the left is equivalent to multiplying the number by 2 and moving the number one bit position to the right amounts to dividing the number by 2. Thus, multiplications and divisions can be accomplished by shifting data bits. Shift registers find considerable application in generating a sequence of control pulses.



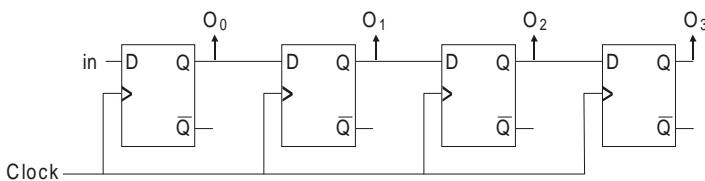
**Fig. 7.1** Data conversion with a shift register

Shift register is simply a set of flip flops (usually D latches or RS flip flops) connected together so that the output of one becomes the input of the next, and so on in series. It is called a shift register because the data is shifted through the register by one bit position on each clock pulse. Fig. 7.2 shows a four bit shift register, implemented with D flip flops.



**Fig. 7.2** 4-bit serial-in serial-out shift register

On the leading edge of the first clock pulse, the signal on the DATA input is latched in the first flip flop. On the leading edge of the next clock pulse, the contents of the first flip-flop is stored in the second flip-flop, and the signal which is present at the DATA input is stored in the first flip-flop, etc. Because the data is entered one bit at a time, this called a serial-in shift register. Since there is only one output, and data leaves the shift register one bit at a time, then it is also a serial out shift register. (Shift registers are named by their method of input and output; either serial or parallel.) Parallel input can be provided through the use of the preset and clear inputs to the flip-flop. The parallel loading of the flip flop can be synchronous (i.e., occurs with the clock pulse) or asynchronous (independent of the clock pulse) depending on the design of the shift register. Parallel output can be obtained from the outputs of each flip flop as shown in Fig. 7.3.



**Fig. 7.3** 4-bit serial-in parallel-out shift register

Communication between a computer and a peripheral device is usually done serially, while computation in the computer itself is usually performed with parallel logic circuitry. A shift register can be used to convert information from serial form to parallel form, and vice versa. Many different kinds of shift registers are available, depending upon the degree of sophistication required.

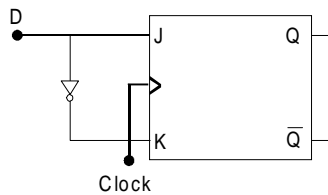
Here we deal with the basic characteristics of shift registers and their applications. Normally shift registers are obtained through D-Flip-Flops. However if required other flip flops may also be used. D-Flip-Flops are used because of simplicity that data presented at input is available at the output. Throughout the chapter it is our strategy to discuss all the shift registers using D-flip-flops only. If one need to use some other Flip-Flop, say JK Flip-Flop, then we recommend following procedure–

1. Design the shift register using D-flip flops only.
2. Take JK Flip-Flop and convert it into D Flip-Flop.
3. Replace each of the D-Flip-Flop of step1 by the flip-flop obtained in step 1 after conversion.

To elaborate this let us consider the shift register shown in Fig. 7.2.

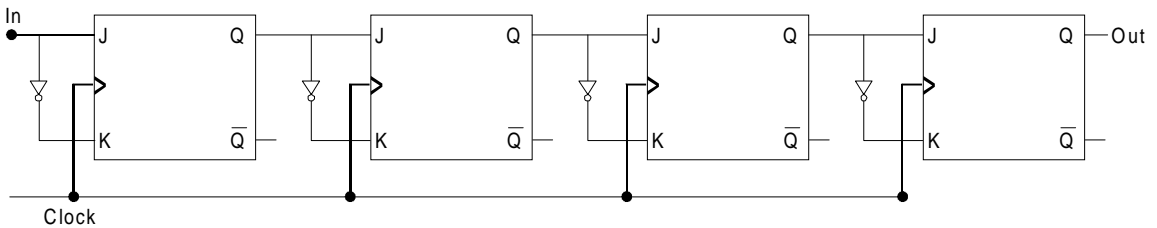
**Step 1:** It is readily obtained in Fig. 7.2.

**Step 2:** Convert JK into D-Flip-Flop. It is shown below in Fig. 7.4

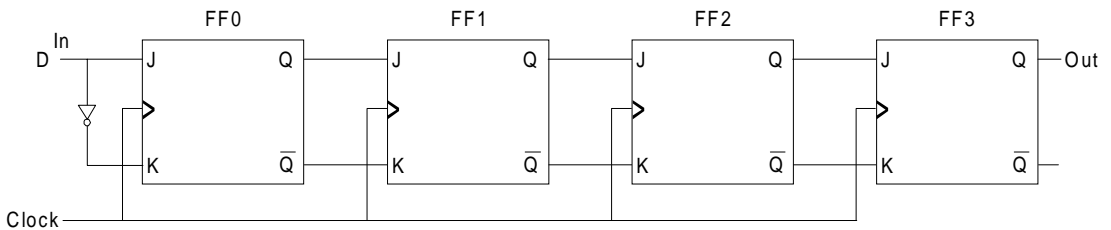


**Fig. 7.4** JK Flip-flop converted into D-Flip-Flop

**Step 3:** Replace each D-Flip-Flop of Fig. 7.2 by the one shown in Fig. 7.4.



**Fig. 7.5 (a)** 4-bit serial in serial out shift register using JK Flip-Flop



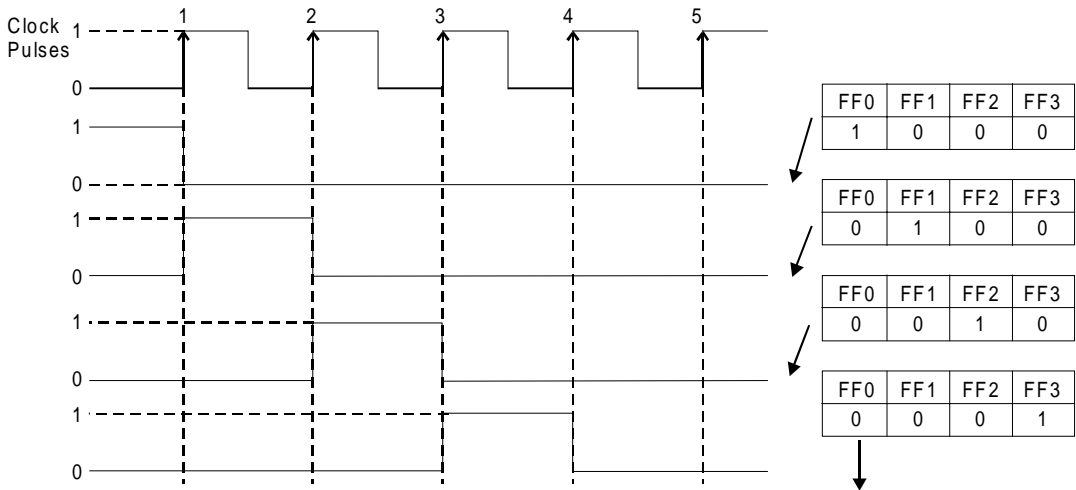
**Fig. 7.5 (b)** 4-bit serial in–serial out shift register using JK Flip-flop



**OPERATION**

A 4-bit shift register constructed with D type flip-flop (Fig. 7.2) and JK flip-flop (Fig. 7.5). By addition or deletion of flip-flop more or fewer bits can be accommodated. Except for FF0, the logic level at a data input terminal is determined by the state of the preceding flip-flop. Thus,  $D_n$  is 0 if the preceding flip-flop is in the reset state with  $Q_{n-1} = 0$ , and  $D_n = 1$  if  $Q_{n-1} = 1$ . The input at FF0 is determined by an external source.

From the characteristic of D-flip-flop we know that immediately after the triggering transition of the clock, the output Q of flip-flop goes to the state present at its input D just before this clock transition. Therefore, at each clock transition, pattern of bits, 1s and 0s, is shifted one flip-flop to the right. The bit of the last flip-flop (FF3 in Fig. 7.6) is lost, while the first flip-flop (FF0) goes to the state determined by its input  $D_0$ . This operation is shown in Fig. 7.6. We have assumed that the flip-flop triggers on the positive-going transition of the clock waveform, and initially we have  $D_0 = 0$ ,  $FF_0 = 1$  and  $FF_2 = FF_3 = FF_4 = 0$ .



**Fig. 7.6** A 4-bit shift register operation

**7.2 MODES OF OPERATION**

This section describes, the basic modes of operation of shift registers such as Serial In-Serial Out, Serial In-Parallel Out, Parallel In-Serial Out, Parallel In-Parallel Out, and bi-directional shift registers.

**7.2.1 Serial In-Serial Out Shift Registers**

A basic four-bit shift register can be constructed using four D-flip-flops, as shown in Fig. 7.7. The operation of the circuit is as follows. The register is first cleared, forcing all four outputs to zero. The input data is then applied sequentially to the D input of the first flip-flop on the left (FF0). During each clock pulse, one bit is transmitted from left to right. Assume a data word to be 1001. The least significant bit of the data has to be shifted through the register from FF0 to FF3.

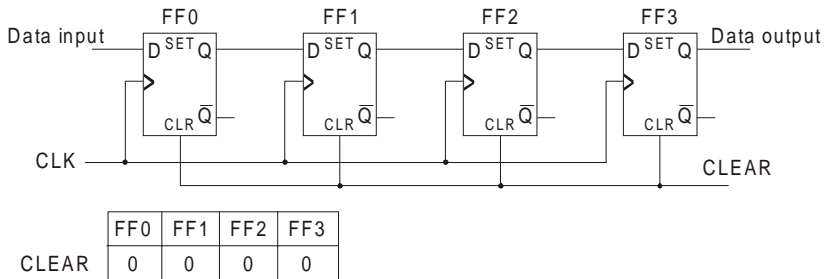


Fig. 7.7

In order to get the data out of the register, they must be shifted out serially. This can be done destructively or non-destructively. For destructive readout, the original data is lost and at the end of the read cycle, all flip-flops are reset to zero.

0000	FF0	FF1	FF2	FF3	0000
	1	0	0	1	

To avoid the loss of data, an arrangement for a non-destructive reading can be done by adding two AND gates, an OR gate and an inverter to the system. The construction of this circuit is shown in Fig. 7.8.

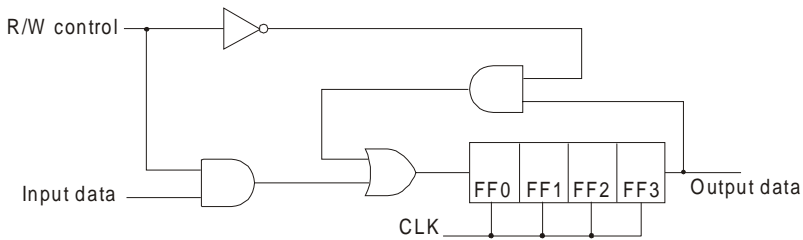


Fig. 7.8

The data is loaded to the register when the control line is HIGH (*i.e.* WRITE). The data can be shifted out of the register when the control line is LOW (*i.e.* READ).

### 7.2.2 Serial In-Parallel Out Shift Registers

For this kind of register, data bits are entered serially in the same manner as discussed in the last section. The difference is the way in which the data bits are taken out of the register. Once the data are stored, each bit appears on its respective output line, and all bits are available simultaneously. A construction of a four-bit serial in-parallel out register is shown in Fig. 7.9.

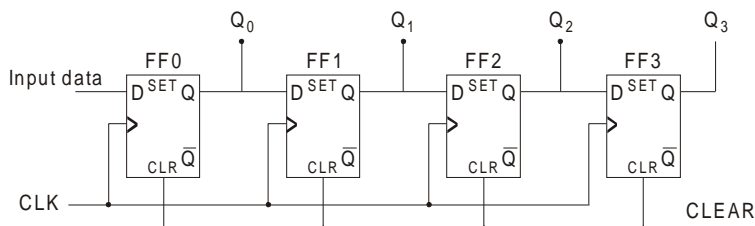


Fig. 7.9

### 7.2.3 Parallel In-Serial Out Shift Registers

A four-bit parallel in-serial out shift register is shown in Fig. 4.10. The circuit uses D-flip-flops and NAND gates for entering data (*i.e.*, writing) to the register.

$D_0$ ,  $D_1$ ,  $D_2$  and  $D_3$  are the parallel inputs, where  $D_0$  is the most significant bit and  $D_3$  is the least significant bit. To write data in, the mode control line is taken to LOW and the data is clocked in. The data can be shifted when the mode control line is HIGH as SHIFT is active high. The register performs right shift operation on the application of a clock pulse.

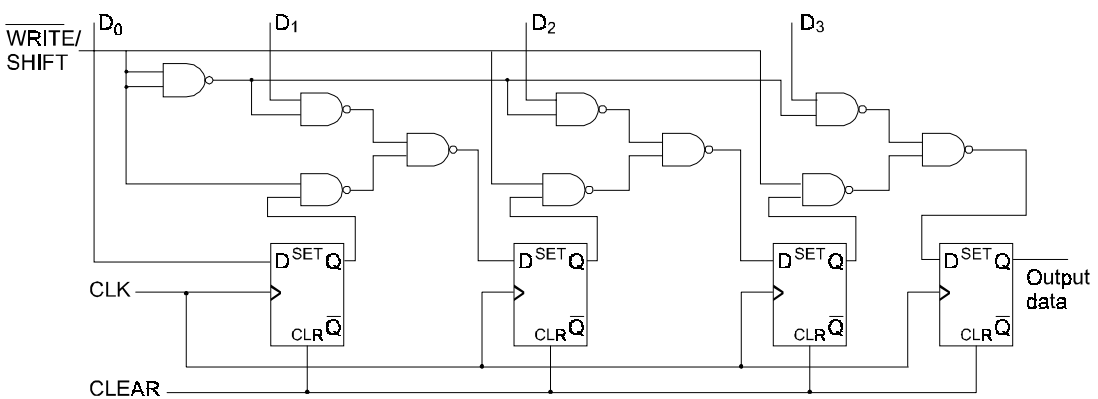


Fig. 7.10

### 7.2.4 Parallel In-Parallel Out Shift Registers

For parallel in-parallel out shift registers, all data bits appear on the parallel outputs immediately following the simultaneous entry of the data bits. The following circuit is a four-bit parallel in-parallel out shift register constructed by D-flip-flops.

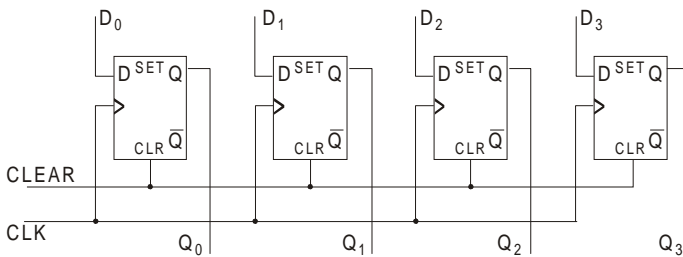


Fig. 7.11

The D's are the parallel inputs and the Q's are the parallel outputs. Once the register is clocked, all the data at the D inputs appear at the corresponding Q outputs simultaneously.

### 7.2.5 Bidirectional Shift Registers (Universal Shift Register)

The registers discussed so far involved only right shift operations. Each right shift operation has the effect of successively dividing the binary number by two. If the operation is reversed (left shift), this has the effect of multiplying the number by two. With suitable gating arrangement a serial shift register can perform both operations.

A bi-directional, or reversible shift register is one in which the data can be shift either left or right. A four-bit bi-directional shift register using D-flip-flops is shown in Fig. 7.12.

Here a set of NAND gates are configured as OR gates to select data inputs from the right or left adjacent bistables, as selected by the  $\overline{\text{LEFT/RIGHT}}$  control line.

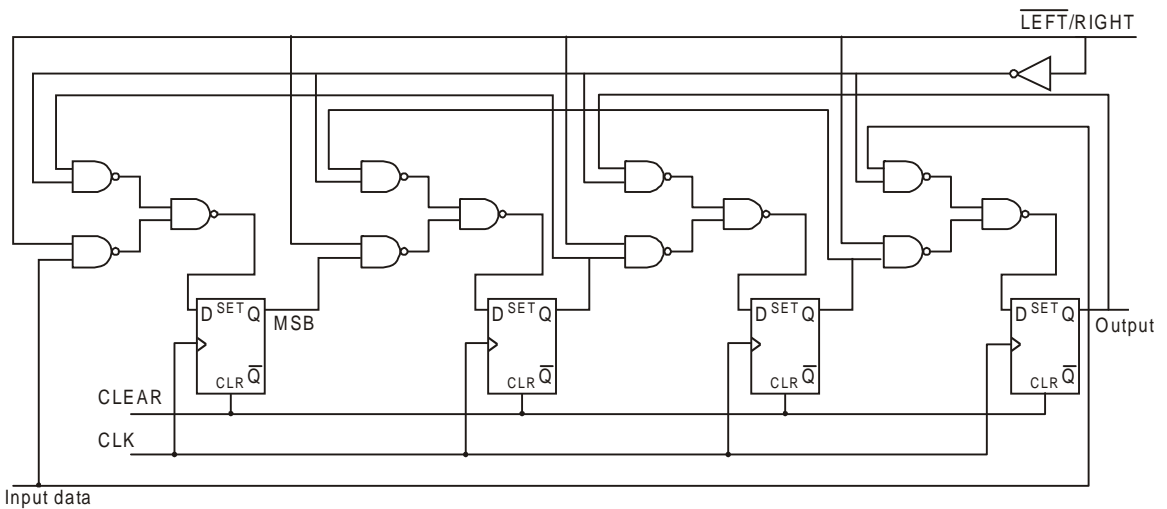


Fig. 4.12

### 7.3 APPLICATIONS OF SHIFT REGISTERS

Shift registers can be found in many applications. Here is a list of a few.

#### 7.3.1 To Produce Time Delay

The serial in-serial out shift register can be used as a time delay device. The amount of delay can be controlled by:

- the number of stages in the register ( $N$ )
- the clock frequency ( $f$ )

The time delay  $\Delta T$  is given by

$$\Delta T = N * f$$

#### 7.3.2 To Simplify Combinational Logic

The ring counter technique can be effectively utilized to implement synchronous sequential circuits. A major problem in the realization of sequential circuits is the assignment of binary codes to the internal states of the circuit in order to reduce the complexity of circuits required. By assigning one flip-flop to one internal state, it is possible to simplify the combinational logic required to realize the complete sequential circuit. When the circuit is in a particular state, the flip-flop corresponding to that state is set to HIGH and all other flip-flops remain LOW.

### 7.3.3 To Convert Serial Data to Parallel Data

A computer or microprocessor-based system commonly requires incoming data to be in parallel format. But frequently, these systems must communicate with external devices that send or receive serial data. So, serial-to-parallel conversion is required. As shown in the previous sections, a serial in-parallel out register can achieve this.

## 7.4 COUNTERS

### 7.4.1 Introduction

Both counters and registers belong to the class of sequential circuits. Here we will mainly deal with counters and also consider design procedures for sequential logic circuits. As the important characteristic of these circuits is memory, flip-flops naturally constitute the main circuit element of these devices and, therefore, there will be considerable emphasis on their application in circuit design.

You must already be familiar with some sequential devices, in which operations are performed in a certain sequence. For instance, when you dial a phone number, you dial it in a certain sequence, if not, you cannot get the number you want. Similarly, all arithmetic operations have to be performed in the required sequence.

While dealing with flip-flops, you have dealt with both clocked and unclocked flip-flops. Thus, there are two types of sequential circuits, clocked which are called synchronous, and unclocked which are called asynchronous.

In asynchronous devices, a change occurs only after the completion of the previous event. A digital telephone is an example of an asynchronous device.

If you are dialing a number, say 6354, you will first punch 6 followed by 3, 5 and 4. The important point to note is that, each successive event occurs after the previous event has been completed.

Sequential logic circuits find application in a variety of binary counters and storage devices and they are made up of flip-flops. A binary counter can count the number of pulses applied at its input. On the application of clock pulses, the flip-flops incorporated in the counter undergo a change of state in such a manner that the binary number stored in the flip-flops of the counter represents the number of clock pulses applied at the input. By looking at the counter output, you can determine the number of clock pulses applied at the counter input.

Digital circuits use several types of counters which can count in the pure binary form and in the standard BCD code as well as in some special codes. Counters can count up as well as count down. In this section we will be looking at some of the counters in common use in digital devices.

Another area of concern to us will be the design of sequential circuits. We will be considering both synchronous and asynchronous sequential circuits.

### 7.4.2 Binary Ripple Up-Counter

We will now consider a 3-bit binary up-counter, which belongs to the class asynchronous counter circuits and is commonly known as a ripple counter. Fig. 7.13 shows a 3-bit counter, which has been implemented with three T-type (toggle) flip-flops. The number of states of which this counter is capable is  $2^3$  or 8. This counter is also referred to as a modulo 8 (or divide by 8) counter. Since a flip-flop has two states, a counter having  $n$  flip-flops will have  $2^n$  states.

When clock pulses are applied to a ripple counter, the counter progresses from state to state and the final output of the flip-flop in the counter indicates the pulse count. The circuit recycles back to the starting state and starts counting all over again.

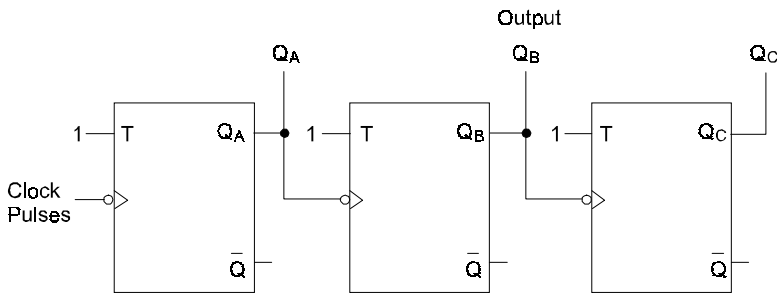


Fig. 7.13 3-Bit binary up-counter

There are two types of ripple counters, (a) asynchronous counters and (b) synchronous counters. In asynchronous counters all flip-flops are not clocked at the same time, while in synchronous counters all flip-flops are clocked simultaneously.

You will notice from the diagram that the normal output, Q, of each flip-flop is connected to the clock input of the next flip-flop. The T inputs of all the flip-flops, which are T-type, are held high to enable the flip-flops to toggle (change their logic state) at every transition of the input pulse from 1 to 0. The circuit is so arranged that flip-flop B receives its clock pulse from the  $Q_A$  output of flip-flop A and, as a consequence, the output of flip-flop B will change its logic state when output  $Q_A$  of flip-flop A changes from binary 1 to 0. This applies to all the other flip-flops in the circuit. It is thus an asynchronous counter, as all the flip-flops do not change their logic state at the same time.

Let us assume that all the flip-flops have been reset, so that the output of the counter at the start of the count is 0 0 0 as shown in the first row of Table 7.1. Also refer to Fig. 7.14 which shows the output changes for all the flip-flops at every transition of the input pulse from 1 → 0.

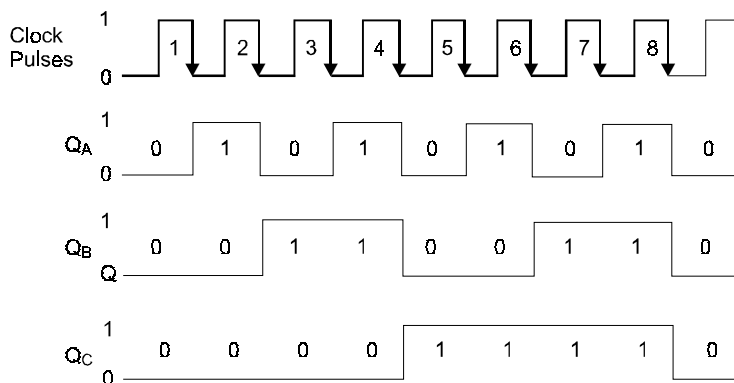


Fig. 7.14 Waveform for 3-bit binary ripple up-counter

When the trailing edge of the first pulse arrives, flip-flop A sets and  $Q_A$  becomes 1, which does not affect the output of flip-flop B. The counter output now is as shown in row 2 of the table. As a result of the second clock pulse, flip-flop A resets and its output  $Q_A$  changes from 1 to 0, which sets flip-flop B and the counter output now is as shown in row 3 of the table.

When the third clock pulse arrives, flip-flop A sets and its output  $Q_A$  becomes 1, which does not change the state of the B or the C flip-flop. The counter output is now as shown in row 3 of the table. When the fourth pulse occurs, flip-flop A resets and  $Q_B$  becomes 0 which in turn resets flip-flop B and  $Q_B$  becomes 0, which sets flip-flop C and its output changes to 1.

**Table 7.1 Count-up sequence of a 3-bit binary counter**

Input pulse	$2^2$	$2^1$	$2^0$
	$Q_C$	$Q_B$	$Q_A$
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

When the 5th clock pulse arrives, flip-flop A sets and  $Q_A$  becomes 1; but the other flip-flops remain unchanged. The number stored in the counter is shown in the 6th row of the table. The 6th pulse resets flip-flop A and at the same time flip-flop B and C are set. The 7th pulse sets all the flip-flops and the counter output is now shown in the last row of the table.

The next clock pulse will reset all the flip-flops, as the counter has reached its maximum count capability. The counter has in all 8 states. In other words it registers a count of 1 for every 8 clock input pulses. It means that it divides the number of input pulses by 8. It is thus a divide by 8 counter.

**Count Capability of Ripple Counters**

If you refer to Table 7.1 and the waveform diagram, Fig. 7.14, it will be apparent to you that the counter functions as a frequency divider. The output frequency of flip-flop A is half the input frequency and the output of flip-flop B is one-fourth of the clock input frequency. Each flip-flop divides the input frequency to it by 2. A 3-bit counter will thus divide the clock input frequency by 8.

Another important point about counters is their maximum count capability. It can be calculated from the following equation

$$N = 2^n - 1$$

where N is the maximum count number and

n is the number of flip-flops.

For example if  $n = 12$ , the maximum count capability is

$$N = 2^{12} - 1 = 4095$$

If you have to calculate the number of flip-flops required to have a certain count capability, use the following equation :

$$n = 3.32 \log_{10} N$$

For example if the required count capability is 5000

$$n = 3.32 \log_{10} 5000 = 12.28$$

which means that 13 flip-flops will be required.

### Counting Speed of Ripple Counters

The primary limitation of ripple counters is their speed. This is due to the fact that each successive flip-flop is driven by the output of the previous flip-flop. Therefore, each flip-flop in the counter contributes to the total propagation delay. Hence, it takes an appreciable time for an impulse to ripple through all the flip-flops and change the state of the last flip-flop in the chain. This delay may cause malfunction, if all the flip-flops change state at the same time. In the counter we have just considered, this happens when the state changes from 011 to 100 and from 111 to 000. If each flip-flop in the counter changes state during the course of a counting operation, and if each flip-flop has a propagation delay of 30 nanoseconds, a counter having three flip-flops will cause a delay of 90 ns. The maximum counting speed for such a flip-flop will be less than.

$$\frac{1}{90} \times 10^9 \text{ or } 11.11 \text{ MHz.}$$

If the input pulses occur at a rate faster than 90 ns, the counter output will not be a true representation of the number of input pulses at the counter. For reliable operation of the counter, the upper limit of the clock pulses of the counter can be calculated from

$$f = \frac{1}{nt} \times 10^9$$

where  $n$  is the number of flip-flops and

$t$  is the propagation delay of each flip-flop.

#### 7.4.3 4-BIT BINARY RIPPLE UP-COUNTER

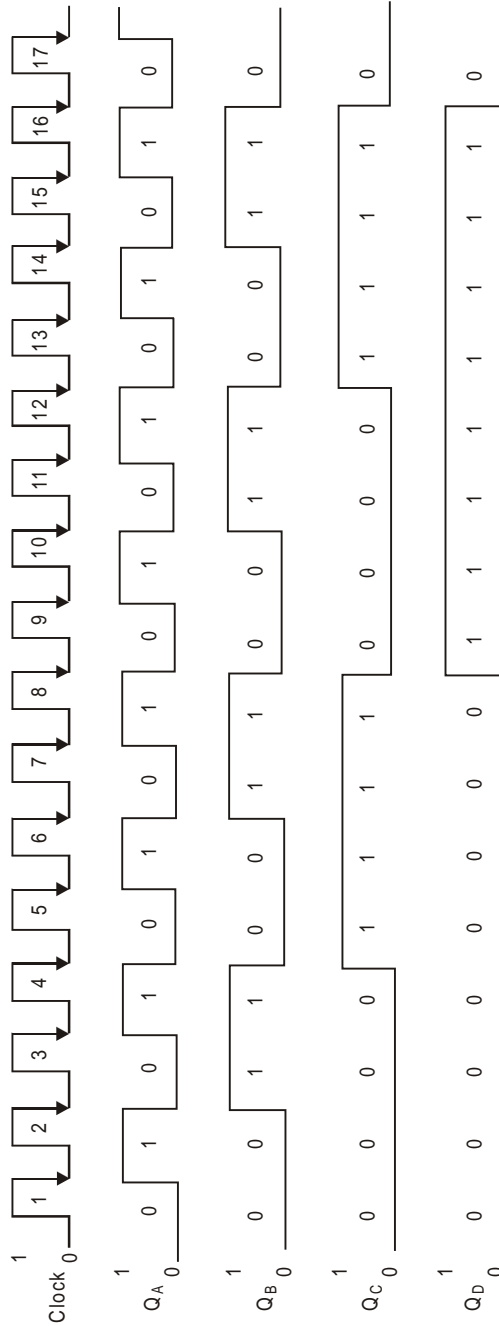
A 4-bit binary ripple up-counter can be built with four T-type flip-flops. The diagram will follow the same pattern as for a 3-bit up-counter. The count-up sequence for this counter is given in Table 7.2 and a waveform diagram is given in Fig. 7.15. After the counter has counted up to

**Table 7.2 Count-up sequence of a 4-bit binary up-counter**

Input pulse	Count			
	$2^3$ $Q_D$	$2^2$ $Q_C$	$2^1$ $Q_B$	$2^0$ $Q_A$
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	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16 or 0	0	0	0	0



1111, it recycles to 0000 like the 3-bit counter. You must have observed that each flip-flop divides the input frequency by 2 and the counter divides the frequency of the clock input pulses by 16.



**Fig. 7.15** Waveform for 4-bit binary up-counter

### 7.4.4 3-Bit Binary Ripple Down Counter

The binary ripple up-counter we have just considered increases the count by one, each time a pulse occurs at its input. The binary ripple down counter which we are going to consider in this section decreases the count by one, each time a pulse occurs at the input. A circuit for a 3-bit down counter is given in Fig. 7.16. If you compare this counter with the up-counter in Fig. 7.13 the only difference you will notice is that, in the down counter in Fig. 7.16 the complement output  $\bar{Q}$ , instead of the normal output, is connected to the clock input of the next flip-flop. The counter output which is relevant even in the down counter is the normal output,  $Q$ , of the flip-flops.

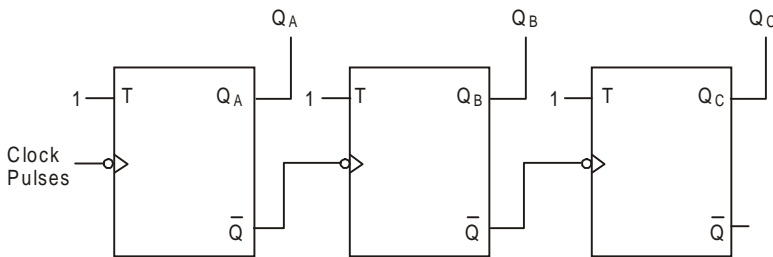


Fig. 7.16 3-bit binary ripple down counter

We can now analyze the circuit and examine its operation. It will help you to follow the operation of the counter; if you refer to Table 7.3 and waveform of the counter given in Fig. 7.17 for each input pulse count. Let us assume that the counter is initially reset, so that the counter output is 0 0 0. When the first input pulse is applied, flip-flop A will set, and its complement output will be 0. This will set flip-flop B, as there will be a 1 → 0 transition at the clock input. The counter output will now be 1 1 1.

Table 7.3 Count-down sequence of a 3-bit binary counter

Clock pulse	Count		
	2 <sup>2</sup> Q <sub>C</sub>	2 <sup>1</sup> Q <sub>B</sub>	2 <sup>0</sup> Q <sub>A</sub>
0	0	0	0
1	1	1	1
2	1	1	0
3	1	0	1
4	1	0	0
5	0	1	1
6	0	1	0
7	0	0	1
8	0	0	0

When the second clock pulse is applied, flip flop A will reset and its complement output will become 1, which will not affect the other flip-flops. The counter output will now be 1 1 0 as shown in row 3 of the Table 7.3.

When the third clock pulse occurs, flip-flop A will set and its complement output will become 0, which will reset flip-flop B, its output becomes 0, and the complement output will

be 1, which will not affect the other flip-flops. The counter will now show an output of 1 0 1, as in the fourth row of the table.

You will notice that every clock pulse decrements the counter by 1. After the eighth clock pulse, the counter output will be 0 0 0 and the counter will recycle thereafter.

The waveform for this 3-bit down counter is given in Fig. 7.17.

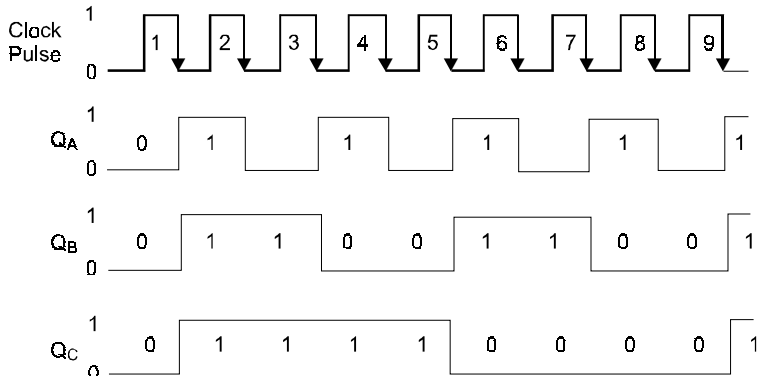


Fig. 7.17 Waveform for 3-bit binary down-counter.

### 7.4.5 Up-Down Counters

The counters which we have considered so far can only count up or down; but they cannot be programmed to count up or down. However, this facility can be easily incorporated by some modification in the circuitry. You might recall that in an up-counter the normal output of a flip-flop is connected to the clock input of the following flip-flop, and in a down counter it is the complement output which is connected to the clock input of the following flip-flop. The change from normal to complement connection to the clock input of the following flip-flop can be easily managed. A circuit for this purpose is shown in Fig. 7.18.

The normal and complement outputs of flip-flops are connected to AND gates D and E and the output of the AND gates goes to the clock input of the next flip-flop via OR gates F. When the up-down control is binary 1, gates D and F are enabled and the normal output of each flip-flop is coupled via OR gates F to the clock input of the next flip-flop. Gates E are inhibited, as one input of all these gates goes low because of the Inverter. The counter, therefore, counts up.

When the up-down control is binary 0, gates D are inhibited and gated E are enabled. As a consequence the complement output of each flip-flop is coupled via OR gates F to the clock input of the next flip-flop. The counter, therefore, counts down.

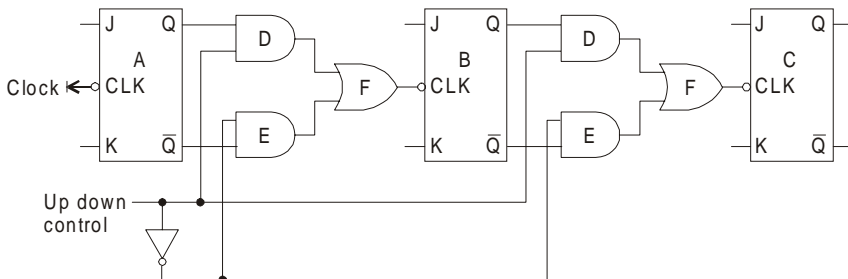


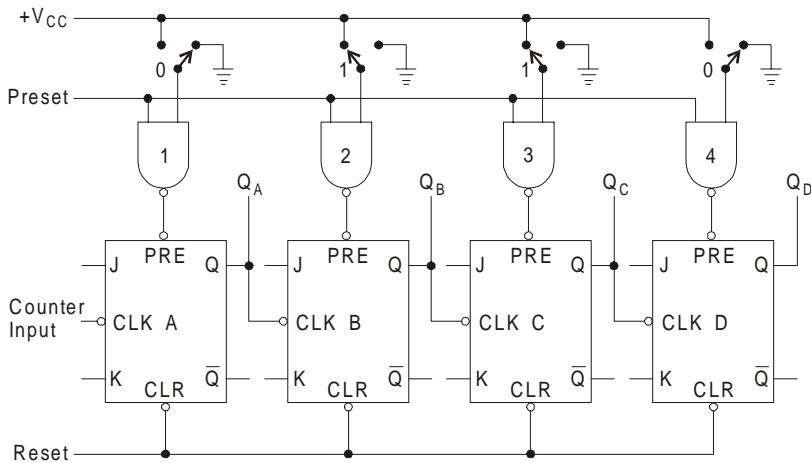
Fig. 7.18 Up-down counter

### 7.4.6 Reset and Preset Functions

Reset and Preset functions are usually necessary in most counter applications. When using a counter you would, in most cases, like the counter to begin counting with no prior counts stored in the counter. Resetting is a process by which all flip-flops in a counter are cleared and they are thus in a binary 0 state. JK flip-flops have a CLEAR or RESET input and you can activate them to reset flip-flops. If there are more than one flip-flop, the reset inputs of all flip-flops are connected to a common input line as shown in Fig. 7.19.

You will notice that the reset inputs of all the flip-flops in the counter are active low, and therefore, to reset the counter you take the reset input line low and then high. The output of the counter will then be 0 0 0 0.

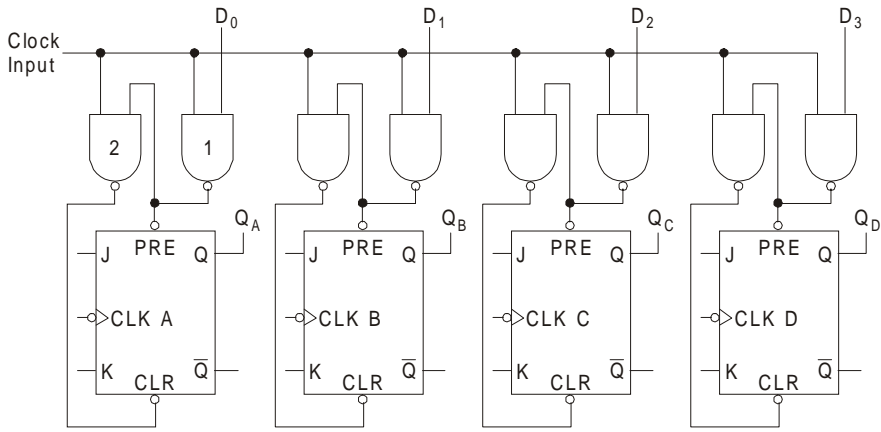
At times you may want the counter to start the count from a predetermined point. If you load the required number into the counter, it can start counting from that point. This can be easily accomplished by using the arrangement shown in diagram. The preset inputs of all the flip-flops are connected to NAND gate outputs. One input of each NAND gate is connected to a common PRESET line and the desired number is fed into the other inputs of the NAND gates. To load a number into the counter, first clear the counter and then feed the required number into the NAND gates as indicated in the diagram. When you take the PRESET line high momentarily, the output of NAND gates 1 and 4 will be 1, so flip-flops A and D will remain reset. The output of gates 2 and 3 will be 0 and so flip-flops B and C will be set. The number stored in the counter will now be 0 1 1 0, which is the number required to be loaded in the counter.



**Fig. 7.19**

It is also possible to load a number in a counter in a single operation, by using the arrangement shown in Fig. 7.20.

The arrangement for data transfer, which is a single pulse operation makes use of the Preset and Clear inputs of the flip-flops. When the clock pulse is low, the output of both NAND gates 1 and 2 is high, which has no effect on the Preset and Clear inputs of the flip-flop and there is no change in its output. If the  $D_0$  input is high, the output of NAND gate 1 will go low when the clock pulse goes high. This will result in output  $Q_A$  going high at the same time. Since one input of NAND gate 2 will be low at this time, the clear input to the flip-flop remains high.



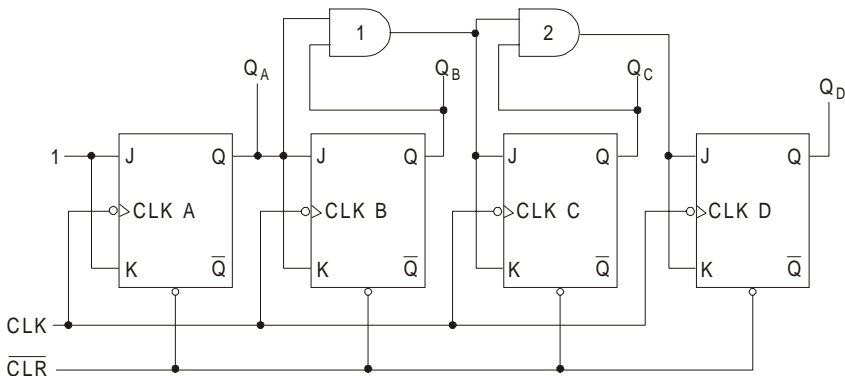
**Fig. 7.20** Single pulse data transfer

If the  $D_0$  input is low and the clock pulse goes high, the output of NAND gate 1 will remain high, which will have no effect on the Preset input. The output of NAND gate 2 will go low, which will clear the flip-flop and  $Q_A$  will go low.

### 7.4.7 Universal Synchronous Counter Stage

The up and down counters which we have considered so far are asynchronous counters, also known as ripple counters, for the simple reason that, following the application of a clock pulse, the count ripples through the counter, since each successive flip-flop is driven by the output of the previous flip-flop. In a synchronous counter all flip-flops are driven simultaneously by the same timing signal.

The asynchronous counter, therefore, suffers from speed limitation as each flip-flop contributes to the total propagation delay. To overcome this draw-back, flip-flops with lower propagation delay can be used; but the ideal solution is to use synchronous counters. In these counters the circuit is so arranged that triggering of all flip-flops is done simultaneously by the input signal, which is to be counted. In these counters the total propagation delay is the delay contributed by a single flip-flop.



**Fig. 7.21 (a)** Synchronous counter

The design concept used in the synchronous counter shown in Fig. 7.21 (a) uses counter stage blocks and this design concept lends itself to building large synchronous counters. Counter modules of the type used in this circuit and also shown separately in Fig. 7.21 (b) can be interconnected to build counters of any length.

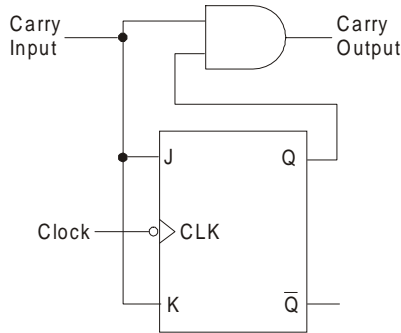


Fig. 7.21 (b) Universal counter stage block.

Let us consider the synchronous counting circuit shown in Fig. 7.21(a). It is a 4-bit counter and the clock inputs of all the flip-flops are connected to a common clock signal, which enables all flip-flops to be triggered simultaneously. The Clear inputs are also connected to a common Clear input line. The J and K inputs of each flip-flop are connected together, so that they can toggle when the JK input is high. The JK input of flip-flop A is held high. Also notice the two AND gates 1 and 2, and the way they are connected. Gate 1 ensures that the JK input to flip-flop C will be binary 1 when both inputs  $Q_A$  and  $Q_B$  are binary 1. AND gate 2 ensures that the JK input to flip-flop D will be binary 1 only when outputs  $Q_A$ ,  $Q_B$  and  $Q_C$  are binary 1.

We can now look into the output states required for the flip-flops to toggle. This has been summarized below :

1. Flip-flop A toggles on negative clock edge.
2. Flip-flop B toggles when  $Q_A$  is 1
3. Flip-flop C toggles when  $Q_A$  and  $Q_B$  are 1
4. Flip-flop D toggles when  $Q_A$ ,  $Q_B$  and  $Q_C$  are 1

This means that a flip-flop will toggle only if all flip-flops preceding it are at binary 1 level.

We can now look into the counting process of this counter. We begin by resetting the counter, which is done by taking CLR temporarily low.

MSB		LSB	
$Q_D$	$Q_C$	$Q_B$	$Q_A$
0	0	0	0

Since  $Q_A$  is low and J and K are high, the first negative clock edge will set flip-flop A. The counter output will now be as follows:

$Q_D$	$Q_C$	$Q_B$	$Q_A$
0	0	0	1

After 1st clock pulse.

When the second negative clock edge occurs, both A and B flip-flops will toggle and the counter output will change to the following:

$Q_D$	$Q_C$	$Q_B$	$Q_A$
0	0	1	0

After 2nd clock pulse.

When the third clock pulse arrives, flip-flop B will not toggle as  $Q_A$  is 0 but flip-flop A will toggle. The counter will show the following output.

$Q_D$	$Q_C$	$Q_B$	$Q_A$	
0	0	1	1	After 3rd clock pulse.

The fourth clock pulse will toggle flip-flops A, B and C, as both  $Q_A$  and  $Q_B$  are 1. The counter output is now as follows:

$Q_D$	$Q_C$	$Q_B$	$Q_A$	
0	1	0	0	After 4th clock pulse.

The counter will continue to count in the binary system until the counter output registers 1 1 1 1, when it will be reset by the next clock pulse and the counting cycle will be repeated.

#### 7.4.8 Synchronous Counter ICs

Many types of counter ICs are available and it is very likely that one of these will meet your design requirements. You may not, therefore, find it necessary to design your own counter. However, should that become necessary, a variety of JK flip-flops are available, with which you can design one to meet your specific needs.

Some of the counter ICs available are listed below :

<i>Counter type</i>	<i>Parallel load</i>	<i>IC No.</i>
1 Decade Up	Synchronous	74160
	Synchronous	74162
2 Decade Up/Down	Synchronous	74168
	Synchronous	74ALS568
	Asynchronous	74ALS190
	Asynchronous	74ALS192
3 4-bit binary Up counter	Synchronous	74161
	Synchronous	74163
4 4-bit binary Up/Down counter	Synchronous	74169
	Asynchronous	74191
	Asynchronous	74193
	Asynchronous	74ALS569

All the counters listed here have parallel load capability, which implies that a binary data input applied to the counter will be transferred to the output when the load input on the counter is asserted. The load operation may be synchronous or asynchronous. If it is asynchronous, the data applied will be transferred to the output as soon as the load input is asserted.

In case it is synchronous, the data will not be transferred to the output until the next clock pulse occurs. In either case the counters will begin to count from the loaded count when clock pulses are applied.

The decade up-counters count from 0000 to 1001. Decade up-and down-counters can be programmed to count from 0000 to 1001, or from 1001 to 0000. 4-Bit binary counters in the up-count mode count from 0000 and 1111 and in the down-count mode count from 1111 to 0000.

We will now discuss the facilities available in counter IC 74193 and its operating procedures. Pin connections for this IC are given in Fig. 7.22. Fig. 7.23(a) gives its traditional symbol and Fig. 7.23 (b) gives the IEEE/IEC symbol.

### 7.4.8.1 Counter Functions

#### Clear (Reset) Function

As the Clear input is active high, it is normally held low. To clear the counter it is taken high momentarily.

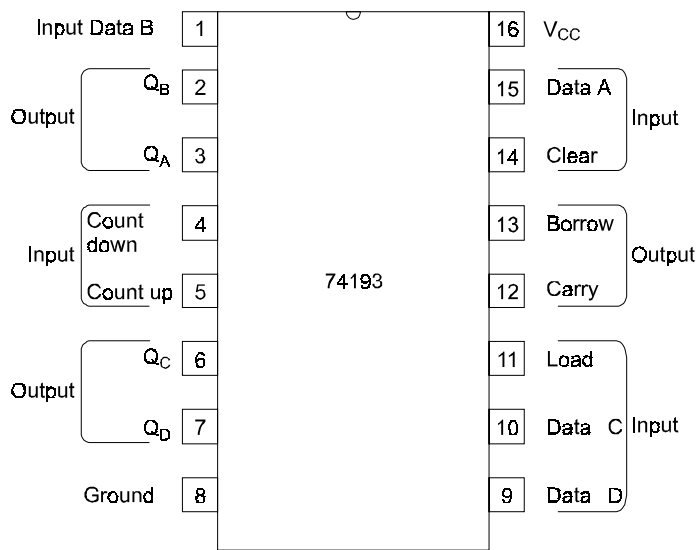


Fig. 7.22 Pin connections for IC 74193

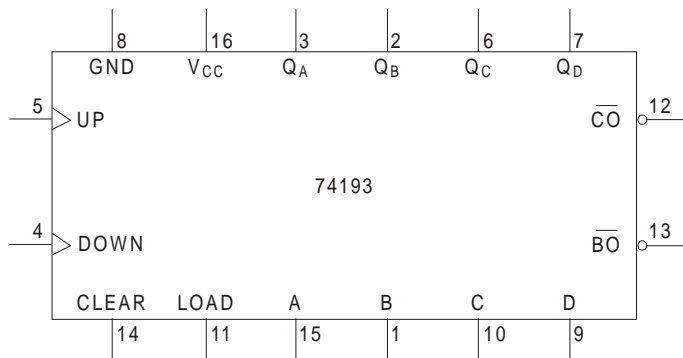
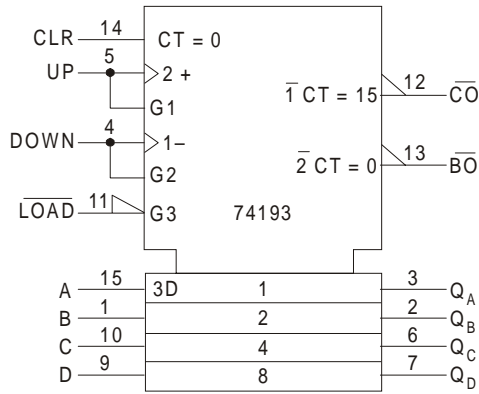


Fig. 7.23 (a) Traditional symbol for IC 74193

#### Load (Preset) Function

To load the counter with a predetermined 4-bit binary number, it is fed into the parallel data inputs A, B and C and D. The number is shifted into the counter by taking the LOAD input momentarily low. Both Clear and LOAD inputs are asynchronous and will override all synchronous counting functions.





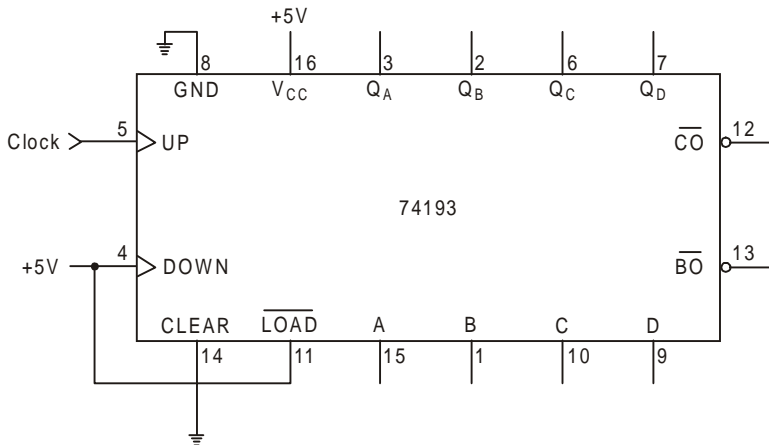
**Fig. 7.23** (b) IEEE/IEC symbol for IC 74193

**Carry out ( $\overline{CO}$ ) and Borrow Out ( $\overline{BO}$ ) Functions**

These inputs are used to drive the next IC74193, if a larger count capability is required. While cascading these counters, the  $\overline{CO}$  and  $\overline{BO}$  outputs of a previous counter are connected to the UP and Down inputs respectively, of the next counter in the chain.

**Up-counting Function**

For counting up, the counter is connected as shown in Fig. 7.24. In the up-counting mode the carry output  $\overline{CO}$  remains high, until the maximum count 1111 is reached, when the carry output goes low. At the next clock pulse the counter output falls to 0 0 0 0 and the carry output  $\overline{CO}$  goes high. If there is another IC in cascade, it will be incremented from 0 0 0 0 to 0 0 0 1.

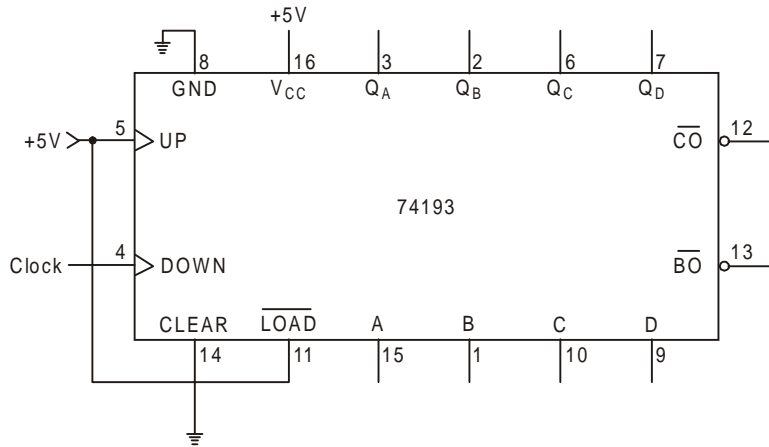


**Fig. 7.24** Counter connected to count up

**Down-counting Function**

For down-counting connection are made as shown in Fig. 7.25. In the down-counting operation the borrow output  $\overline{BO}$  stays high until the minimum count 0 0 0 0 is reached, when the borrow output  $\overline{BO}$  drops low. The borrow output detects the minimum counter value.

At the next input pulse the counter output rises to 1 1 1 1 and there is a 0  $\rightarrow$  1 transition at the borrow output  $\overline{BO}$ . If another counter is connected in cascade it will be decremented.



**Fig. 7.25** Counter connected to count down

#### *Presetting (Up-Counting Mode)*

The counter can be preset to any 4-bit binary number, which is first fed into the parallel inputs A, B, C and D, and the load input is held low momentarily, which shifts the number into the counter. It is not necessary to reset the counter before presetting it. Let us suppose that the number shifted into the counter 1 0 1 0 and the counter is made to count up. The counter output will be stepped up after each input pulse and after the 6th pulse the output will be 0 0 0 0. The counting up begins from the number preset in the counter and the 6th pulse resets the counter and then it starts counting up from this point.

#### *Presetting (Down-Counting Mode)*

The counter is set up in the down-counting mode and, as before, suppose the number fed into the counter is 1 0 1 0 and the counter is made to count down. The 10th, input pulse will reset the counter to 0 0 0 0 and the 11th, pulse will show a count of 1 1 1 1 and then it will begin to count down from this number.

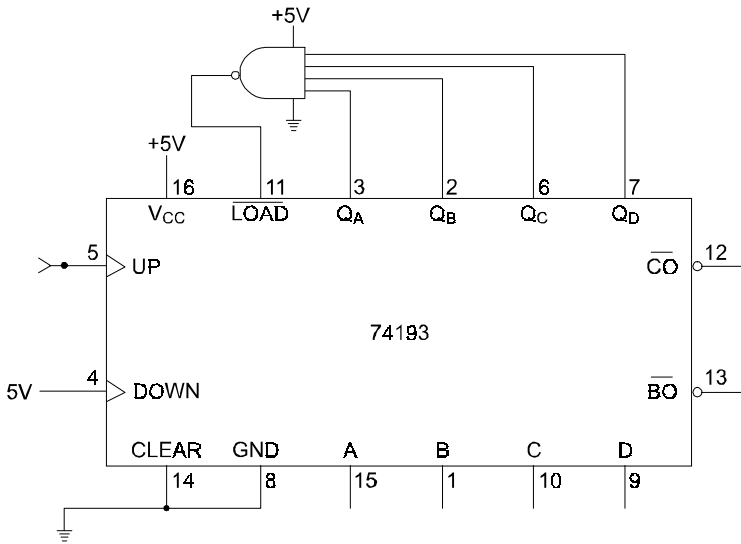
#### *Presetting (Using Counter Output)*

The counter can also be preset by connecting it as shown in Fig. 7.26. The desired number, say 1 0 1 0 is fed into the A B C D inputs and the counter input is connected to a 1 Hz clock signal when the counter reaches the maximum, count, 1 1 1 1, the NAND gate output will go low and the binary number 1 0 1 0 will be shifted into the counter. The counter will now begin to count up from this preset number and when the count again reaches 1 1 1 1, the counter will return to the preset number 1 0 1 0 and will again begin to count up as before. You will notice that as soon as the counter reaches the maximum count 1 1 1 1 (or decimal 15), it is immediately preset to 1 0 1 0 (or decimal 10). Since state 15 is being used to preset the counter, it is no longer a stable state. The stable states in this counting operation will be 10, 11, 12, 13 and 14, and the modulus (number of discrete states) of the counter will be 5.

The counter modulus in the up-counting mode for any preset number  $n$  is given by

$$\text{Modulus} = 16 - n - 1$$

In this case       $\text{Modulus} = 16 - 10 - 1 = 5$



**Fig. 7.26** Presetting using counter output

In the down-counting mode, the counter will count down from the preset number, 1010 (or decimal 10). As before the counter will count down as follows; 9, 8, 7, 6, 5, 4, 3, 2, 1, 0. In this case the counter modulus will be as follows :

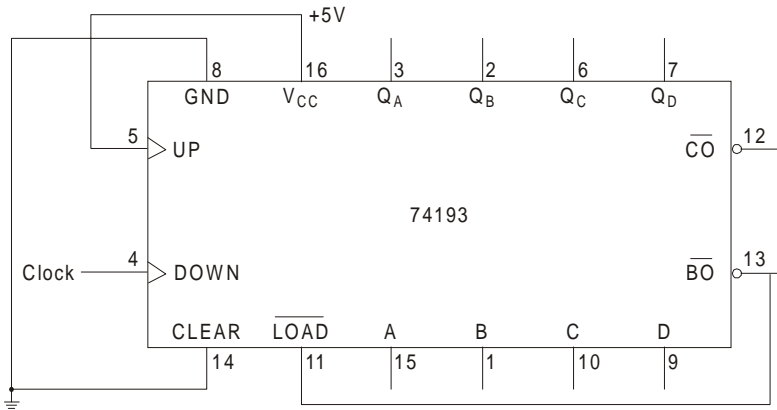
$$\text{Modulus} = n + 1$$

In this case             $\text{Modulus} = 10 + 1 = 11$

When the counter is preset in this manner, it should never be preset to number 15 as this is not a stable state, and it is likely to get latched up in this state.

**Modulo N Counter Using IC 74193**

In Sec. 7.4.8 you have seen how this IC can count up or down from a preset number. In fact it can function as a modulo counter by using a NAND gate to preset the counter to the desired number. There is a simpler way of implementing a modulo counter with this IC, but it has some drawbacks which we will discuss shortly. A circuit for a modulo counter using this IC in the down-counting mode is given in Fig. 7.27.



**Fig. 7.27** IC 74193 connected as a modulus counter

Clock pulses are applied at the down-count input and the up-count input is held high. Also observe that the borrow output  $\overline{BO}$  is connected to the load input. The borrow output detects the state of the borrow output when the count reaches 0 0 0 0. Since it is connected back to the load input, the binary number loaded into the A B C D inputs is shifted into the counter. The decimal number loaded into the counter represents its modulus.

To operate the counter load, the binary equivalent of the decimal number representing the required modulus into the ABCD inputs and apply clock pulses at the down-count input. If the binary number loaded into the counter is 1 0 0 0 (decimal 8) and the counter is decremented with clock pulses, the modulus number that is 1 0 0 0 will be loaded into the counter, as soon as the output reaches the state 0 0 0 0. It will again count down to 0 0 0 0 and will again be preset to 1 0 0 0.

You must have realized that as soon as the preset number is loaded into the counter, the borrow output, that is 0 0 0 0 will disappear. It is important, therefore, that the borrow output state, 0 0 0 0, must be of sufficient duration to enable the preset number to be shifted into the counter. This implies that the propagation delay of the gates responsible for presetting the counter to the number at the A B C D inputs must be of shorter duration than the duration of the clock pulse. To a certain extent this can be ensured by introducing some delay between the borrow output and the load input. This can be done by connecting an even number of inverters between the borrow output  $\overline{BO}$  and the load input.

#### 7.4.9 Modulus Counters

The modulus of a counter, as discussed before, is the number of discrete states a counter can take up. A single flip-flop can assume only two states 0 and 1, while a counter having two flip-flops can assume any one of the four possible states. A counter with three flip-flops will have 8 states and so on. In short the number of states is a multiple of 2. With  $n$  flip-flops the number of possible states will be  $2^n$ . Thus by building counters which count in the normal binary sequence, we can build counters with modulus of 2, 4, 8, 16 etc. In these counters the count increases or decreases by 1 in pure binary sequence. The problem arises in building counters whose modulus is 3, 5, 7, 9 etc. For instance, if we need, a counter with a modulus of 3, we have to use a counter with a modulus of 4 and so arrange the circuit that it skips one of the states. Similarly, for a counter with a modulus of 5 we require  $2^3$  or 8 states and arrange the circuit so that it skips 3 states to give us a modulus of  $2^n - 3$  or 5 states. Thus for a modulus N counter the number  $n$  of flip-flops should be such that  $n$  is the smallest number for which  $2^n > N$ . It, therefore, follows that for a decade (mod-10) counter the number of flip-flops should be 4. For this counter we shall have to skip  $2^4 - 10$  or 6 states. Which of these states are to be skipped is a matter of choice, which is largely governed by decisions which will make the circuit as simple as possible.

Many methods have been developed for designing such counters. We will consider the following:

##### (1) Counter Reset Method

In this method the counter is reset after the desired count has been reached and the count cycle starts all over again from the reset state.

##### (2) Logic Gating Method

This method provides the exact count sequence required without any need to reset the counter at some stage.

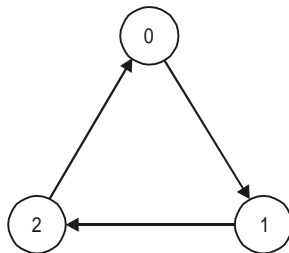
**(3) Counter Coupling Method**

This method is used to implement counters of the required modulus. For instance we can interconnect mod-2 and mod-3 counters to implement a modulus  $3 \times 2$  or mod-6 counter.

**7.4.10 Counter Reset Method (Asynchronous Counters)**

Let us first consider the typical case of a counter which has 3 states as shown in Fig. 7.28.

**7.4.10.1 Mod-3 Counter**



**Fig. 7.28** State diagram for a mod-3 counter

It is obvious that a mod-3 counter will require two flip-flops which, when connected as a counter, will provide four states as shown in Table 7.4.

**Table 7.4 States for a two flip-flop counter**

$Q_A$ <i>LSB</i>	$Q_B$	Count value <i>(Decimal)</i>
0	0	0
1	0	1
0	1	2
1	1	3
0	0	0

This counter counts in the binary sequence 0, 1, 2, 3 and then it returns to 0, the starting point. Each count is referred to as a state. If we are building a mod-3 counter, the most convenient solution is to skip state 3 and then return to state 0 from state 2 and then again go through states 0, 1, 2 before returning to state 0. What we need is a combinational logic circuit, which will feed a reset pulse to the counter during state 3, and immediately after state 2, which is the last desired state. This reset pulse is applied to the CLR inputs which resets the counter to 0 after state 2.

A circuit diagram for a mod-3 counter together with the required combinational logic is given in Fig. 7.29.

When both outputs  $Q_A$  and  $Q_B$  are 1, the output of the NAND gate, which provides the reset pulse, goes low and both the flip-flops are reset. The counter returns to state 0 and it starts counting again in 0, 1, 2, 0 sequence. The waveforms for this counter are given in Fig. 7.30.

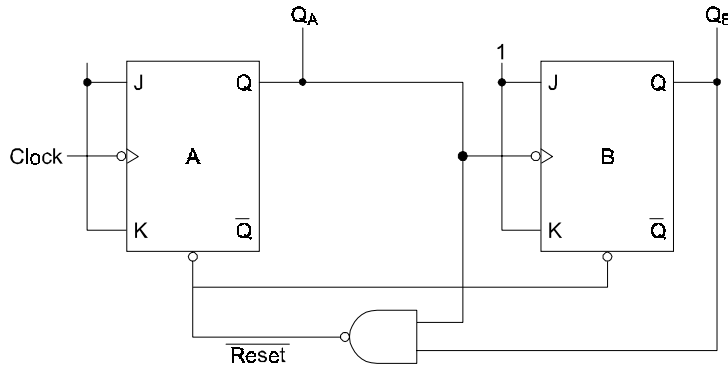


Fig. 7.29 Modulo-3 counter

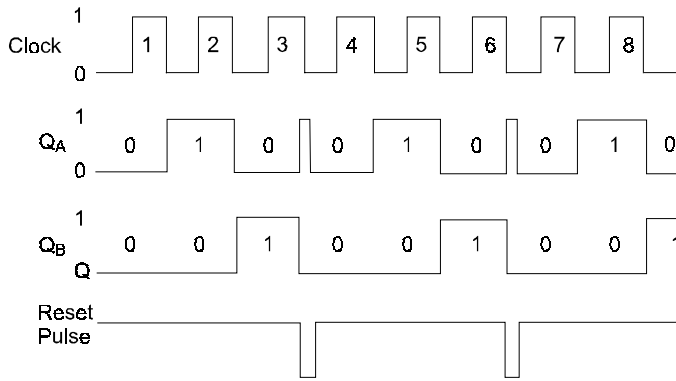


Fig. 7.30 Waveform for Mod-3 counter

7.4.10.2 Mod-5 Counter

The minimum number of flip-flops required to implement this counter is three. With three flip-flops, the number of states will be 8. A modulo-5 counter will have only 5 states. A state diagram for this counter is given in Fig. 7.31. It will progress from state 000 through 100. The truth table for this counter, which will determine the stage at which the reset pulse should be applied, is given in Table 7.5.

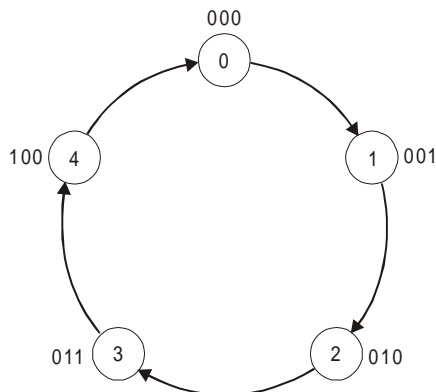


Fig. 7.31 State diagram for Mod-5 counter

The truth table shows that state 5 will be the reset state and that states 6 and 7 will be the don't care states. The next step is to plot the states on a map as shown in Fig. 4.39.

**Table 7.5 Truth table for Mod-5 Counter**

$Q_A$ <i>LSB</i>	$Q_B$	$Q_C$	State
0	0	0	0
1	0	0	1
0	1	0	2
1	1	0	3
0	0	1	4
1	0	1	5
0	1	1	6 X
1	1	1	7 X

X, Don't care states

	$\overline{B} \overline{C}$ 00	$\overline{B} C$ 01	$B C$ 11	$B \overline{C}$ 10
$\overline{A} \ 0$	0 0	0 4	X 6	0 2
$A \ 1$	0 1	1 5	X 7	0 3

**Fig. 7.32**

The map shows that the reset pulse is determined by  $R = Q_A \cdot \overline{Q_B} \cdot Q_C$ . The logic diagram for this counter is given in Fig. 7.33. The diagram shows that a reset pulse will be applied when both A and C are 1. You may have noticed that the reset pulse shown in Fig. 7.30 for the Mod-3 counter was very narrow and in some cases it may not be suitable to control other logic devices associated with the counter. The Mod-5 counter circuit Fig. 7.33 incorporates an RS flip-flop, which produces a reset pulse, the width of which is equal to the duration for which the clock pulse is low. The way it works is like this. State 5 is decoded by gate D, its output goes low, the RS flip-flop is set, and output  $\overline{Q}$  goes low, which resets all the flip-flops. The leading edge of the next clock pulse resets the RS flip-flop,  $\overline{Q}$  goes high which removes the reset pulse. The counter thus remains reset for the duration of the low time of the clock pulse. When the trailing edge of the same clock pulse arrives, a new cycle is started. The waveform for Mod-5 counter is given in Fig. 7.34.

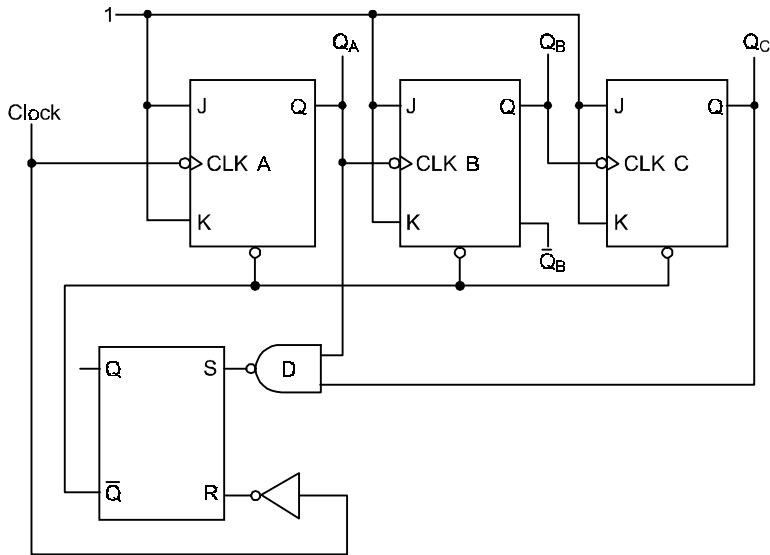


Fig. 7.33 Modulus-5 counter

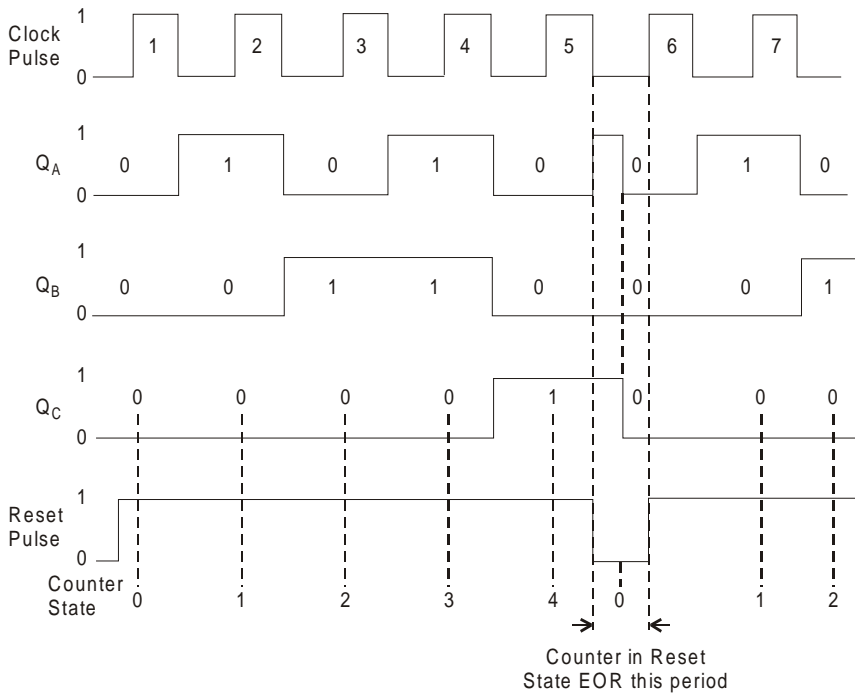
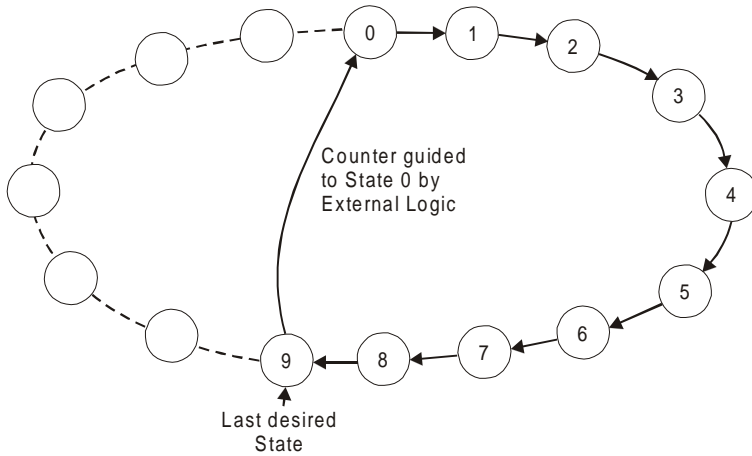


Fig. 7.34 Waveform for Modulus-5 asynchronous counter

### 7.4.10.3 Mod-10 (Decade) Counter

The decade counter discussed here is also an asynchronous counter and has been implemented using the counter reset method. As the decade counter has ten states, it will require four flip-flops to implement it. A state diagram for this counter is given in Fig. 7.35 and the truth table is given in Table 7.6.





**Fig. 7.35** State diagram for decade counter

**Table 7.6 Truth table for decade counter**

$Q_A$ LSB	$Q_B$	$Q_C$	$Q_D$	State
0	0	0	0	0
1	0	0	0	1
0	1	0	0	2
1	1	0	0	3
0	0	1	0	4
1	0	1	0	5
0	1	1	0	6
1	1	1	0	7
0	0	0	1	8
1	0	0	1	9
0	1	0	1	10
1	1	0	1	11 X
0	0	1	1	12 X
1	0	1	1	13 X
0	1	1	1	14 X
1	1	1	1	15 X

The table shows that state 9 will be the last desired state and state 10 will be the reset state. State 11, 12, 13, 14 and 15 will be the don't care states. The next step is to plot the states on a map to determine the reset pulse. This has been done in Fig. 7.36.

The map shows that the reset pulse is determined by the following expression:

$$R = \bar{Q}_A \cdot Q_B \cdot \bar{Q}_C \cdot Q_D$$

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

Fig. 7.36

The decade counter circuit Fig. 7.37 is essentially a binary ripple counter, which can count from 0000 to 1111; but since a decade counter is required to count only from 0000 to 1001, a reset pulse is applied at count 10 when the counter output is  $\overline{Q}_A \cdot \overline{Q}_B \cdot \overline{Q}_C \cdot \overline{Q}_D$ . In order to have control over the reset pulse width, a 4-input NAND gate is used to decode state 10.

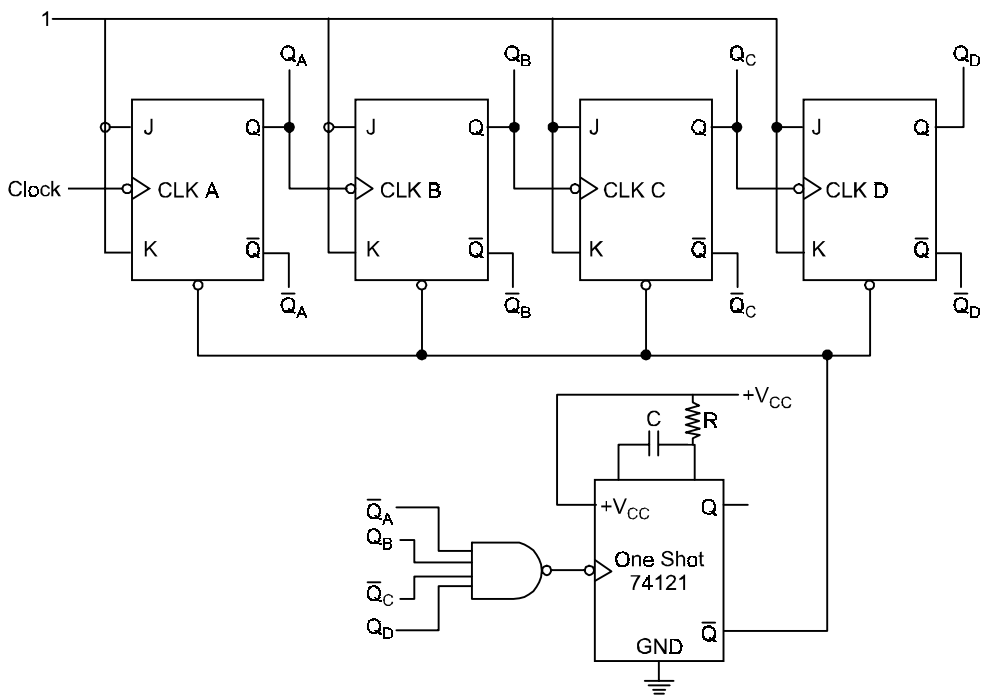
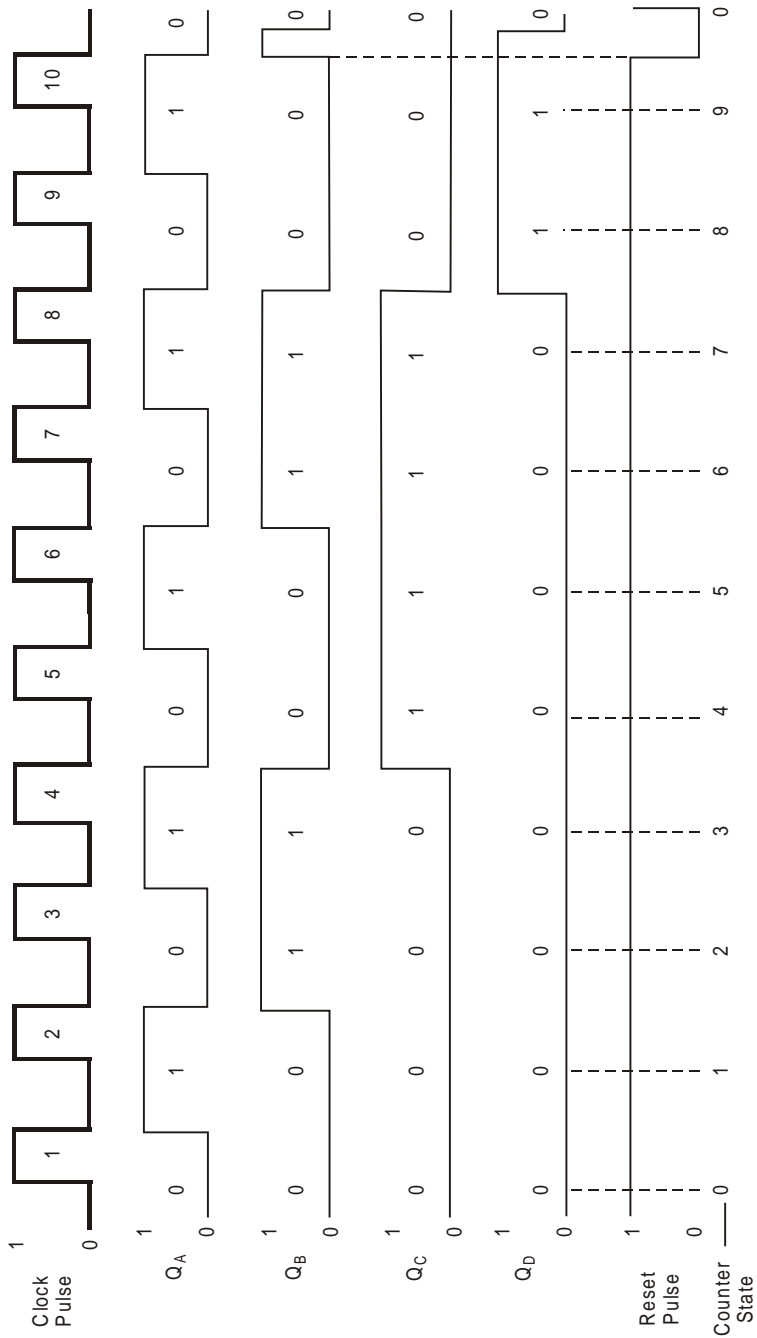


Fig. 7.37 Decade (Mod-10) asynchronous counter using count reset and pulse width control.

To decode count 10, logic inputs that are all one at the count of 10, are used to feed the NAND gate. At this count the NAND gate output goes low providing a  $1 \rightarrow 0$  change which triggers the one-shot unit. The  $\bar{Q}$  output of the one shot unit is used, as it is normally high and it goes low during the one-shot timing period, which depends on the RC constants of the circuit. The timing period of the one-shot can be adjusted, so that the slowest counter state resets. Although only A and D flip-flops need to be reset, the reset pulse is applied to all the flip-flop to make doubly sure that all flip-flops are reset.



**Fig. 7.38** Waveform for decade counter

Since decade (Modulus-10) counters have 10 discrete states, they can be used to divide the input frequency by 10. You will notice that at the output of the D-flip-flop, there is only one output pulse for every 10 input pulses. These counters can be cascaded to increase count capability.

The waveform for this counter is shown in Fig. 7.38.

#### 7.4.11 Logic Gating Method

The counter reset method of implementing counters, which we have discussed in the previous section, has some inherent drawbacks. In the first place, the counter has to move up to a temporary state before going into the reset state. Secondly, pulse duration timing is an important consideration in such counters, for which purpose special circuits have to be incorporated in counter design.

We will now consider another approach to counter design, which provides for the exact count sequence. We will discuss the design of some modulus counters to illustrate the procedures.

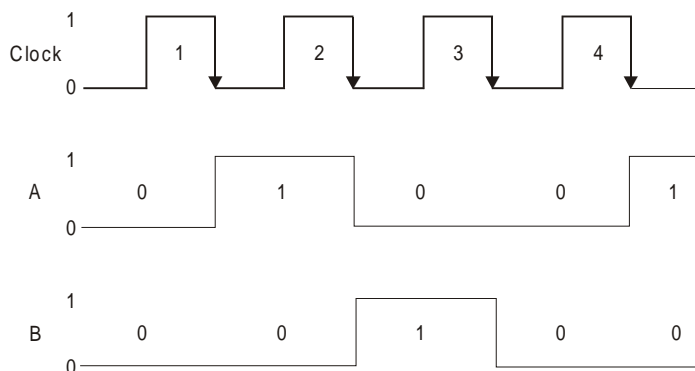
##### 7.4.11.1 Mod-3 Counter (Synchronous)

Let us suppose that we are required to design a modulo-3 counter which conforms to the truth table given in Table 7.7.

**Table 7.7 Truth table for Mod-3 Counter**

Input pulse count	Counter states	
	A	B
0	0	0
1	1	0
2	0	1
3	0	0

Based on this truth table, the output waveform for this Mod-3 counter should be as shown in Fig. 7.39.



**Fig. 7.39** Waveform for Mod-3 counter

You will notice from the waveform of the counter, that flip-flop A toggles on the trailing edge of the first and second pulses. Also observe that flip-flop B toggles only on the second

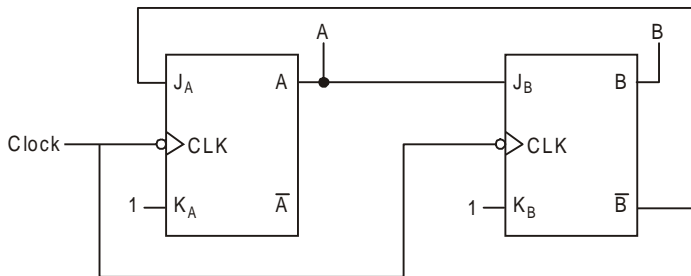
and third clock pulses. We have to bear this in mind, in figuring out logic levels for the J and K inputs of the flip-flops.

Suppose that initially both the flip-flops are reset. Since flip-flop A has to toggle when the trailing edges of the first and the second clock pulses arrive, its J and K inputs should be at logic 1 level during this period. This is achieved by connecting the K input to logic 1 level and the J input to the complement output of flip-flop B, as during this period the  $\bar{B}$  output of flip-flop B is at a high logic level. In this situation, the first clock pulse produces a logic 1 output and the second clock pulse produces a logic 0 output.

The J input of flip-flop B is connected to the normal output of flip-flop A. Therefore, when the first clock pulse arrives, the J input of flip-flop B is low. Its output will remain low as you will notice from the truth table and the waveform. The second pulse is required to toggle this flip-flop and its K input is, therefore held high. When the second clock pulse arrives, the flip-flop will toggle as both the J and K inputs are high. The output will go high. At the same time its complement output will be low, which makes the J input of flip-flop A low.

When the third clock pulse arrives, the output of flip-flop A will go low. Since after the second clock pulse the output of flip-flop A was already low, the third clock pulse produces a low output at flip-flop B. Both the A and B flip-flops are now reset and the cycle will be repeated.

A logic diagram for the Mod-3 counter is given in Fig. 7.40.



**Fig. 7.40** Mod-3 counter (Synchronous)

**7.4.11.2 Mod-5 Counter (Asynchronous)**

We will use the same procedure to design a mod-5 counter as before. The truth table required for this counter is given in Table 7.8.

**Table 7.8** Truth table for Mod-5 counter

Input pulse count	Counter states		
	A	B	C
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	0	0	0

The waveform for this counter based on this truth table is given in Fig. 7.41. You will notice from the truth table and the waveform that the A flip-flop complements each input pulse, except when the normal output of flip-flop C is logic 1, which is so after the trailing edge of the 4th, clock pulse. It, therefore, follows that the K input of flip-flop A should be a constant logic 1 and the J input should be connected to the complement output of flip-flop C will be 0 when C is 1 so that the output of flip-flop A remains low after the trailing edge of the 5th clock pulse.

If you observe the changing pattern of the output of the B flip-flop, you will notice that it toggles at each transition of the A output from 1 → 0. It is, therefore, obvious that the A output should be connected to the clock input of the B-flip-flop and the J and K inputs of this flip-flop should be at logic 1 level.

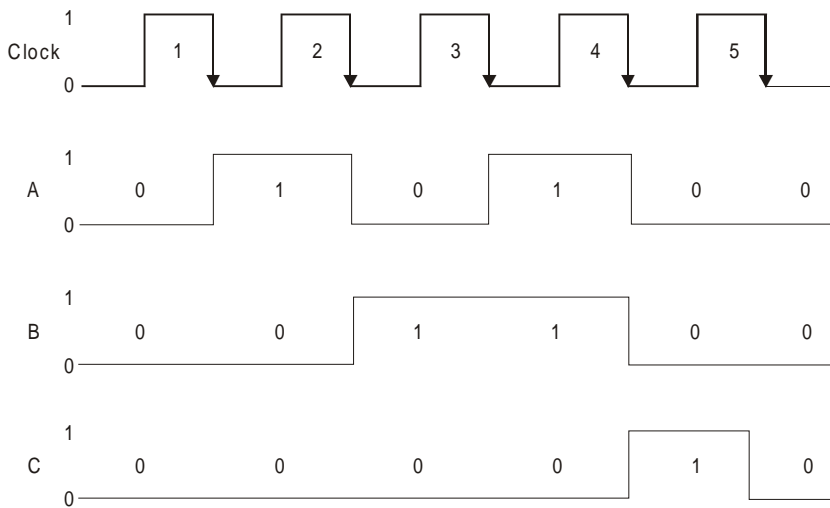


Fig. 7.41 Waveform for Mod-5 counter

After the 3rd clock pulse, the outputs of A and B flip-flops are 1. An AND gate is used to make the J input to flip-flop C as 1 when both A and B are 1. The K input to flip-flop C is also held at logic 1 to enable it to toggle. The clock is also connected to the clock input to flip-flop C, which toggles it on the 4th, clock pulse and its output becomes 1. When the 5th, clock pulse arrives, the J input to flip-flop C is 0 and it resets on the trailing edge of this clock pulse. Thereafter the cycles are repeated. The logic diagram for the mod-5 counter is given in Fig. 7.42.

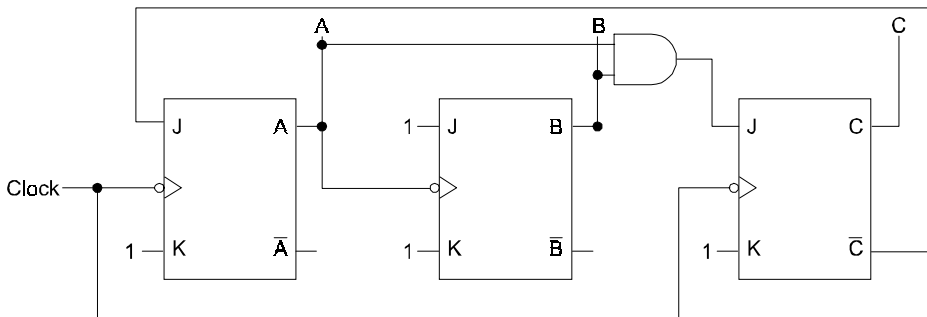


Fig. 7.42 Logic diagram for Mod-5 counter

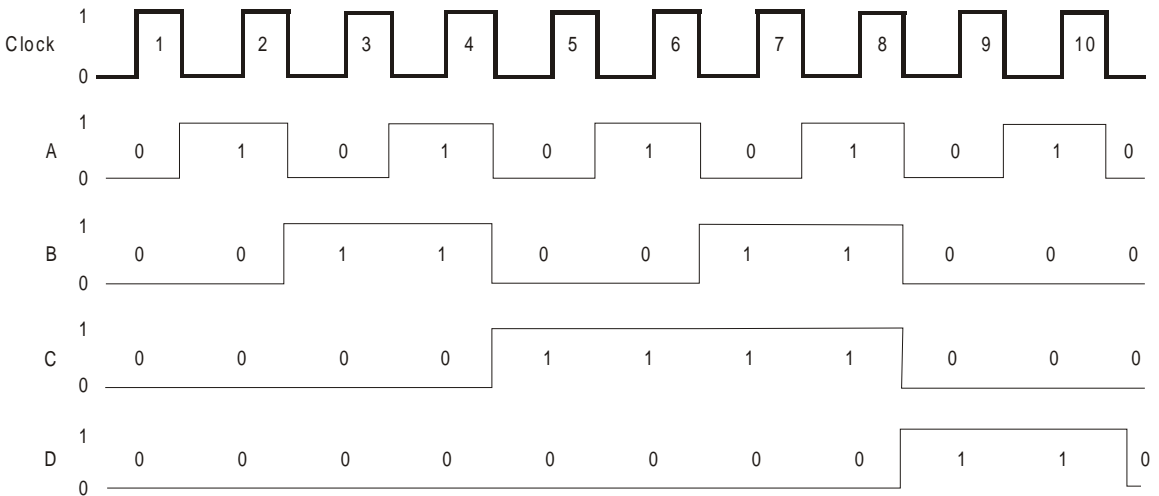
7.4.11.3 Mod-10 (Decade) Counter (Asynchronous)

The truth table for a Decade counter is given in Table 7.9.

**Table 7.9 Truth Table for Decade counter**

Input pulse count	Counter states			
	A	B	C	D
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10 (0)	0	0	0	0

The waveform for this counter based on this truth table is given in Fig. 7.43.



**Fig. 7.43** Waveform for Decade Counter

If you compare truth Table 7.9 for the Decade counter with Table 7.2 which gives the count-up sequence for a 4-bit binary up-counter, you will notice a close similarity between the two, up to input pulse 8. You will also notice a close resemblance between waveforms of Fig. 7.43 and Fig. 7.15 up to a certain point.

The count ripples through the A, B and C flip-flops for the first seven input pulses, as in the standard 4-bit binary up-counter. At this point the counter will show an output of 1 1 1 0 (decimal 7). On the application of the 8th pulse, flip-flops A, B and C must reset and the D output should be 1, that is the counter state should change from 1 1 1 0 to 0 0 0 1. In order that the J input to flip-flop D is 1, so that when K is 1 the D flip-flop output goes from 0 to

1; B and C outputs are applied to the input of an AND gate and its output goes to the J input. In order that the B and C outputs are 0, when D output is 1 for the 8th and the 9th count, the complement output of the D flip-flop which will be 0 when D is 1, is connected to the J input of the B flip-flop.

After the trailing edge of the 8th pulse D becomes 1 and A, B and C become 0, the 9th pulse is required to change the output from 0 0 0 1 to 1 0 0 1. Since no change is required in the D output, the D-flip-flop is triggered by the A output. When the 9th pulse arrives, the A output changes from 0 to 1, but this causes no change in the D output. When the 10th input pulse arrives, it changes the A output from 1 to 0, which changes the D output from 1 to 0. The counter output changes from 1 0 0 1 to 0 0 0 0. During the 9th and the 10th pulses, the B and C outputs will remain unchanged.

A logic diagram for the Decade counter is given in Fig. 7.44.

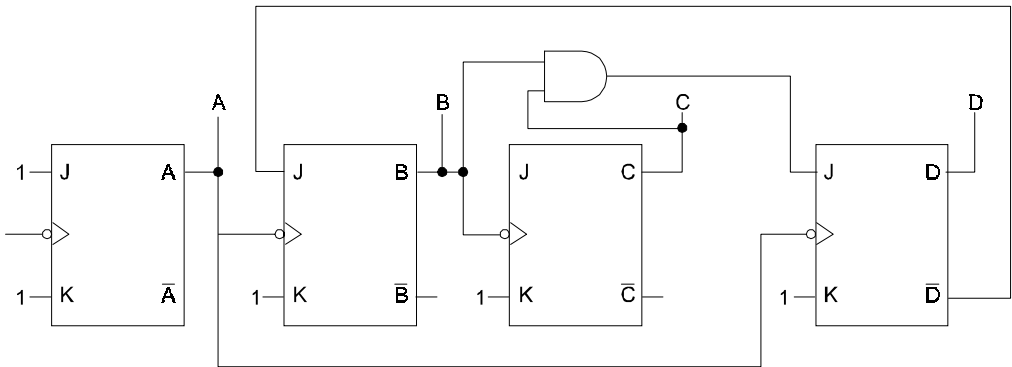


Fig. 7.44 Logic diagram for Decade counter

### 7.4.12 Design of Synchronous Counters

In most of the counter designs we have considered so far, the flip-flops are not triggered simultaneously. In synchronous counters all stages are triggered at the same time. The output of each stage depends on the gating inputs of the stage. If you refer to previous counter designs, you will observe that the gating inputs have been assigned values to give the desired outputs.

The basic framework of a synchronous counter would be somewhat like the partial logic diagram given in Fig. 7.45. You will notice that all the clock inputs are connected to a common line and the J and K inputs of the flip-flops have been left open. They are required to have the values necessary to give the required outputs after each input pulse. The J and K inputs of each flip-flop are therefore required to have the values which produce the desired counter states at each input pulse. The entire purpose of the exercise is to determine the input values for each stage. A typical design procedure can be summed up in the following steps.

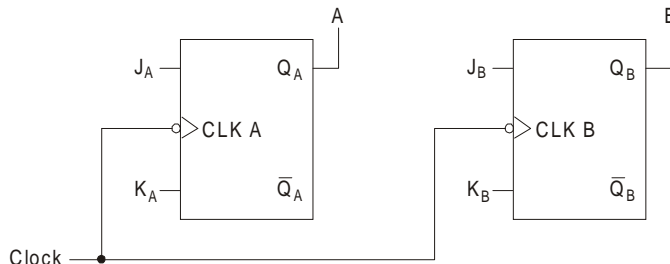


Fig. 7.45



- Write the desired truth table for the counter.
- Write the counter transition table which should list the starting state and the subsequent states the counter is required to take up.
- With the help of the excitation table and using the counter transition table, write down the input values for the J and K inputs to enable each flip-flop to attain the output state as required by the transition table.
- Prepare Karnaugh maps for the J and K inputs of each stage.
- Derive Boolean algebra expressions for each of the inputs to the flip-flops.
- Draw the synchronous counter circuit incorporating the J and K input values obtained from the above steps.

We will take up a specific case to illustrate the above procedure.

#### 7.4.12.1 Mod-3 Synchronous Counter

We have implemented a Mod-3 synchronous counter as described in Sec. 7.4.11.1. We will implement the same counter by the procedure described here. We will follow the truth table given in Table 7.7. For your convenience the excitation table for JK flip-flops is reproduced here.

**Table 7.10 Excitation table for JK flip-flop**

<i>Present state</i>	<i>Next state</i>	<i>J</i>	<i>K</i>
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

We now prepare a counter design table listing the two flip-flops and their states and also the four inputs to the two flip-flops as in table 7.11.

**Table 7.11 Counter design table**

<i>Counter state</i>		<i>Flip-flop inputs</i>			
<i>A</i>	<i>B</i>	<i>A</i>		<i>B</i>	
		<i>J<sub>A</sub></i>	<i>K<sub>A</sub></i>	<i>J<sub>B</sub></i>	<i>K<sub>B</sub></i>
0	0	1	X	0	X
1	0	X	1	1	X
0	1	0	X	X	1
0	0				

The table shows that if the counter is in the state  $A = 0, B = 0$  and a clock pulse is applied, the counter is required to step up to  $A = 1, B = 0$ . When the counter is in the state  $A = 1, B = 0$  and a clock pulse is applied, the counter has to step up to  $A = 0, B = 1$ . Lastly when another clock pulse is applied the counter has to reset.

From the excitation table for JK flip-flops we can determine the states of the J and K inputs, so that the counter steps up as required. For instance for the A flip-flop to step up from 0 to 1,  $J_A$  should be 1 and  $K_A$  should be X. Similarly the J and K input values of both the flip-flops for the remaining counter states have been worked out as shown in the table.

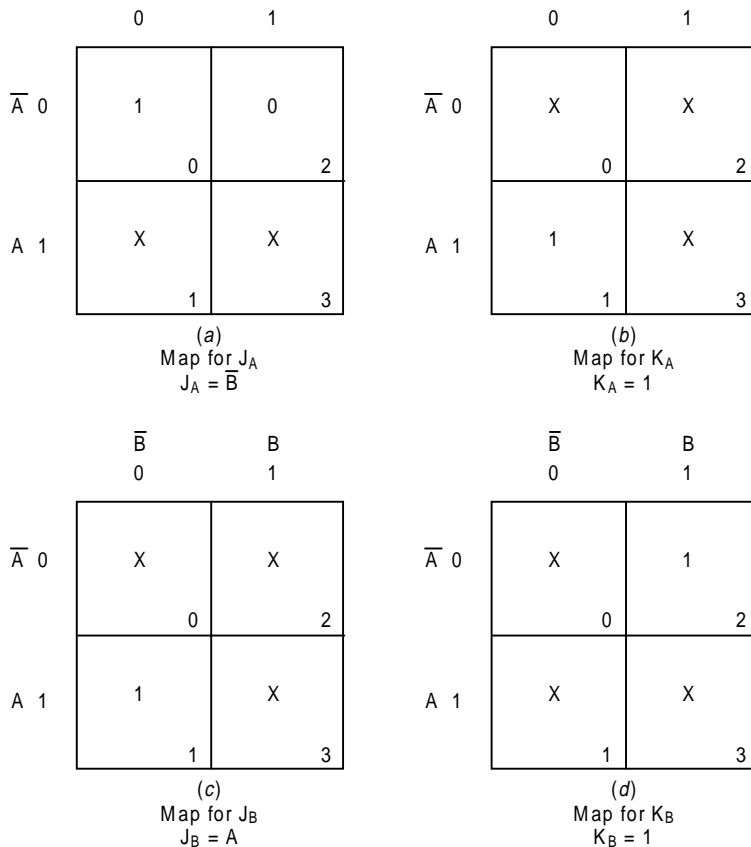
The next step is to derive boolean algebra expressions for each of the inputs to the flip-flops. In the above exercise, our effort was to generate flip-flop inputs in a given row, so that

when the counter is in the state in that row, the inputs will take on the listed values, so that the next clock pulse will cause the counter to step up to the counter state in the row below.

We now form boolean algebra expressions from this table for the  $J_A$ ,  $K_A$ ,  $J_B$  and  $K_B$  inputs to the flip-flops and simplify these expressions using Karnaugh maps. Expressions for these inputs have been entered in Karnaugh maps in Fig. 7.46 (a), (b), (c) and (d). The simplified expressions obtained for the inputs are also indicated under the maps.

The counter circuit when drawn up with the following resultant data will be the same as worked out before in Fig. 7.40.

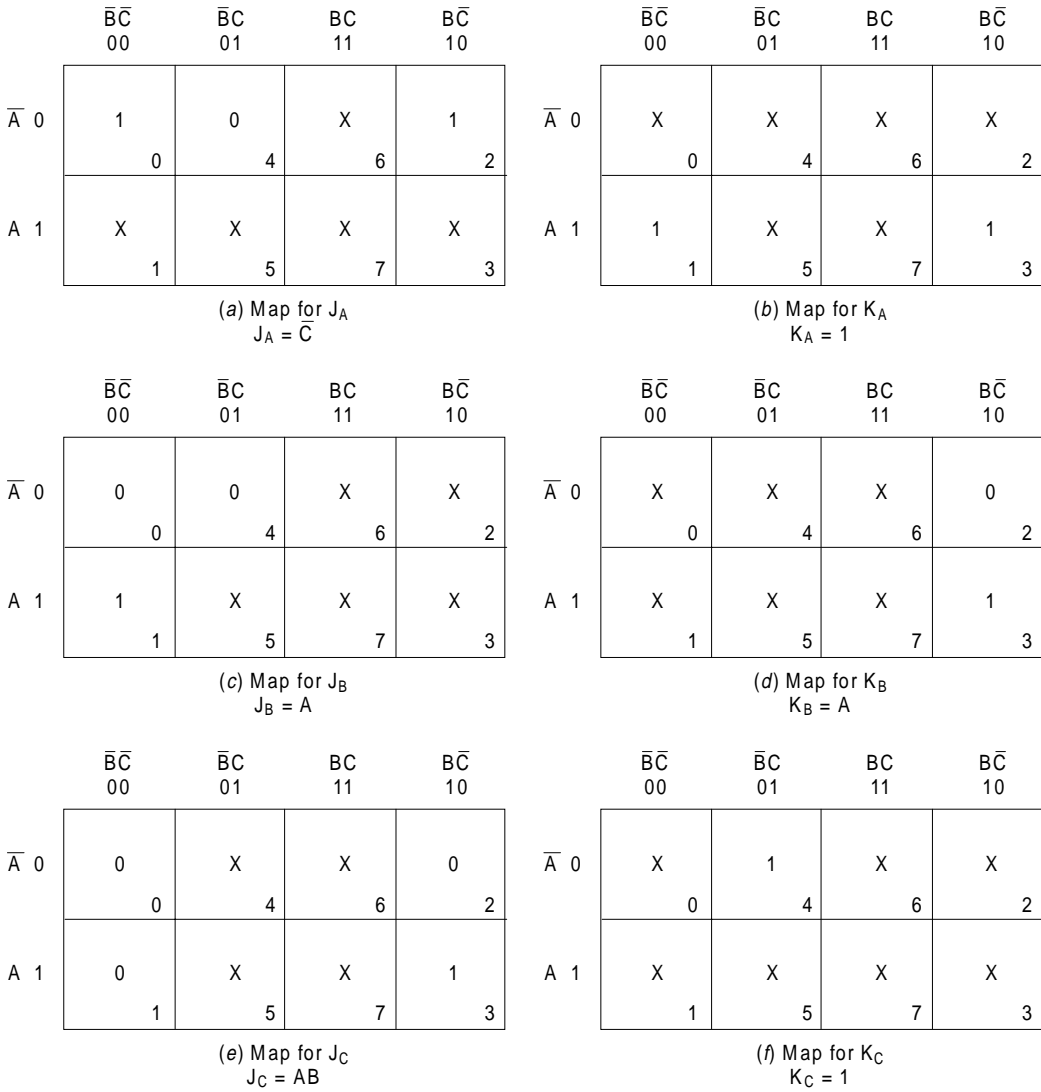
$$\begin{aligned}
 J_A &= \bar{B} \\
 K_A &= 1 \\
 J_B &= A \\
 K_B &= 1
 \end{aligned}$$



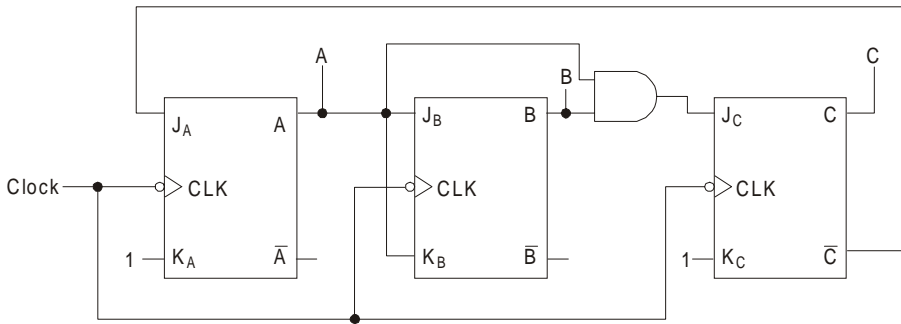
**Fig. 7.46** (a), (b), (c) and (d)

#### 7.4.12.2 Mod-5 Counter (Synchronous)

The Mod-5 counter we are going to implement will be a synchronous counter, but it will have the same counter states as given earlier in Table 7.8. The counter design table for this counter lists the three flip-flops and their states as also the six inputs for the three flip-flops. The flip-flop inputs required to step up the counter from the present to the next state have been worked out with the help of the excitation table (Table 7.10). This listing has been shown in Table 7.12.



**Fig. 7.47**



**Fig. 7.48** Synchronous Mod-5 counter

**Table 7.12 Counter design table for Mod-5 counter**

Input pulse count	Counter states			Flip-flop inputs					
	A	B	C	$J_A$	$K_A$	$J_B$	$K_B$	$J_C$	$K_C$
0	0	0	0	1	X	0	X	0	X
1	1	0	0	X	1	1	X	0	X
2	0	1	0	1	X	X	0	0	X
3	1	1	0	X	1	X	1	1	X
4	0	0	1	0	X	0	X	X	1
5 (0)	0	0	0						

The flip-flop inputs have been determined with the help of the excitation table. Table 7.10. Some examples follow:

#### A flip-flop

The initial state is 0. It changes to 1 after the clock pulse. Therefore  $J_A$  should be 1 and  $K_A$  may be 0 or 1 (that is X).

#### B flip-flop

The initial state is 0 and it remains unchanged after the clock pulse. Therefore  $J_B$  should be 0 and  $K_B$  may be 0 or 1 (that is X).

#### C flip-flop

The state remains unchanged. Therefore  $J_C$  should be 0 to  $K_B$  should be X.

The flip-flop input values are entered in Karnaugh maps Fig. 7.47 [(a), (b), (c), (d), (e) and (f)] and a boolean expression is formed for the inputs to the three flip-flops and then each expression is simplified. As all the counter states have not been utilized, Xs (don't) are entered to denote un-utilized states. The simplified expressions for each input have been shown under each map. Finally, these minimal expressions for the flip-flop inputs are used to draw a logic diagram for the counter, which is given in Fig. 7.48.

#### 7.4.12.3 Mod-6 Counter (Synchronous)

The desired counter states and the JK inputs required for counter flip-flops are given in the counter design table (Table 7.13).

**Table 7.13 Counter design table for Mod-6 counter**

Input pulse count	Counter states			Flip-flop inputs					
	A	B	C	$J_A$	$K_A$	$J_B$	$K_B$	$J_C$	$K_C$
0	0	0	0	1	X	0	X	0	X
1	1	0	0	X	1	1	X	0	X
2	0	1	0	1	X	X	0	0	X
3	1	1	0	X	1	X	1	1	X
4	0	0	1	1	X	0	X	X	0
5	1	0	1	X	1	0	X	X	1
6 (0)	0	0	0						

As before, the JK inputs required for this have been determined with the help of the excitation table, (Table 7.10). These input values have been entered in Karnaugh maps Fig. 7.49 and a boolean expression is formed for the inputs to the three flip-flops and then each expression is simplified. Xs have been entered in those counter states which have not been utilized. The simplified expressions for each input have been shown under each map and finally a logic diagram based on these expressions has been drawn, as given in Fig. 7.50.

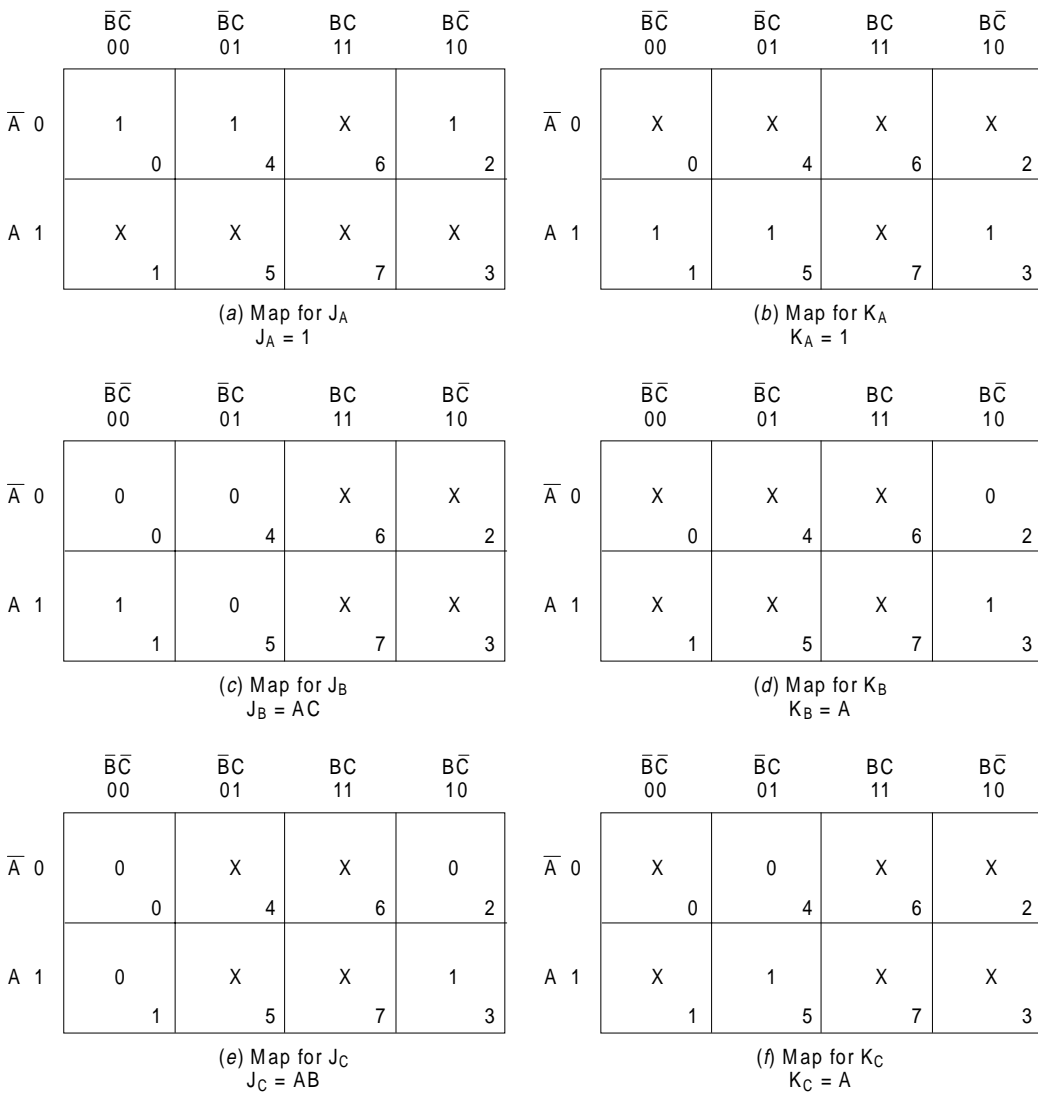


Fig. 7.49

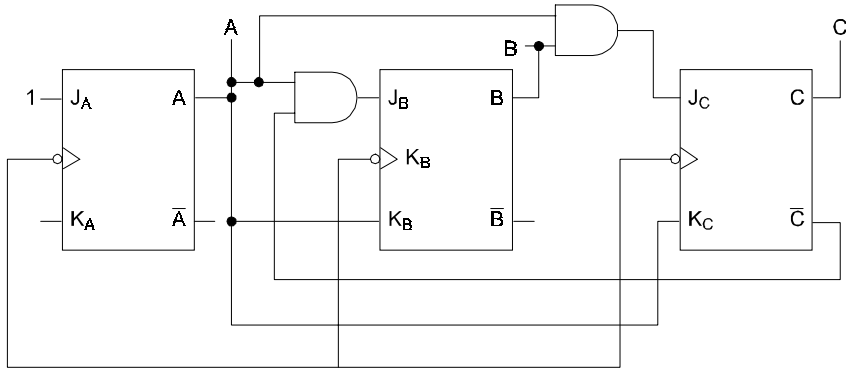


Fig. 7.50 Synchronous Mod-6 counter.

7.4.13 Lockout

The mod-6 counter we have just discussed utilizes only six out the total number of eight states available in a counter having three flip-flops. The state diagram for the mod-6 counter given in Fig. 7.51, shows the states which have been utilized and also states 011 and 111 which have not been utilized. The counter may enter one of the unused states and may keep shuttling between the unused states and not come out of this situation. This condition may develop because of external noise, which may affect states of the flip-flops. If a counter has unused states with this characteristic, it is said to suffer from lockout.

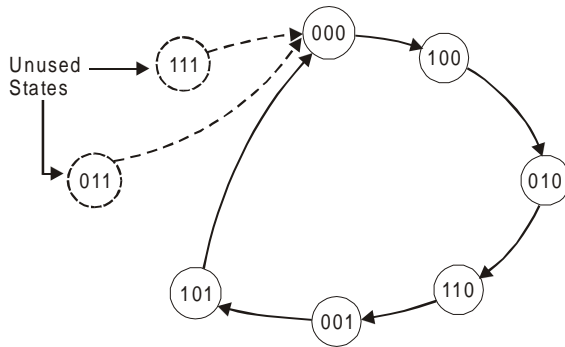


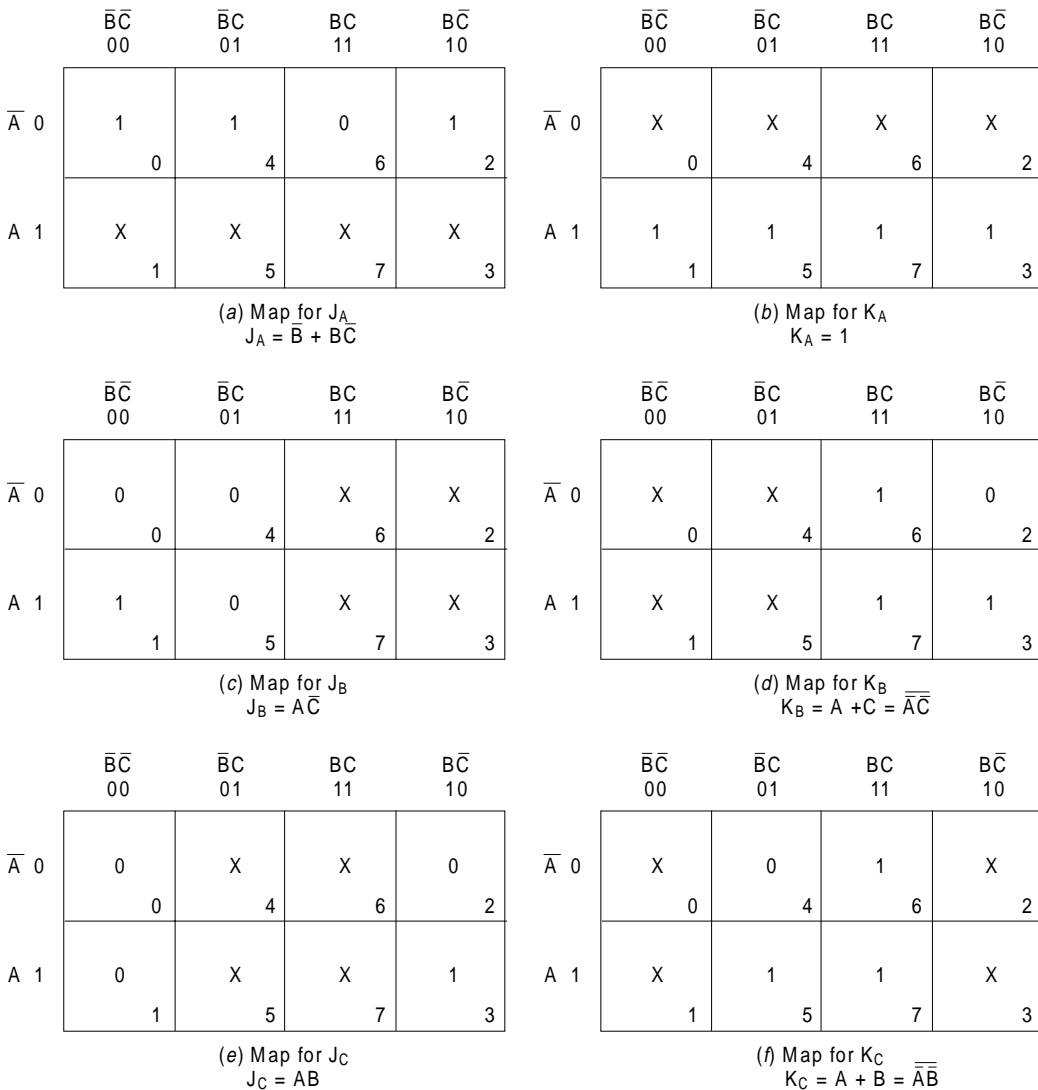
Fig. 7.51 State diagram for Mod-6 counter.

The lockout situation can be avoided by so arranging the circuit that whenever the counter happens to be in an unused state, it reverts to one of the used states. We will redesign the mod-6 counter so that whenever it is in state 0 1 1 or 1 1 1, the counter switches back to the starting point 0 0 0. You will notice from Fig. 7.49 that Js and Ks were marked X in squares corresponding to the unused states. We will now assign values for Js and Ks for the unused states, so that the counter reverts to state 0 0 0. This has been done in Table 7.14.

**Table 7.14**

Counter states			Flip-flop inputs					
A	B	C	$J_A$	$K_A$	$J_B$	$K_B$	$J_C$	$K_C$
0	1	1	0	X	X	1	X	1
1	1	1	X	1	X	1	X	1
0	0	0						

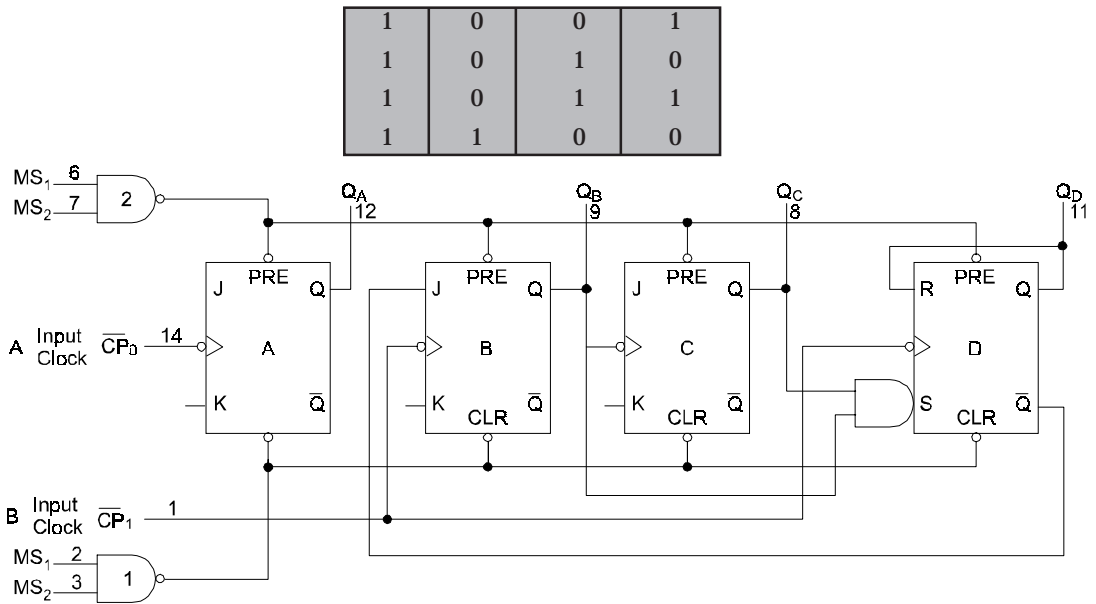
These values of Js and Ks have been entered in K-maps for those counter states where Xs had been entered previously. K-maps for the revised values of Js and Ks are given in Fig. 7.52. Boolean expressions are formed for the inputs to the three flip-flops and the expressions so obtained are simplified. The expressions for each input have been shown under each map and the logic diagram for the improved mod-6 counter is given in Fig. 7.53.



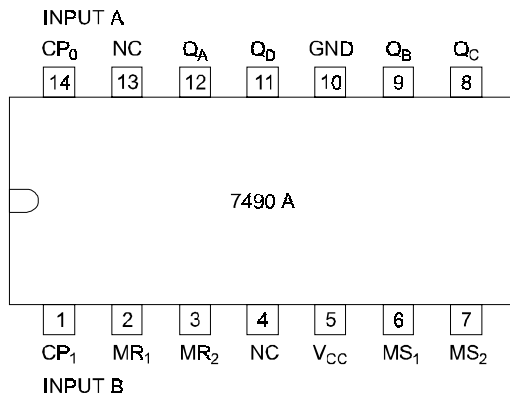
**Fig. 7.52**



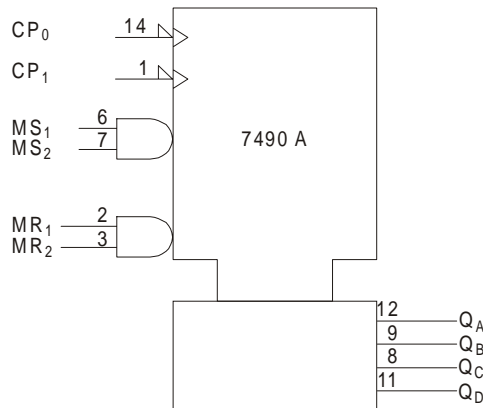




**Fig. 7.54** Logic diagram for counter IC 7490 A



**Fig. 7.55** Pin connections for counter IC 7490 A

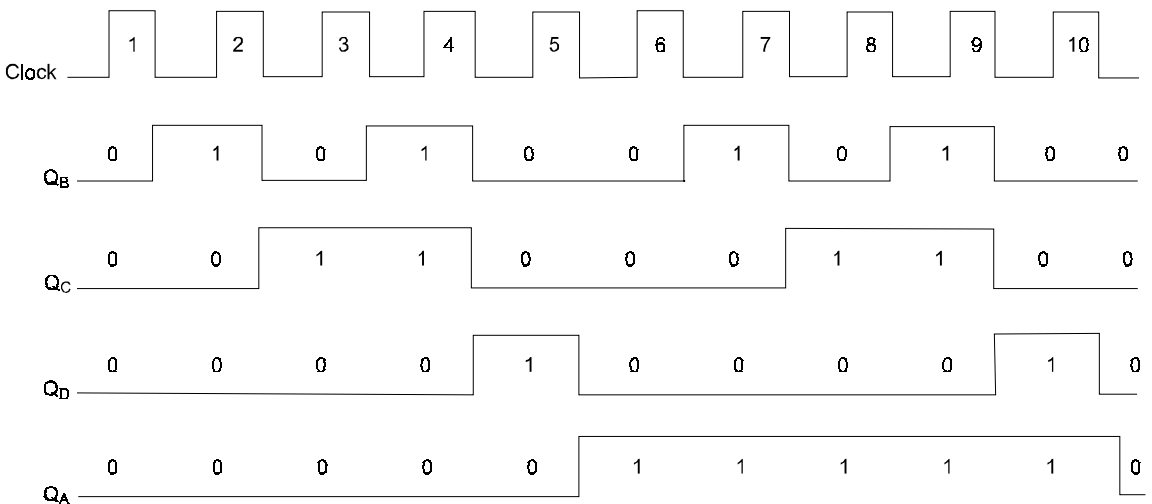


**Fig. 7.56** Logic symbol for counter IC 7490 A

When this counter is used as a decade counter in the  $2 \times 5$  configuration,  $Q_A$  (Pin 12) is connected to pin 1 as in Fig. 7.57 and clock pulse are applied at pin 14. The output at  $Q_D$  (Pin 11) will be low from a count of 0000 to 1110. The output at pin 11 will be high at counts of 0001 and 1001 or decimal 9. At the next 1 to 0 transition of the clock pulse, the counter will be reset and at the same time, if there is another counter in cascade as shown in Fig. 7.58 its count will go up from 0000 to 1000 or decimal 1 on the  $1 \rightarrow 0$  transition of the same clock pulse. In other words, when the first counter has reached its maximum count of 9, the next pulse will reset it to 0 and the second counter will be stepped up from 0000 to 1000 and the two put together will show a count of 10 and a maximum count of 99.

The waveform for this counter, connected in the  $2 \times 5$  configuration, will be as shown in Fig. 7.43 and the counting sequence will be as shown in Table 7.9, which is in pure binary sequence. If you look at the D output in Fig. 7.43, you will observe that it is not symmetrical. If this counter is connected in the  $5 \times 2$  mode, the output will have a symmetrical shape.

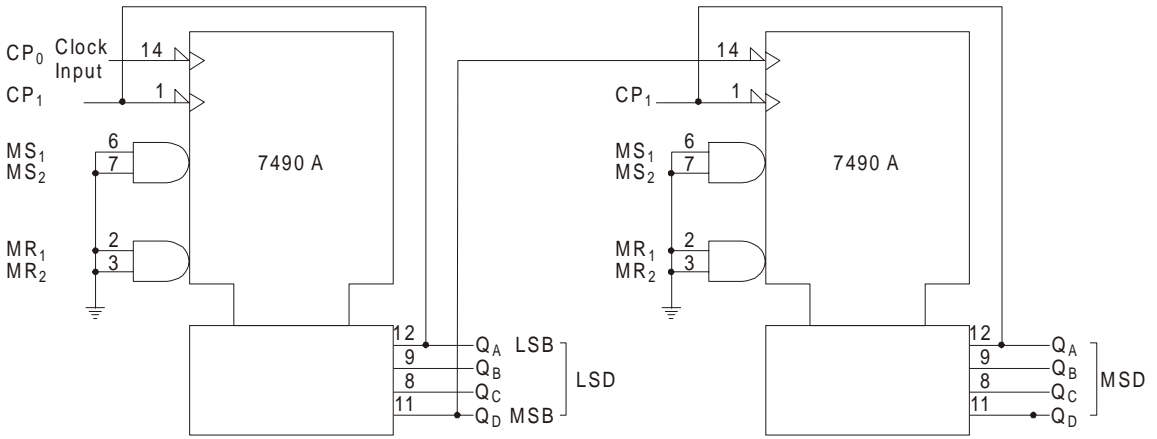
To operate this counter in the  $5 \times 2$  mode, pin 14 is connected to  $Q_D$  (pin 11) and the clock signal is applied at pin 1 as has been shown in Fig. 7.59. The output will now follow the bi-quinary ( $5 \times 2$ ) sequence as show in Table 7.15, which is different from the pure binary sequence in Table 7.9. Bi-quinary ( $5 \times 2$ ) sequence waveforms are shown in Fig. 7.60. The counter will show a count of 0001 after the first clock pulse and the 10th pulse will reset the counter to 0000 on its trailing edge, which will also increment the next counter if another counter is connected in cascade. Two counters connected in cascade in the  $5 \times 2$  configuration are shown in Fig. 7.61.



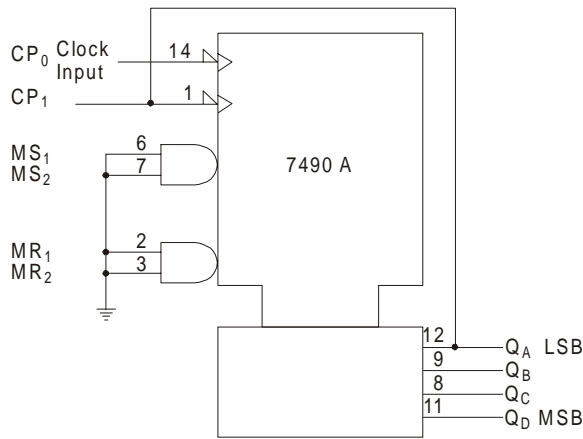
**Fig. 7.57** IC 7490 A connected as a decade counter ( $2 \times 5$ ) configuration

For two decade counters connected in cascade, there will be only one output pulse for every ten input clock pulses, which shows that the frequency of the train of input pulses is scaled down by a factor of 10. In other words if the input frequency is  $f$ , the output frequency will be  $f/10$ .

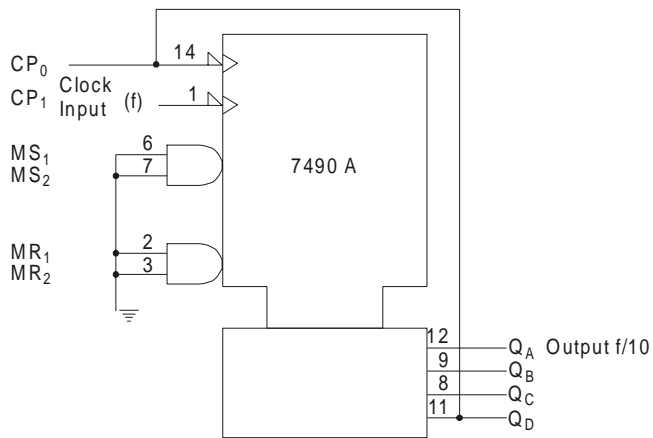
In digital instruments, it is often necessary to divide a frequency by a given factor and for this function scalars are used. Scalars will accept input pulses and output a single pulse at the required interval. A single decade scalar will divide a train of pulses by 10 and four decade scalars will divide the input frequency by  $10^4$ .



**Fig. 7.58** IC 7490 A connected as a two-decade counter (00–99)



**Fig. 7.59** IC 7490 A connected as a decimal scaler ( $5 \times 2$ ) configuration



**Fig. 7.60** Waveforms in bi-quinary ( $5 \times 2$ ) sequence

7.4.14.1 Modulo-N counters based on IC 7490 A

IC 7490 A has found several applications in circuits requiring frequency division. Some of the circuits in common use based on this IC have been discussed here.

Modulo-5 Counter

When used as a Mod-5 counter, connections are to be made as shown in Fig. 7.62. The count sequence for this counter is as given in Table 7.16.

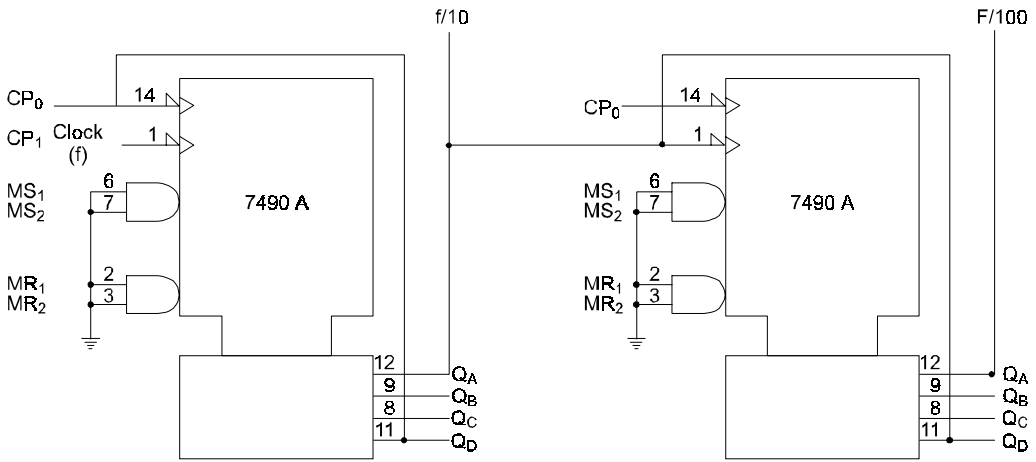


Fig. 7.61 Divide by 100 scaler

Table 7.16 Count sequence for Mod-5 counter

Input pulse count	Counter output		
	$Q_D$	$Q_C$	$Q_B$
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5 (0)	0	0	0

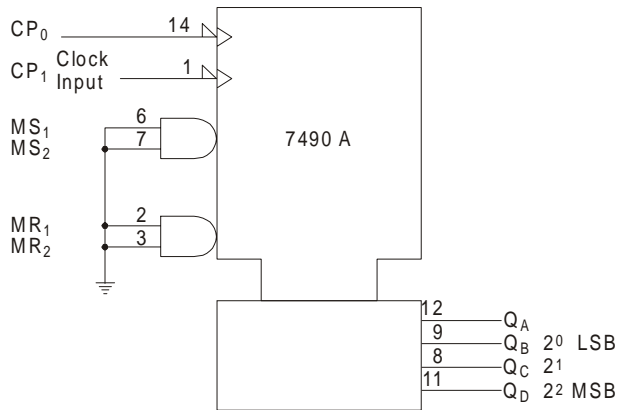


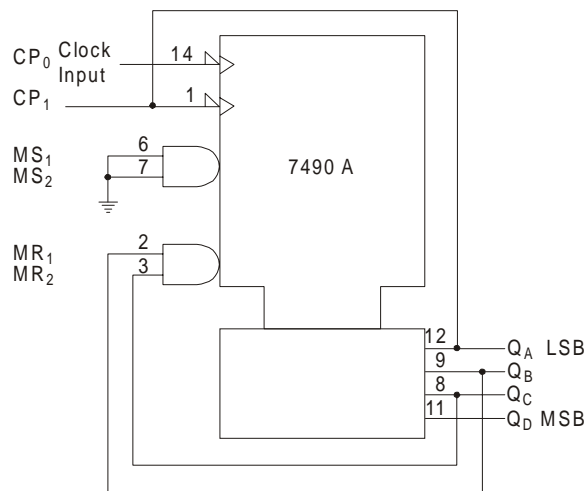
Fig. 7.62 Mod-5 counter using IC 7490 A

**Modulo-6 Counter**

IC 7490 A is to be connected as in Fig. 7.63 to obtain a Mod-6 counter. Its counter sequence is given in Table 7.17.

**Table 7.17 Count sequence for Mod-6 counter**

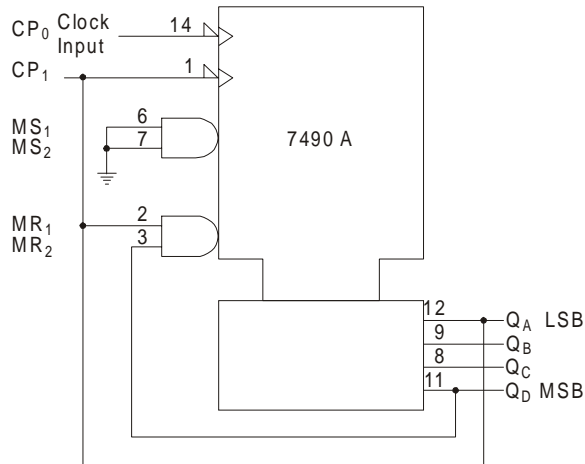
Input pulse count	Counter output			
	$Q_D$	$Q_C$	$Q_B$	$Q_A$
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6 (0)	0	1	1	0

**Fig. 7.63** Mod-6 counter using IC 7490 A

As the counter is required to reset when the count reaches 6, that is 0 1 1 0 when both  $Q_B$  and  $Q_C$  outputs are high, these two outputs are connected to the reset inputs so that the counter resets at this count. Thus the counter sequences from 0000 to 0101 and thereafter it resets and the cycle is repeated.

**Modulo-9 Counter**

When used as a Mod-9 counter, this IC is required to be connected as shown in Fig. 7.64. The counter is required to reset when the count reaches 1001. Therefore pins 11 and 12 are connected to pins 3 and 2. Also notice that pin 12 is connected to pin 1 and clock input is applied to pin 14, so that it looks like a decade counter which resets when the count reaches decimal 9.



**Fig. 7.64** Mod-9 counter using IC 7490A

#### 7.4.15 MSI Counter IC 7492A

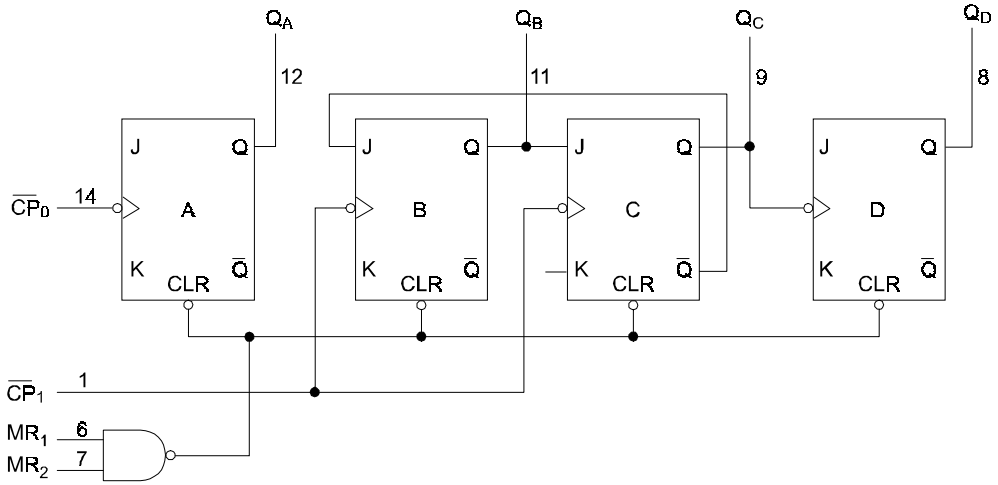
Counter IC 7492 A, which is very similar to IC 7490 A, also finds considerable application in circuits requiring frequency division. A logic diagram of this IC is given in Fig. 7.65 and its pin connections and logic symbol are given in Fig. 7.66 and 7.67 respectively.

In this counter flip-flops B, C, and D are connected in the  $3 \times 2$  configuration and, therefore, if input is applied at pin 1, and outputs are taken from  $Q_B$ ,  $Q_C$  and  $Q_D$ , this counter will function as mod-6 counter.

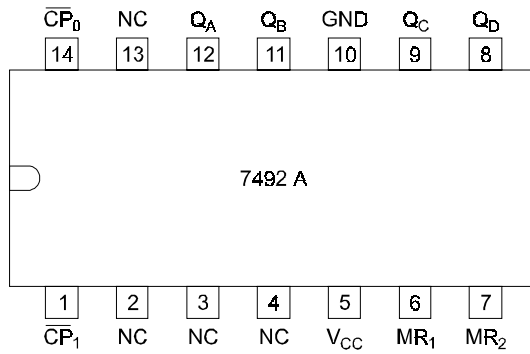
If output  $Q_A$ , pin 12, is connected to input pin 1, this IC functions as a  $2 \times 3 \times 2$  or mod-12 counter. Table 7.18 gives the truth table for this IC when  $Q_A$ , pin 12, is connected to input, pin 1. Some frequency division circuits based on this IC have been considered here.

**Table 7.18**

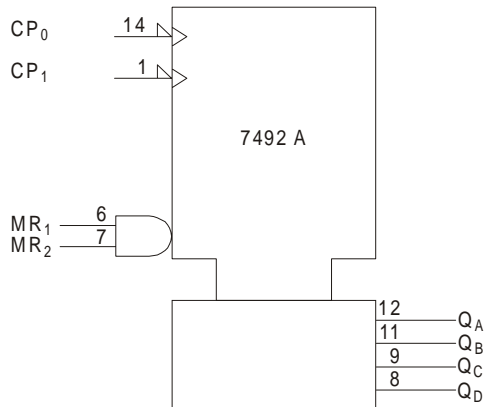
Input pulse count	Counter output			
	$Q_D$	$Q_C$	$Q_B$	$Q_A$
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	1	0	0	0
7	1	0	0	1
8	1	0	1	0
9	1	0	1	1
10	1	1	0	0
11	1	1	0	1
12	0	0	0	0



**Fig. 7.65** Logic diagram for IC 7492 A



**Fig. 7.66** Pin connections for IC 7492 A



**Fig. 7.67** Logic symbol for IC 7492 A

7.4.15.1 Divide-by-N circuits based on IC 7492 A

This IC like 7490 A is also a useful tool for circuits which require frequency division. Some circuits based on this IC have been considered here.

Divide-by-6-Circuit

As has been mentioned earlier, flip-flops B, C and D in this IC are connected in the  $3 \times 2$  configuration. To operate as a divide-by-6, circuit input is applied at pin 1 and output is taken from  $Q_D$ , pin 8, as shown in Fig. 7.68.

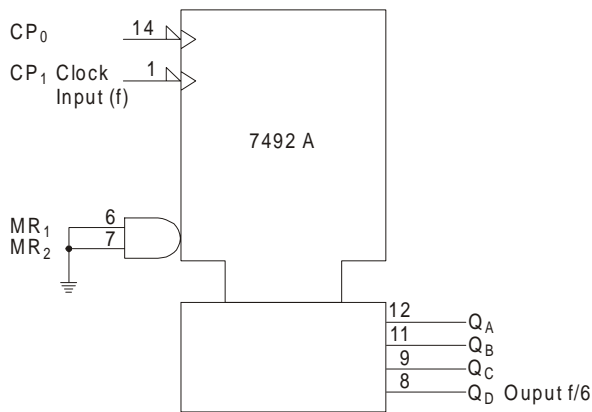


Fig. 7.68 Divide-by-6 circuit

Divide-by-9 Circuit

Fig. 7.68 gives the diagram for a circuit using IC 7492 A, which divides the input frequency by 9. Input is applied at pin 1 and, when the input pulse count reaches decimal 9, outputs  $Q_D$  and  $Q_A$  go high, and as they are connected to reset inputs, pins 7 and 6, the counter is reset. Output is taken from  $Q_A$ , pin 12.

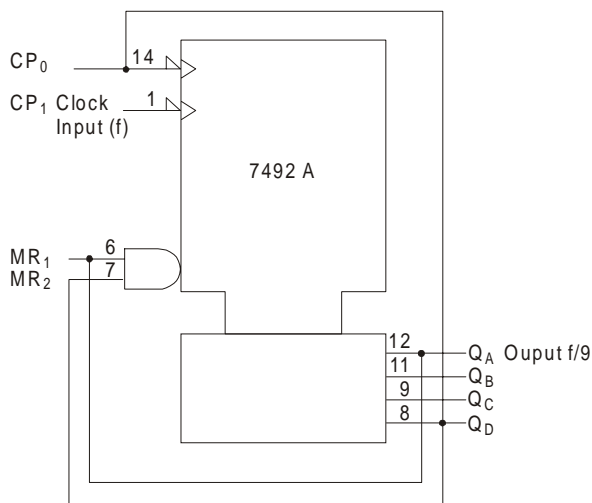
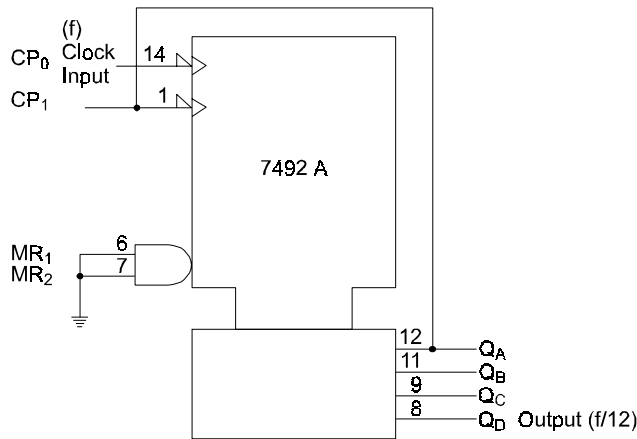


Fig. 7.69 Divide-by-9 circuit



**Divide-by-12-Circuit**

A circuit based on IC 7492 A which will divide the input frequency by 12 is given in Fig. 7.70. Clock input is applied at pin 1 and the output is taken from  $Q_D$ , pin 12. When the circuit reaches pulse count 12, it is automatically reset and it repeats the cycle all over again.



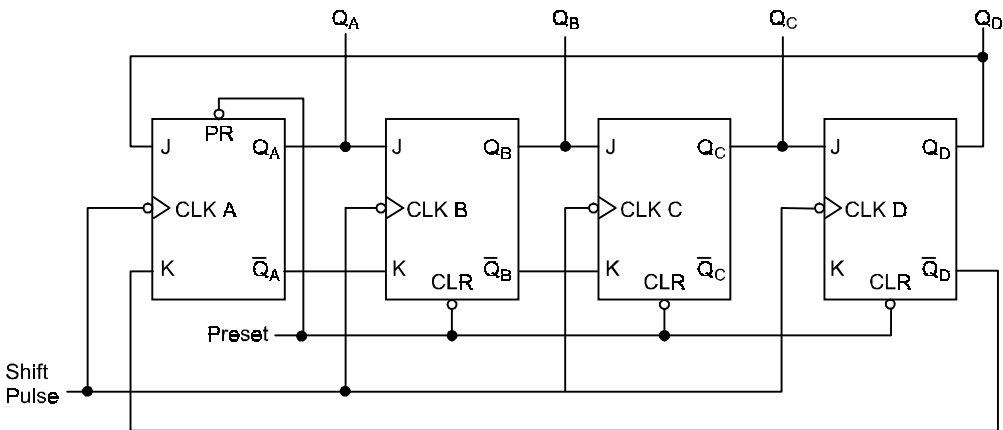
**Fig. 7.70** Divide-by-12 circuit

**7.4.16 Ring Counter**

Ring counters provide a sequence of equally spaced timing pulses and, therefore, find considerable application in logic circuits which require such pulses for setting in motion a series of operations in a predetermined sequence at precise time intervals. Ring counters are a variation of shift registers.

The ring counter is the simplest form of shift register counter. In such a counter the flip-flops are coupled as in a shift register and the last flip-flop is coupled back to the first, which gives the array of flip-flops the shape of a ring as shown in Fig. 7.71. In particular two features of this circuit should be noted.

- (1) The  $Q_D$  and  $\bar{Q}_D$  outputs of the D flip-flop are connected respectively, to the J and K inputs of flip-flop A.
- (2) The preset input of flip-flop A is connected to the reset inputs of flip-flops B, C and D.



**Fig. 7.71** Ring Counter

If we place only one of the flip-flops in the set state and the others in the reset state and then apply clock pulses, the logic 1 will advance by one flip-flop around the ring for each clock pulse and the logic 1 will return to the original flip-flop after exactly four clock pulses, as there are only four flip-flops in the ring. The ring counter does not require any decoder, as we can determine the pulse count by noting the position of the flip-flop, which is set. The total cycle length of the ring is equal to the number of flip-flop stages in the counter. The ring counter has the advantage that it is extremely fast and requires no gates for decoding the count. However it is uneconomical in the number of flip-flops. Whereas a mod-8 counter will require four flip-flops, a mod-8 ring counter will require eight flip-flops.

The ring counter is ideally suited for applications where each count has to be recognized to perform some logical operation.

We can now consider how the modified shift register shown in Fig. 4.78 operates. When the preset is taken low momentarily, flip-flop A sets and all other flip-flops are reset. The counter output will now be as follows:

$$\begin{array}{cccc} Q_A & Q_B & Q_C & Q_D \\ 1 & 0 & 0 & 0 \end{array}$$

At the negative clock edge of the 1st pulse, flip-flop A resets  $Q_A$  becomes 0,  $Q_B$  becomes 1 and  $Q_C$  and  $Q_D$  remain 0. The counter output is now as follows:

$$\begin{array}{cccc} Q_A & Q_B & Q_C & Q_D \\ 0 & 1 & 0 & 0 \end{array}$$

After the 4th clock pulse, the counter output will be as follows:

$$\begin{array}{cccc} Q_A & Q_B & Q_C & Q_D \\ 1 & 0 & 0 & 0 \end{array}$$

You will notice that this was the position at the beginning of the operation, when the preset input was activated. A single logic 1 has travelled round the counter shifting one flip-flop position at a time and has returned to flip-flop A. The states of the flip-flops have been summarized in Table 7.19.

**Table 7.19 Ring counter states**

States	Counter output			
	$Q_A$	$Q_B$	$Q_C$	$Q_D$
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	1	0	0	0

The relevant waveforms are shown in Fig. 7.72.

If preset and clear inputs are not available, it is necessary to provide the required gating, so that the counter starts from the initial state. This can be simply arranged by using a NOR gate as shown in Fig. 7.73.

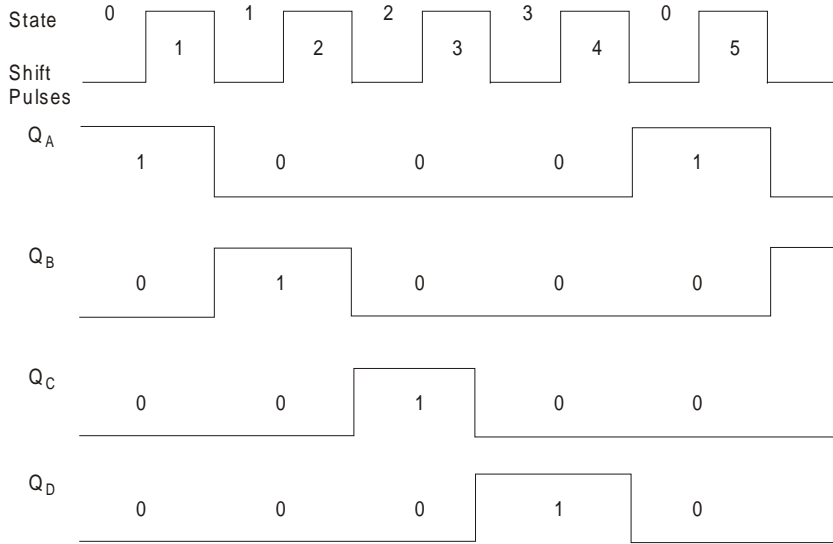


Fig. 7.72

The NOR gate ensures that the input to flip-flop A will be 0 if any of the outputs of A, B, C flip-flops is a logic 1. Now, on the application of clock pulses 0s will be moved right into the counter until all A, B and C flip-flops are reset. When this happens, a logic 1 will be shifted into the counter, and as this 1 is shifted right through the A, B and C flip-flops it will be preceded by three more 0s, which will again be followed by a logic 1 from the NOR gate when flip-flops, A, B and C are all reset.

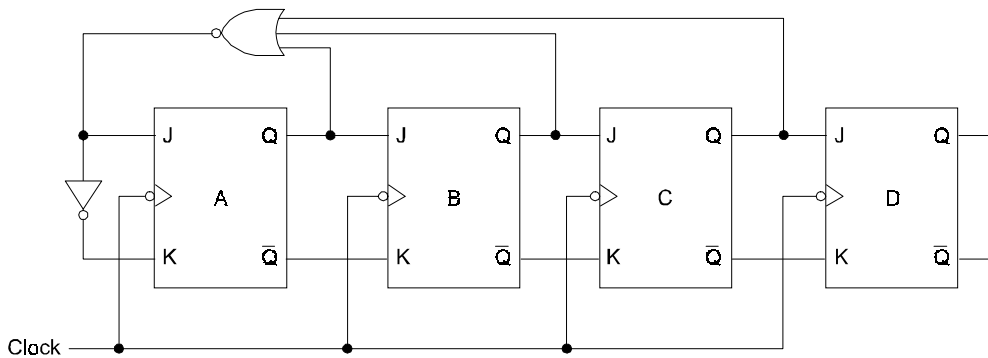


Fig. 7.73 Ring counter with correcting circuit

### 7.4.17 Johnson Counter

The ring counter can be modified to effect an economy in the number of flip-flops used to implement a ring counter. In modified form it is known as a switchtail ring counter or Johnson counter. The modified ring counter can be implemented with only half the number of flip-flops.

In the ring counter circuit shown in Fig. 7.71, the Q<sub>D</sub> and  $\bar{Q}_D$  outputs of the D-flip-flop were connected respectively, to the J and K inputs of flip-flop A. In the Johnson counter, the

outputs of the last flip-flop are crossed over and then connected to the J and K inputs of the first flip-flop. Fig. 7.74 shows a Johnson counter using four JK flip-flops in the shift register configuration, shown  $Q_D$  and  $\bar{Q}_D$  outputs connected respectively, to the K and J inputs of flip-flop A. Because of this cross-connection, the Johnson counter is sometimes referred to as a twisted ring counter.

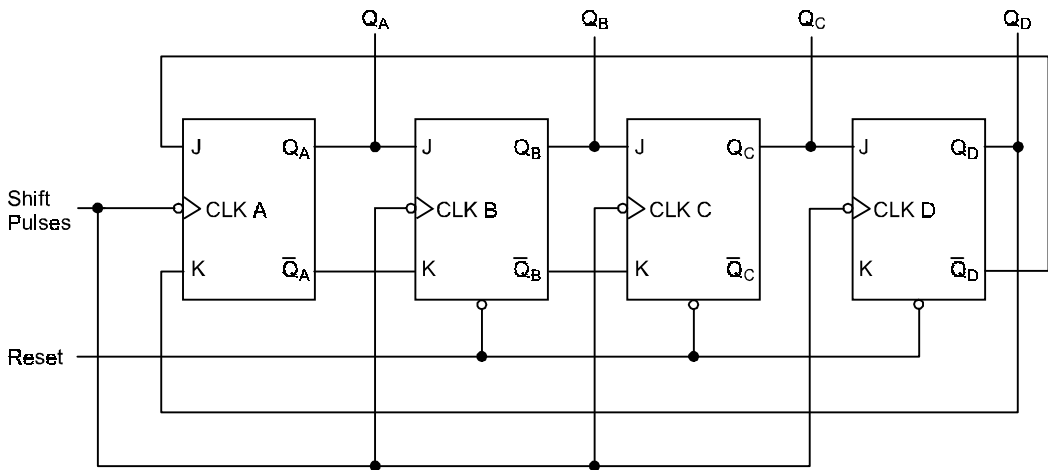


Fig. 7.74 Four-stage Johnson counter

To enable the counter to function according to the desired sequence, it is necessary to reset all the flip-flops. Initially therefore,  $Q_D$  is 0 and  $Q_A$  is 1, which makes the J input of flip-flop A logic 1. We will now study how shift pulses alter the counter output.

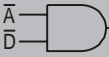
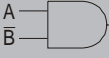

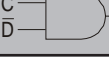




- (1) Since the J input of flip-flop A is 1, the 1st shift pulse sets the A flip-flop and the other flip-flops remain reset as the J inputs of these flip-flops are 0 and K inputs are 1.
- (2) When the 2nd shift pulse is applied, since  $Q_D$  is still 1, flip-flop A remains set and flip-flop B is set, while flip-flop C and D remain reset.
- (3) During the 3rd shift pulse, flip-flop C also sets, while flip-flops A and B are already set; but flip-flop D remains reset.
- (4) During the 4th, pulse, flip-flop D also sets while flip-flops A, B and C are already set.
- (5) During the 5th pulse as  $\bar{Q}_D$  is 0, flip-flop A resets, while flip-flops B, C and D remain set.

The entire sequence of states, which are 8 in all, is as shown in Table 7.20.

You will notice from Table 7.20 that Johnson counter with four flip-flops has eight valid states. Since four flip-flops have been used, the total number of states is 16, out of which 8 are invalid, which have been listed in Table 7.21.

The valid states require decoding, which is different from normal decoding used for standard pure binary count sequence. You will notice that state 1 is uniquely defined, when the outputs of flip-flops A and D are low. Thus a 2-input AND gate with inputs as shown in the table can decode state 1. State 2 is also fully defined by A high and B low. Similarly, the other outputs can be decoded by the gates with inputs as shown in Table 7.20.

**Table 7.20**

State	$Q_D$	$Q_C$	$Q_B$	$Q_A$	Binary equivalent	Output decoding
1	0	0	0	0	0	
2	0	0	0	1	1	
3	0	0	1	1	3	
4	0	1	1	1	7	
5	1	1	1	1	15	
6	1	1	1	0	14	
7	1	1	0	0	12	
8	1	0	0	0	8	

**Table 7.21 Invalid States**

$Q_D$	$Q_C$	$Q_B$	$Q_A$	Binary equivalent
0	1	0	0	4
1	0	0	1	9
0	0	1	0	2
0	1	0	1	5
1	0	1	1	11
0	1	1	0	6
1	1	0	1	13
1	0	1	0	10

In order to ensure that the counter counts in the prescribed sequence given in Table 7.20, an initial reset pulse may be applied, which will reset all the flip-flops. If this is not done, there is no surety that the counter will revert to the valid counting sequence. If the counter should find itself in an unused state, it may continue to advance from one disallowed state to another. The solution to the problem lies in applying extra feedback, so that the counter reverts to the correct counting sequence. For this purpose, the self-correcting circuit given in Fig. 7.76 may be used. The input to the AND gate is  $Q_A \overline{Q_B} \overline{Q_C} Q_D$  and thus it decodes the word 1 0 0 1, which overrides the input, which is 0 and the counter produces an output of 1 1 0 0, which is a part of the allowed counting sequence. From then onwards the counter functions in the desired sequence.

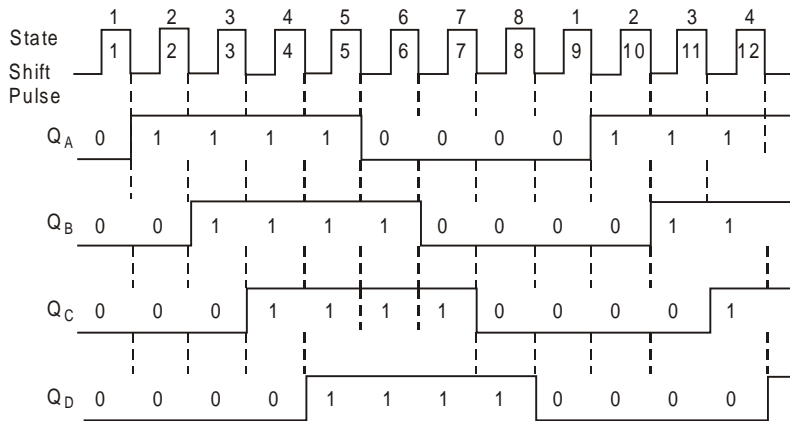


Fig. 7.75 Waveforms for a 4-stage Johnson counter

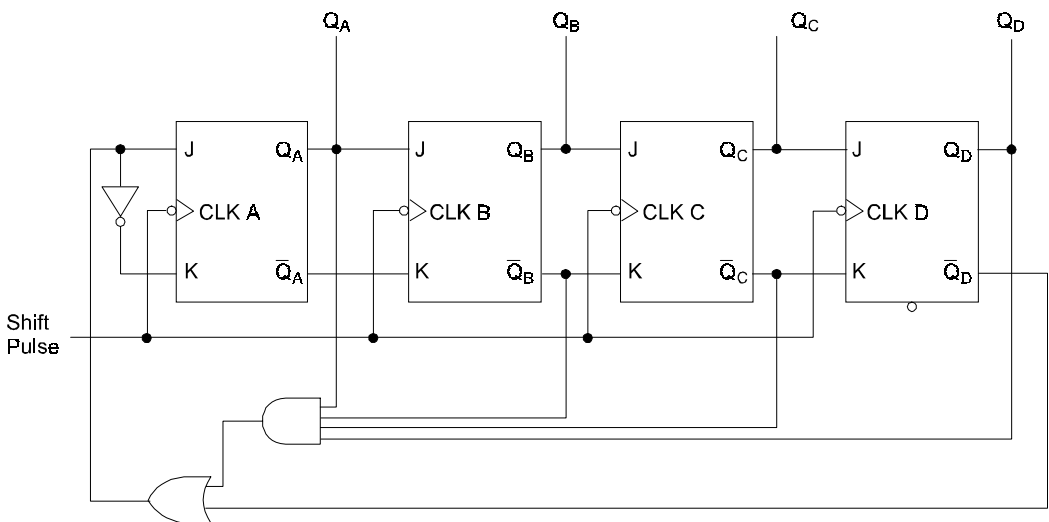


Fig. 7.76 Self-starting and self-correcting Johnson counter

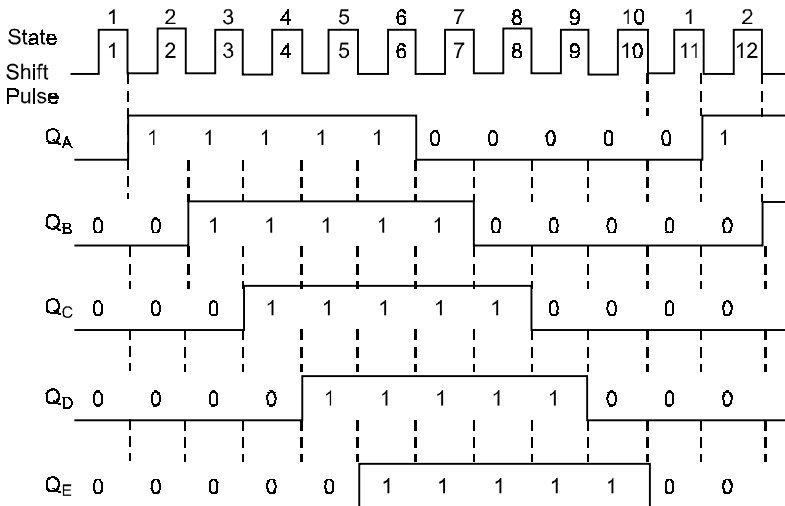
7.4.17.1 Five-stage Johnson Counter

While discussing the 4-stage Johnson counter, you must have observed that this counter divides the clock frequency by 8. Therefore, a Johnson counter with  $n$  flip-flops will divide the clock frequency by  $2n$  or, in other words, there will be  $2n$  discrete states. If we have five flip-flops connected as a Johnson counter, we will have 10 discrete states. Consequently, we will have a decade counter. However, it should be noted that this counter will have in all 32 states, out of which the desired count sequence will utilize only 10 states and the remaining 22 will have to be disallowed. As in the case of a four flip-flop Johnson counter, some form of feedback will have to be incorporated, to disallow the illegal states. A self-correcting circuit like the one shown in Fig. 7.76 may be used with this counter Table 7.22 shows the sequence of the ten allowed states for this counter. The waveforms are shown in Fig. 7.77.

**Table 7.22**

State	<i>E</i>	<i>D</i>	<i>C</i>	<i>B</i>	<i>A</i>	Output decoding
1	0	0	0	0	0	$\overline{A}\overline{E}$
2	0	0	0	0	1	$A\overline{B}$
3	0	0	0	1	1	$B\overline{C}$
4	0	0	1	1	1	$C\overline{D}$
5	0	1	1	1	1	$D\overline{E}$
6	1	1	1	1	1	<i>A E</i>
7	1	1	1	1	0	$\overline{A}B$
8	1	1	1	0	0	$\overline{B}C$
9	1	1	0	0	0	$\overline{C}D$
10	1	0	0	0	0	$\overline{D}E$

For decoding the output of the 5-stage Johnson counter use 2-input AND gates. The inputs to these gates have been indicated in Table 7.22.



**Fig. 7.77** Waveform for a 5-stage Johnson counter

### 7.4.18 Ring Counter Applications

Ring counters find many applications as

- (1) Frequency dividers
- (2) Counters
- (3) Code generators and
- (4) Period and sequence generators

### Frequency dividers

If you look at the waveform Fig. 7.72 of the 4-stage ring counter shown in Fig. 7.71, you will notice that the B flip-flop produces one output pulse for two input pulses, that is it divides the frequency of the shift pulse by 2. Similarly, flip-flop C produces one output pulse for every three input pulses, that is it divides the input frequency by 3, and flip-flop D divides the input frequency by 4. If there are  $n$  flip-flops they will divide the shift pulse by  $n$ . Thus a shift register connected as a ring counter can be used as a frequency divider.

### Counters

A shift register, when connected as a ring counter, can also be used as a counter. For instance, the flip-flop outputs of the ring counter in Fig. 7.71 also give an indication of the number of pulses applied and, therefore counting requires no decoding.

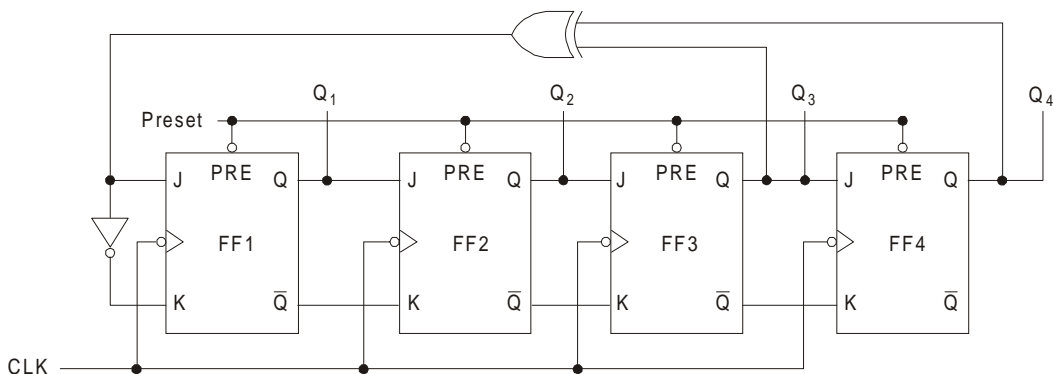
### Sequence generators

Sequence generators are circuits which generate a prescribed sequence of bits in synchronism with a clock. By connecting the outputs of flip-flops in a ring counter to the logic circuits whose operations are to be controlled according to a certain sequence, a ring counter can perform a very useful function. Since ring counters are activated by fixed frequency clocks, the timing intervals between the logic circuits to be controlled can be very precise.

This is of particular importance in computers where instructions have to be executed at the right time and in the correct sequence.

#### 7.4.18.1 Feedback Counters

The ring counters which we have considered so far have a cycle length which is the same as the number of flip-flops in the counter. For instance, the ring counter in Fig. 7.71 has a cycle length of 4. It is possible to design a ring counter which produces a longer cycle length of  $2^n - 1$ , where  $n$  is the number of flip-flops in the ring counter. The trick lies in decoding the outputs of the shift register and feeding the decoded output back to the input. This technique can be used to develop a wide variety of count sequences and output waveforms. To achieve a cycle length of  $2^n - 1$ , an exclusive-OR gate may be used as the feedback element, which provides a feedback term from an even number of stages to the first stage. Table 7.23 intended for counters up to 12 stages, shows the stages the outputs of which are to be fed back to the first flip-flop in the chain.



**Fig. 7.78** Four-stage feedback counter



This table can be used for designing counters of the type shown in Fig. 7.78, when the feedback element consists of a single XOR gate. The count sequence for this 4-stage counter is given in Table 7.24. When you refer to Table 7.23, you will notice that the feedback term for a 4-stage counter using an XOR gate as the feedback element is  $F = (Q_3 \oplus Q_4)$ . The truth table for an XOR gate reproduced below will enable you to determine the input to the first stage in the counter.

<i>Input</i>		<i>Output</i>
<i>A</i>	<i>B</i>	<i>F</i>
0	0	0
0	1	1
1	0	1
1	1	0

**Table 7.23 Feedback terms for counter design**

<i>No. of stage</i>	<i>Feedback stage</i>			
2		$Q_1$	$Q_2$	
3		$Q_2$	$Q_3$	
4		$Q_3$	$Q_4$	
5		$Q_3$	$Q_5$	
6		$Q_5$	$Q_6$	
7		$Q_6$	$Q_7$	
8	$Q_4$	$Q_5$	$Q_6$	$Q_8$
9		$Q_5$	$Q_9$	
10		$Q_7$	$Q_{10}$	
11		$Q_9$	$Q_{11}$	
12	$Q_6$	$Q_8$	$Q_{11}$	$Q_{12}$

In determining the counter states, all that is necessary is to determine the feedback input to the first flip-flop and, since JK flip-flops have been used, the input to the first flip-flop will be the same as the output of the XOR gate, which depends on the outputs of FF3 and FF4. Table 7.24 has been prepared on this basis.

It is important to note that the 0 state of count sequence has to be excluded by additional gating or by using the preset input. If you refer to the first row of the table, you will observe that both outputs  $Q_3$  to  $Q_4$  are 1 and therefore  $F = 0$ . Consequently, the input to the first flip-flop is also 0, which will make its output on the first clock pulse 0. The outputs of FF2 and FF3 will remain unchanged on the first clock pulse. You can determine the outputs in the remaining rows on this basis.

A close look at the table will show you that the output of FF2 resembles the output of FF1, but it is delayed by one clock pulse from that of FF1. Similarly, the outputs of FF3 and FF4 are also delayed by one clock pulse as compared to the outputs of the immediately preceding flip-flops.

**Table 7.24** Count sequence for 4-stage feedback counter

Clock input	Output			
	$Q_1$	$Q_2$	$Q_3$	$Q_4$
0	1	1	1	1
1	0	1	1	1
2	0	0	1	1
3	0	0	0	1
4	1	0	0	0
5	0	1	0	0
6	0	0	1	0
7	1	0	0	1
8	1	1	0	0
9	0	1	1	0
10	1	0	1	1
11	0	1	0	1
12	1	0	1	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

This procedure can be used for designing counters which are required to cycle through a large number of states. For instance a counter which uses 8 flip-flops will cycle through  $2^8 - 1$  or 255 states. We have used only a single XOR gate as the feedback element, but the feedback logic can be designed differently to sequence through any desired sequence or waveform.

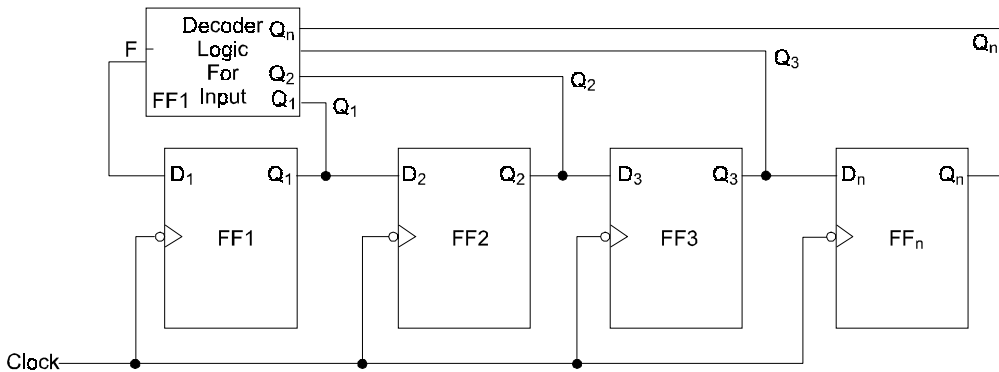
#### 7.4.18.2 Sequence generators

Here we are concerned with pseudo-random sequence generators. They will be random in the sense that the output generated will not cycle through the normal binary count. The sequence is termed pseudo, as it is not random in the real sense, because it will sequence through all the possible states once every  $2^n - 1$  clock cycles. The random sequence generator given in Fig. 7.79 has  $n$  stages and it will therefore sequence through  $2^n - 1$  values before it repeats the same sequence of values.

Let us consider the sequence 100110011001. The bit sequence in this number has a length of 4 that is 1001, if you read it from the first bit on the left. You can also read the sequence from the 2nd and 3rd bits on the left, when the bit patterns will appear to be 0011 and 0110. No matter how you read it, the bit length does not change, nor does the sequence of bits change. You can describe the pattern of bits as 1001, 0011 or 0110.

We can now consider the structure of a sequence generator given in a simple form in Fig. 7.79 using D-type flip-flops connected as in a shift register. The output of the flip-flops are connected through a feedback decoder to the input of the first flip-flop. The output of the decoder is a function of the flip-flop outputs connected to it and the decoder circuitry. We can state this as follows :

$$F = f(Q_1, Q_2, Q_3, Q_n)$$



**Fig. 7.79** Basic structure of a sequence generator

The desired sequence of bits will appear at the output of each of the flip-flops, but the output of each of the successive flip-flops will show a delay in the appearance of the sequence by one clock interval over the one which precedes it.

The minimum number of flip-flops required to generate a sequence of length  $S$  is given by

$$S = 2^n - 1$$

Where  $n$  is the number of flip-flops in the chain.

However, if the minimum number of flip-flops is used, it is not possible to say off hand, that it will be possible to generate a sequence of the required length; but for a given number of flip-flops there is invariably one sequence which has the maximum length.

It is important that in the generation of a sequence no state should be repeated, as that will put a limit on the number of states, because every state determines the development of the future sequence. Besides, the all 0 state has to be excluded, as in this case the input to the first flip-flop in the chain will be 0, which implies that the next state will also be 0, in which case the sequence generator would stop functioning.

We will now consider the steps in the generation of the sequence 1001001 of seven bits. The number of stages that will be required to generate this sequence can be determined as follows:

$$S = 2^n - 1$$

Since  $S = 7$ ;  $n$  should be 3, that is three flip-flops will be required.

However, there is no guarantee that a 7-bit sequence can be generated in 3 stages. If it is not possible, we can try to implement the sequence by using a 4-stage counter; but in this particular case, as you will see, it will be possible to generate this sequence with three stages. The basic arrangement for generating this sequence is shown in Fig. 7.80, which uses three JK flip-flops. The outputs of FF2 and FF3 constitute the inputs to the logic decoder, which in this case is an XOR gate. The output of the XOR gate, which constitutes the input  $F$  to FF1 can be stated as follows:

$$F = (Q_2 \oplus Q_3)$$

You must have noticed that the outputs of FF2 to FF3 are one CLK pulse behind the outputs of flip-flops immediately preceding them. After the first sequence of 7 states has been completed, the sequence is repeated when the 8th (or 1st) CLK pulse arrives. Also observe

that no output state has been repeated, which shows that it has been possible to implement the sequence with only 3 flip-flops.

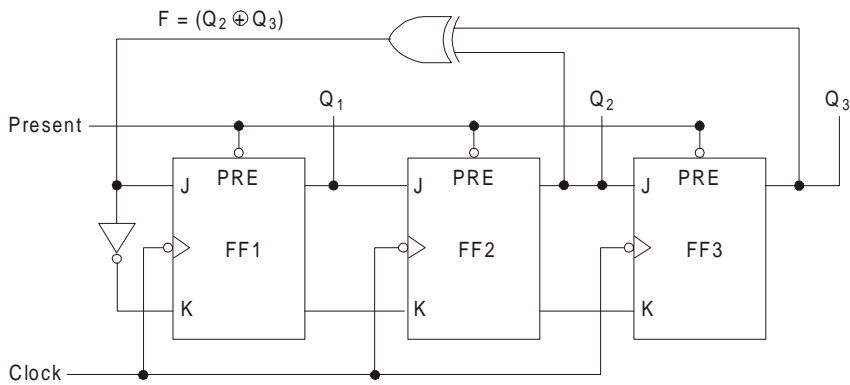


Fig. 7.80 Three-stage sequence generator

When a larger or smaller number of flip-flops is used, the input to the first flip-flop can be worked out on the same basis; but the feedback logic will be different as shown in Table 7.25 for sequence generators using up to 8 stages. For instance for a generator using four flip-flops, F will be as follows:

$$F = (Q_3 \oplus Q_4)$$

Table 7.25 Logic design table for shift register sequences of maximum length ( $S = 2^n - 1$ )

Clock $n$	Feedback state			
2	$Q_1$	$Q_2$		
3	$Q_2$	$Q_3$		
4	$Q_3$	$Q_4$		
5	$Q_3$	$Q_5$		
6	$Q_5$	$Q_6$		
7	$Q_6$	$Q_7$		
8	$Q_2$	$Q_3$	$Q_4$	$Q_8$

The implementation of sequence 1001011 has been presented in Table 7.26.

The count sequence has been developed as follows: You will notice from the table that at the commencement of the operation, the counter is set as shown against CLK 1. Before CLK 2 is applied at FF1 input, the F input to it should be 0, so that its output changes from 1 to 0. Since  $Q_2$  and  $Q_3$  are both 1, the F input to FF1 will be 0. This condition is, therefore, satisfied. The second clock pulse, therefore, changes  $Q_1$  from 1 to 0 and  $Q_2$  and  $Q_3$  remain on 1 as the inputs to these flip-flops are 1. Since both  $Q_2$  and  $Q_3$  are again 1, the F input to FF1, before the arrival of the 3rd clock pulse will again be 0. Therefore, on the arrival of CLK pulse 3, the output of  $Q_1$  will remain 0, as the input to it is 0. On the same CLK pulse  $Q_2$  will change from 1 to 0 as the input to it is 0 and  $Q_3$  will remain on 1 as the input to  $Q_3$  is still 1. Successive changes in the outputs have been worked out on this basis.

**Table 7.26**

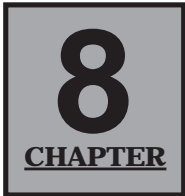
Clock interval <i>CLK</i>	Flip-flop outputs			Input to FF1 $F = (Q_2 \oplus Q_3)$
	$Q_1$	$Q_2$	$Q_3$	
1	1	1	1	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	0
5	0	1	0	1
6	1	0	1	1
7	1	1	0	1
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
1	1	1	1	1

**7.5 EXERCISES**

- Can one store decimal number 12 in an 8-bit shift register.
- The number stored in a 4-bit binary up-counter is 0101. What will be state of the counter after the following clock pulses ?
  - 3rd clock pulse
  - 5th clock pulse
  - 8th clock pulse
  - 12th clock pulse
- In a 4-bit ripple up-counter how many clock pulses will you apply, starting from state 0 0 0 0, so that the counter outputs are as follows ?
  - 0 0 1 0
  - 0 1 1 1
  - 1 0 0 1
  - 1 1 1 0
- Draw the logic diagram for a binary up-counter using four JK flip-flops and draw the truth table and the output waveforms.
- Connect four edge-triggered D-type flip-flops to make an asynchronous up-counter.
- How many JK flip-flops will you require to make the following modulo counters ?
  - Mod-4
  - Mod-6
  - Mod-9
  - Mod-11
- What will be maximum count capability of a counter having 12 JK flip-flops ?

8. How many flip-flops will you require to attain a count capability of 8500 ?
9. An asynchronous counter has four flip-flops and the propagation delay of each flip-flop is 20 ns. Calculate the maximum counting speed of the counter.
10. A synchronous counter has four flip-flops and the propagation delay of each is 20 ns. What is its maximum counting speed ?
11. By how much will a ripple down-counter having three flip-flops divide the input frequency ?
12. Draw a logic diagram, truth table and output waveforms for a ripple down-counter with four flip-flops.
13. What will be the output states of a four flip-flop binary down-counter, after the following input clock pulses, if the initial state of the counter was 1111 ?  
 (a) 4                      (b) 7                      (c) 9                      (d) 14
14. Draw the logic diagram of a presettable down counter with a maximum preset capability of 7.
15. What will be the modulus of IC 74193 in the up-counting mode, if the numbers preset in the counter are as follows ?  
 (a) Decimal 5                      (b) Decimal 7  
 (c) Decimal 9                      (d) Decimal 12
16. What will be the modulus of IC 74193 in the down-counting mode, when the binary numbers preset in the counter are the same as in Problem 15 ?
17. A 74193 up-counter starts counting up from binary number 1 0 0 0. What will be the state of the counter after the 8th clock pulse ?
18. Draw the logic diagram of a Mod-6 counter using the counter reset method. Write its truth table and draw the output waveforms.
19. Show how you will connect two ICs 74193 to build an 8-bit up-down counter.
20. What is the maximum counting capacity of a chain of five BCD counters ?
21. A BCD counter is required to have the following states. After how many clock pulses will these states be reached, if the counter was initially reset ?  
 (a) 0 0 1 0  
 (b) 0 1 0 0  
 (c) 0 1 1 0  
 (d) 1 0 0 1
22. Connect two ICs 74193 to make a modulo-20 divider circuit.
23. Design a mod-10 (Decade) synchronous counter using JK flip-flops.
24. Draw decoding gates for the decade counter in Fig. 4.51.
25. Draw decoding gates for the counter of Fig. 4.49.
26. Redesign the synchronous mod-5 counter circuit discussed in Sec 4.8.12.2 so that whenever the counter reaches the unutilized state 1 0 1, 0 1 1 and 1 1 1 the counter is reset.
27. Design a Mod-7 counter using IC 7490 A.

- 28.** Design a divide-by 120 counter using ICs 7490 A and 7492 A.
- 29.** Design a correcting circuit for a 4-stage ring counter using a NAND gate instead of a NOR gate as used in Fig. 4.80.
- 30.** Determine the maximal length sequence, which can be generated using four JK flip-flops and draw the sequence generated by the first flip-flop in the chain.
- 31.** Draw waveforms to illustrate how a serial binary number 1011 is loaded into a shift register.
- 32.** A binary number is to be divided by 64. By how many positions will you shift the number and in what direction.
- 33.** Describe the working of shift register with PISO/SIPO operation.
- 34.** Design a mod-5 synchronous counter having the states 011, 100, 101, 110, 111 respectively. Obtain a minimal cost design with J-K F/F.
- 35.** Design a shift register counter to generate a sequence length of 8 having self-start feature.



# ASYNCHRONOUS SEQUENTIAL LOGIC

---

## 8.0 INTRODUCTION

Much of today's logic design is based on two major assumptions: all signals are binary, and time is discrete. Both of these assumptions are made in order to simplify logic design. By assuming binary values on signals, simple Boolean logic can be used to describe and manipulate logic constructs. By assuming time is discrete, hazards and feedback can largely be ignored. However, as with many simplifying assumptions, a system that can operate without these assumptions has the potential to generate better results.

Asynchronous circuits keep the assumption that signals are binary, but remove the assumption that time is discrete. This has several possible benefits:

### No Clock Skew

Clock skew is the difference in arrival times of the clock signal at different parts of the circuit. Since asynchronous circuits by definition have no globally distributed clock, there is no need to worry about clock skew. In contrast, synchronous systems often slow down their circuits to accommodate the skew. As feature sizes decrease, clock skew becomes a much greater concern.

### Lower Power

Standard synchronous circuits have to toggle clock lines, and possibly precharge and discharge signals, in portions of a circuit unused in the current computation. For example, even though a floating point unit on a processor might not be used in a given instruction stream, the unit still must be operated by the clock. Although asynchronous circuits often require more transitions on the computation path than synchronous circuits, they generally have transitions only in areas involved in the current computation.

**Note:** that there are techniques being used in synchronous designs to address this issue as well.

### Average-Case Instead of Worst-Case Performance

Synchronous circuits must wait until all possible computations have completed before latching the results, yielding worst-case performance. Many asynchronous systems sense when a computation has completed, allowing them to exhibit average-case performance. For circuits such as ripple-carry adders where the worst-case delay is significantly worse than the average-case delay, this can result in a substantial savings.



## Easing of Global Timing Issues

In systems such as a synchronous microprocessor, the system clock, and thus system performance, is dictated by the slowest (*critical*) path. Thus, most portions of a circuit must be carefully optimized to achieve the highest clock rate, including rarely used portions of the system. Since many asynchronous systems operate at the speed of the circuit path currently in operation, rarely used portions of the circuit can be left unoptimized without adversely affecting system performance.

## Better Technology Migration Potential

Integrated circuits will often be implemented in several different technologies during their lifetime. Early systems may be implemented with gate arrays, while later production runs may migrate to semi-custom or custom ICs. Greater performance for synchronous systems can often only be achieved by migrating all system components to a new technology, since again the overall system performance is based on the longest path. In many asynchronous systems, migration of only the more critical system components can improve system performance on average, since performance is dependent on only the currently active path. Also, since many asynchronous systems sense computation completion, components with different delays may often be substituted into a system without altering other elements or structures.

## Automatic Adaptation to Physical Properties

The delay through a circuit can change with variations in fabrication, temperature, and power-supply voltage. Synchronous circuits must assume that the worst possible combination of factors is present and clock the system accordingly. Many asynchronous circuits sense computation completion, and will run as quickly as the current physical properties allow.

## Robust Mutual Exclusion and External Input Handling

Elements that guarantee correct mutual exclusion of independent signals and synchronization of external signals to a clock are subject to *metastability*. A metastable state is an unstable equilibrium state, such as a pair of cross-coupled CMOS inverters at 2.5V, which a system can remain in for an unbounded amount of time. Synchronous circuits require all elements to exhibit bounded response time. Thus, there is some chance that mutual exclusion circuits will fail in a synchronous system. Most asynchronous systems can wait an arbitrarily long time for such an element to complete, allowing robust mutual exclusion. Also, since there is no clock with which signals must be synchronized, asynchronous circuits more gracefully accommodate inputs from the outside world, which are by nature asynchronous.

With all of the potential advantages of asynchronous circuits, one might wonder why synchronous systems predominate. The reason is that asynchronous circuits have several problems as well. Primarily, asynchronous circuits are more difficult to design in an ad hoc fashion than synchronous circuits. In a synchronous system, a designer can simply define the combinational logic necessary to compute the given functions, and surround it with latches. By setting the clock rate to a long enough period, all worries about hazards (undesired signal transitions) and the dynamic state of the circuit are removed. In contrast, designers of asynchronous systems must pay a great deal of attention to the dynamic state of the circuit. Hazards must also be removed from the circuit, or not introduced in the first place, to avoid incorrect results. The ordering of operations, which was fixed by the placement of latches in a synchronous system, must be carefully ensured by the asynchronous control logic. For complex systems, these issues become too difficult to handle by hand.

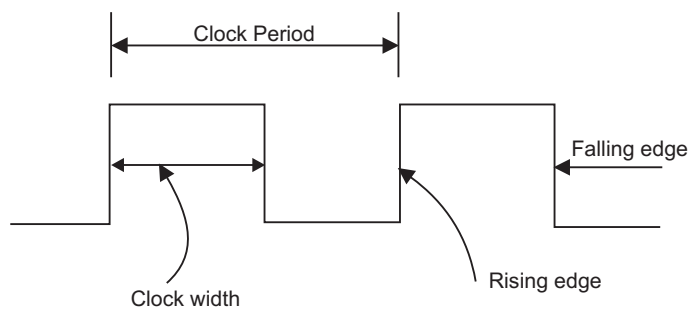
Finally, even though most of the advantages of asynchronous circuits are towards higher performance, it isn't clear that asynchronous circuits are actually any faster in practice. Asynchronous circuits generally require extra time due to their signaling policies, thus increasing average-case delay. Whether this cost is greater or less than the benefits listed previously is unclear, and more research in this area is necessary.

Even with all of the problems listed above, asynchronous design is an important research area. Regardless of how successful synchronous systems are, there will always be a need for asynchronous systems. Asynchronous logic may be used simply for the interfacing of a synchronous system to its environment and other synchronous systems, or possibly for more complete applications.

### 8.1 DIFFERENCE BETWEEN SYNCHRONOUS AND ASYNCHRONOUS

Sequential circuits are divided into two main types: *synchronous* and *asynchronous*. Their classification depends on the timing of their signals.

*Synchronous* sequential circuits change their states and output values at discrete instants of time, which are specified by the rising and falling edge of a free-running *clock signal*. The clock signal is generally some form of square wave as shown in Figure 8.1 below.



**Fig. 8.1** Clock Signal

From the diagram you can see that the *clock period* is the time between successive transitions in the same direction, that is, between two rising or two falling edges. State transitions in synchronous sequential circuits are made to take place at times when the clock is making a transition from 0 to 1 (rising edge) or from 1 to 0 (falling edge). Between successive clock pulses there is no change in the information stored in memory.

The reciprocal of the clock period is referred to as the *clock frequency*. The *clock width* is defined as the time during which the value of the clock signal is equal to 1. The ratio of the clock width and clock period is referred to as the duty cycle. A clock signal is said to be *active high* if the state changes occur at the clock's rising edge or during the clock width. Otherwise, the clock is said to be *active low*. Synchronous sequential circuits are also known as *clocked sequential circuits*.

The memory elements used in synchronous sequential circuits are usually flip-flops. These circuits are binary cells capable of storing one bit of information. A flip-flop circuit has two outputs, one for the normal value and one for the complement value of the bit stored in it. Binary information can enter a flip-flop in a variety of ways, a fact which give rise to the different types of flip-flops.

In *asynchronous* sequential circuits, the transition from one state to another is initiated by the change in the primary inputs; there is no external synchronization. The memory commonly used in asynchronous sequential circuits are time-delayed devices, usually implemented by feedback among logic gates. Thus, asynchronous sequential circuits may be regarded as combinational circuits with feedback. Because of the feedback among logic gates, asynchronous sequential circuits may, at times, become unstable due to transient conditions.

The differences between synchronous and asynchronous sequential circuits are:

- In a clocked sequential circuit a change of state occurs only in response to a synchronizing clock pulse. All the flip-flops are clocked simultaneously by a common clock pulse. In an asynchronous sequential circuit, the state of the circuit can change immediately when an input change occurs. It does not use a clock.
- In clocked sequential circuits input changes are assumed to occur between clock pulses. The circuit must be in the stable state before next clock pulse arrives. In asynchronous sequential circuits input changes should occur only when the circuit is in a stable state.
- In clocked sequential circuits, the speed of operation depends on the maximum allowed clock frequency. Asynchronous sequential circuits do not require clock pulses and they can change state with the input change. Therefore, in general the asynchronous sequential circuits are faster than the synchronous sequential circuits.
- In clocked sequential circuits, the memory elements are clocked flip-flops. In asynchronous sequential circuits, the memory elements are either unclocked flip-flops (latches) or gate circuits with feedback producing the effect of latch operation.

In clocked sequential circuits, any number of inputs can change simultaneously (during the absence of the clock). In asynchronous sequential circuits only one input is allowed to change at a time in the case of the level inputs and only one pulse input is allowed to be present in the case of the pulse inputs. If more than one level inputs change simultaneously or more than one pulse input is present, the circuit makes erroneous state transitions due to different delay paths for each input variable.

## 8.2 MODES OF OPERATION

*Asynchronous* sequential circuits can be classified into two types:

- Fundamental mode asynchronous sequential circuit
- Pulse mode asynchronous sequential circuit

### Fundamental Mode

In fundamental mode, the inputs and outputs are represented by levels rather than pulses. In fundamental mode asynchronous sequential circuit, it is also assumed that the time difference between two successive input changes is larger than the duration of internal changes. Fundamental mode operation assumes that the input signals will be changed only when the circuit is in a stable state and that only one variable can change at a given time.

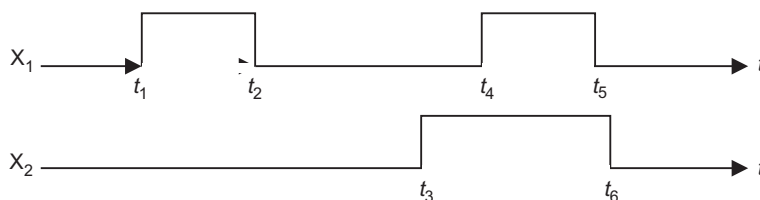
### Pulse Mode

In pulse mode, the inputs and outputs are represented by pulses. In this mode of operation the width of the input pulses is critical to the circuit operation. The input pulse must be long enough for the circuit to respond to the input but it must not be so long as to

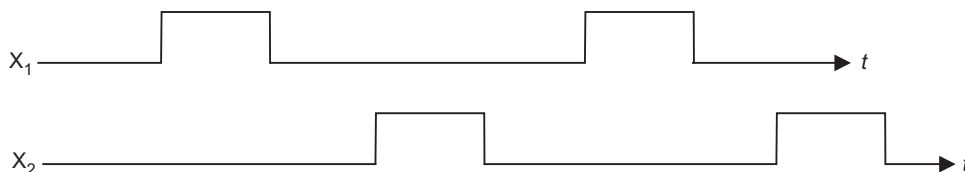
be present even after new state is reached. In such a situation the state of the circuit may make another transition.

The minimum pulse width requirement is based on the propagation delay through the next state logic. The maximum pulse width is determined by the total propagation delay through the next state logic and the memory elements.

In pulse-mode operation, only one input is allowed to have pulse present at any time. This means that when pulse occurs on any one input, while the circuit is in stable state, pulse must not arrive at any other input. Figure 8.2 illustrates unacceptable and acceptable input pulse change.  $X_1$  and  $X_2$  are the two inputs to a pulse mode circuit. In Fig. 8.2 (a) at time  $t_3$  pulse at input  $X_2$  arrives.



**Fig. 8.2 (a)** Unacceptable pulse mode input changes



**Fig. 8.2 (b)** Acceptable pulse mode input changes

While this pulse is still present, another pulse at  $X_1$  input arrives at  $t_4$ . Therefore, this kind of the presence of pulse inputs is not allowed.

Both fundamental and pulse mode asynchronous sequential circuits use unlocked S-R flip-flops or latches. In the design of both types of circuits, it is assumed that a change occurs in only one inputs and no changes occurs in any other inputs until the circuit enters a stable state.

### 8.3 ANALYSIS OF ASYNCHRONOUS SEQUENTIAL MACHINES

Analysis of asynchronous sequential circuits operation in fundamental mode and pulse mode will help in clearly understanding the asynchronous sequential circuits.

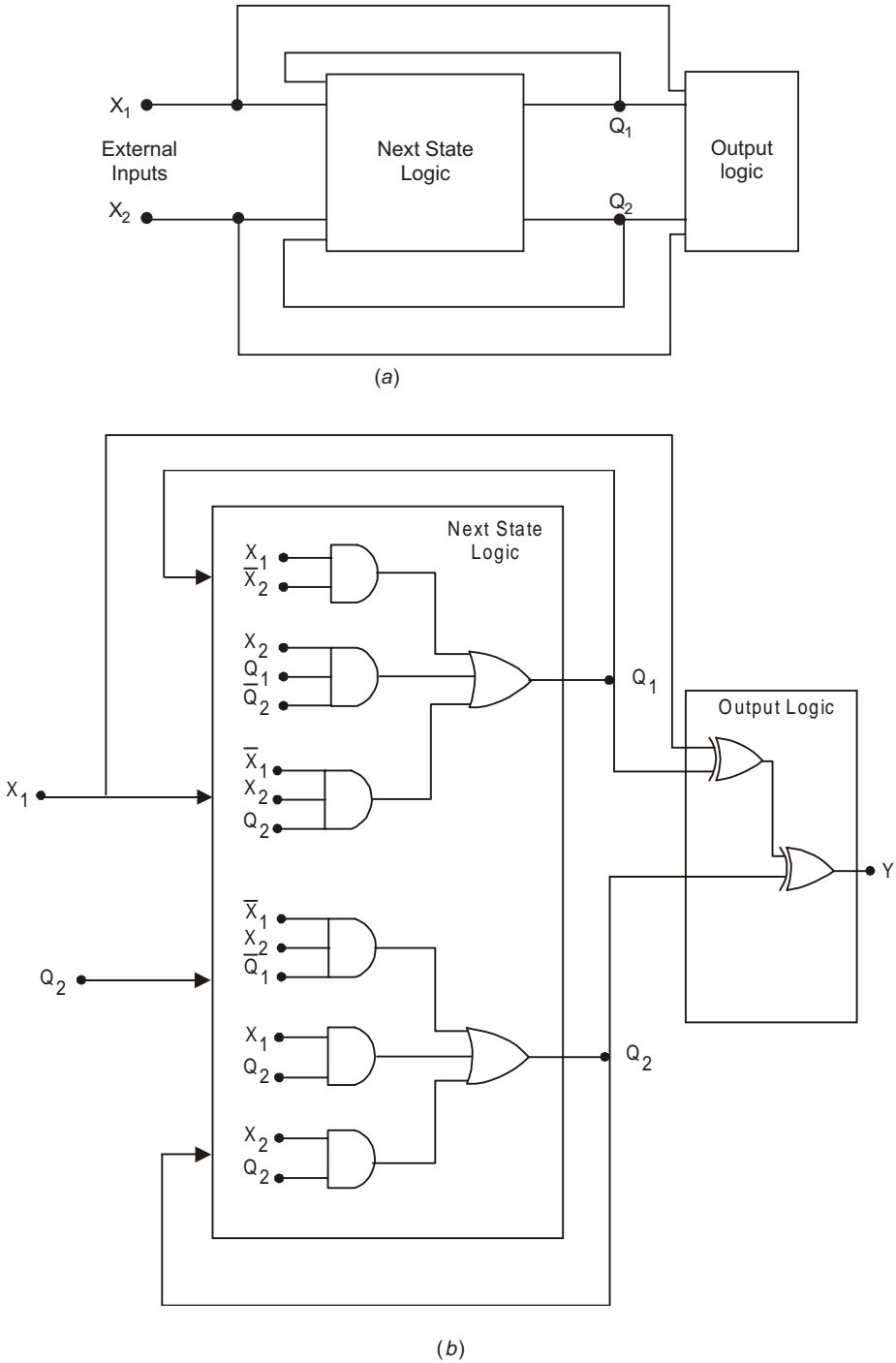
#### 8.3.1 Fundamental Mode Circuits

Fundamental mode circuits are of two types:

- Circuits without latches
- Circuits with latches

#### 8.3.2 Circuits without Latches

Consider a fundamental mode circuit shown in Fig. 8.3.



**Fig. 8.3** Fundamental mode asynchronous sequential circuit without latch  
 (a) block diagram (b) circuit diagram

This circuit has only gates and no explicit memory elements are present. There are two feedback paths from  $Q_1$  and  $Q_2$  to the next-state logic circuit. This feedback creates the latching effect due to delays, necessary to produce a sequential circuit. It may be noted that a memory element latch is created due to feedback in gate circuit.

The first step in the analysis is to identify the *states* and the *state variables*. The combination of level signals from external sources  $X_1, X_2$  is referred to as the input state and  $X_1, X_1$  are the input *state variables*. The combination of the outputs of memory elements are known as *secondary*, or *internal states* and these variables are known as *internal* or *secondary state variables*. Here,  $Q_1$  and  $Q_2$  are the internal variables since no explicit elements are present. The combination of both, input state and the secondary state ( $Q_1, Q_2, X_1, X_2$ ) is known as the *total state*.  $Y$  is the output variable.

The next secondary state and output logic equations are derived from the logic circuit in the next-state logic block. The next-secondary state variables are denoted by  $Q_1^+$  and  $Q_2^+$  these are given by

$$\begin{aligned} Q_1^+ &= X_1 \bar{X}_2 + \bar{X}_1 X_2 Q_2 + X_2 Q_1 \bar{Q}_2 \\ Q_2^+ &= \bar{X}_1 X_2 \bar{Q}_1 + X_1 Q_2 + X_2 Q_2 \\ Y &= X_1 \oplus Q_1 \oplus Q_2 \end{aligned}$$

Here,  $Q_1$  and  $Q_2$  are the present secondary state variables when  $X_1, X_2$  input-state variables occur, the circuit goes to next secondary state. A state table shown in Table 8.1 is constructed using these logic equations. If the resulting next secondary state is same as the present state, i.e.  $Q_1^+ = Q_1$  and  $Q_2^+ = Q_2$ , the total state  $Q_1, Q_2, X_1, X_2$  is said to be stable. Otherwise it is unstable.

The stability of the next total state is also shown in Table 8.1.

**Table 8.1 State Table**

Present total state				Next total state				Stable total state	Output
$Q_1$	$Q_2$	$X_1$	$X_2$	$Q_1^+$	$Q_2^+$	$X_1$	$X_2$	Yes/No	$Y$
0	0	0	0	0	0	0	0	Yes	0
0	0	0	1	0	1	0	1	No	0
0	0	1	1	0	0	1	1	Yes	1
0	0	1	0	1	0	1	0	No	1
0	1	0	0	0	0	0	0	No	1
0	1	0	1	1	1	0	1	No	1
0	1	1	1	0	1	1	1	Yes	0
0	1	1	0	1	1	1	0	No	0
1	1	0	0	0	0	0	0	No	0
1	1	0	1	1	1	0	1	Yes	0
1	1	1	1	0	1	1	1	No	1
1	1	1	0	1	1	1	0	Yes	1
1	0	0	0	0	0	0	0	No	1
1	0	0	1	1	0	0	1	Yes	1
1	0	1	1	1	0	1	1	Yes	0
1	0	1	0	1	0	1	0	Yes	0

### 8.3.3 Transition Table

A state table can be represented in another form known as *transition table*. The transition table for the state table of Table 8.1 is shown in Fig. 8.4.

In a transition table, columns represent input states (one column for each input state) and rows represent secondary states (one row for each secondary state). The next secondary state values are written into the squares, each indicating a total state. The stable states are circled. For any given present secondary state ( $Q_1 Q_2$ ), the next secondary state is located in the square corresponding to row for the present secondary state and the column for the input state ( $X_1 X_2$ ).

For example, for  $Q_1 Q_2 = 11$  and  $X_1 X_2 = 00$ , the next secondary state is 00 (third row, first column) which is an unstable state.

Present internal state $Q_1 Q_2$	Input State $X_1 X_2$				Next State $Q_1^+ Q_2^+$	
	00	01	11	10		
00	00	01	00	10		
01	00	11	01	11		
11	00	11	01	11		
10	00	10	10	10		

**Fig. 8.4** Transition table for Table 8.1.

For a given input sequence, the total state sequence can be determined from the transition table.

**Example.** For the transition table shown in Fig 8.4, the initial total state is  $Q_1 Q_2 X_1 X_2 = 0000$ . Find the total state sequence for an input sequence  $X_1 X_2 = 00, 01, 11, 10, 00$ .

**Solution.** For a given internal state of the circuit, a change in the value of the circuit input causes a horizontal move in the transition table to the column corresponding to the new input value. A change in the internal state of the circuit is reflected by a vertical move. Since a change in the input can occur only when the circuit is in a stable state, a horizontal move can start only from a circled entry.

The initial total state is 0000 (first row, first column) which is a stable state. When the input state changes from 00 to 01, the circuit makes a transition (horizontal move) from the present total state to the next total state 0101 (first row, second column) which is unstable. Next, the circuit makes another transition from 0101 to 1101 (vertical move) (second row, second column) which is also an unstable state. Finally in the next transition (vertical move) it comes to stable state 1101 (third row, second column). All these transitions are indicated by arrows. Thus we see that a single input change produces two secondary state changes

before a stable total state is reached. If the input is next changed to 11 the circuit goes to total state 0111 (horizontal move) which is unstable and then to stable total state 0111 (vertical move). Similarly, the next input change to 10 will take the circuit to unstable total state 1110 (horizontal move) and finally to stable total state 1110 (vertical move). A change in input state from 10 to 00 causes a transition to unstable total state 0000 (horizontal move) and then to stable total state 0000 (vertical move), completing the state transitions for the input sequence. All the state transitions are indicated by arrows.

The total state sequence is



From the preceding discussions we see that from the logic diagram of an asynchronous sequential circuit, logic equations, state table, and transition table can be determined. Similarly, from the transition table, logic equations can be written and the logic circuit can be designed.

### 8.3.4 Flow table

In asynchronous sequential circuits design, it is more convenient to use *flow table* rather than transition table. A flow table is basically similar to a transition table except that the internal states are represented symbolically rather than by binary states. The column headings are the input combinations and the entries are the next states, and outputs. The state changes occur with change of inputs (one input change at a time) and logic propagation delay.

The flow of states from one to another is clearly understood from the flow table. The transition table of Fig. 8.4 constructed as a flow table is shown in Fig. 8.5. Here, *a*, *b*, *c*, and *d* are the states. The binary value of the output variable is indicated inside the square next to the state symbol and is separated by a comma. A stable state is circled.

Present internal state $Q_1 Q_2$		Input State $X_1 X_2$			
		00	01	11	10
Unstable State	<i>a</i>	Ⓐ, 0	<i>b</i> , 0	Ⓐ, 1	<i>d</i> , 1
	<i>b</i>	<i>a</i> , 1	<i>c</i> , 1	Ⓑ, 0	<i>c</i> , 0
	<i>c</i>	<i>a</i> , 0	Ⓒ, 1	<i>b</i> , 1	Ⓒ, 1
	<i>d</i>	<i>a</i> , 1	Ⓓ, 1	Ⓓ, 0	Ⓓ, 0

Fig. 8.5 Flow table

From the flow table, we observe the following behavior of the circuit.

When  $X_1 X_2 = 00$ , the circuit is in state Ⓐ. It is a stable state. If  $X_2$  changes to 1 while  $X_1 = 0$ , the circuit goes to state *b* (horizontal move) which is an unstable state. Since *b* is an unstable state, the circuit goes to *c* (vertical move), which is again an unstable state. This



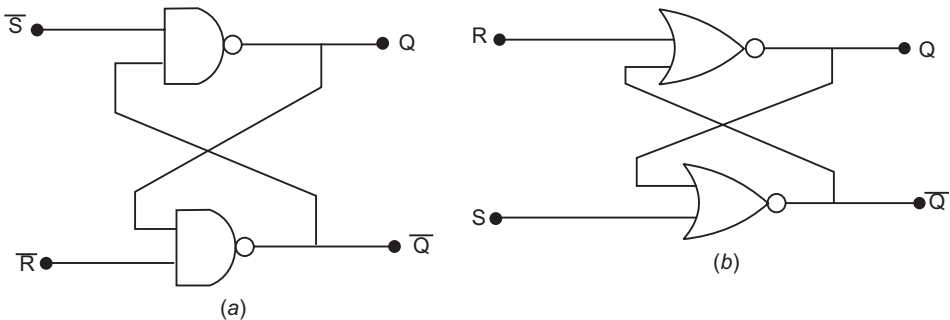
causes another vertical move and finally the circuit reaches a stable state ③. Now consider  $X_1$  changing to 1 while  $X_2 = 1$ , there is a horizontal movement to the next column. Here  $b$  is an unstable state and therefore, there is a vertical move and the circuit comes to a stable state ④. Next change in  $X_2$  from 1 to 0 while  $X_1$  remaining 1 will cause horizontal move to state  $c$  (unstable state) and finally to stable state ③ due to the vertical move. Similarly changing  $X_1$  from 1 to 0 while  $X_2 = 0$  will cause the circuit to go to the unstable state  $a$  and finally to stable state ①. The flow of circuit states are shown by arrows.

In the flow table of Fig. 8.5 there are more than one stable states in rows. For example, the first row contains stable states in two columns. If every row in a flow table has only one stable state, the flow table is known as a *primitive flow table*.

From a flow table, transition table can be constructed by assigning binary values to each state and from the transition table logic circuit can be designed by constructing K-maps for  $Q_1^+$  and  $Q_2^+$ .

### 8.3.5 Circuits with Latches

In Chapter 6 latches were introduced. Latch circuits using NAND and NOR gates are shown in Fig. 8.6.



**Fig. 8.6** (a)  $\bar{S}$  -  $\bar{R}$  latch using NAND gates. (b) S-R latch using NOR gates.

For the circuit of Fig 8.6a, the next-state equation is

$$Q^+ = \bar{\bar{S}} \cdot \bar{\bar{Q}} = \bar{\bar{S}} \cdot \overline{\overline{QR}}$$

$$= S + \bar{R}Q$$

Similarly, for the circuit of Fig. 8.6 b, the next-state equation is

$$Q^+ = \overline{\overline{R + \bar{Q}}} = \overline{\overline{R + (S + Q)}}$$

$$= \bar{R} \cdot (S + Q)$$

$$= \bar{S}\bar{R} + \bar{R}Q$$

Since,  $S = R = 1$  is not allowed, which means  $SR = 0$ , therefore,

$$\bar{S}\bar{R} = \bar{S}\bar{R} + SR = S(\bar{R} + R) = S$$

which gives,

$$Q^+ = S + \bar{R}Q$$

It is same as the next-state equation for the circuit of Fig 8.6a

The transition table of S-R latch is shown in Fig. 8.7.

SR		Q <sup>+</sup>				
		00	01	11	10	
Q	0	⊙	⊙	⊙	1	$Q^+ = S\bar{R} + \bar{R}Q$ $= S + \bar{R}Q$
	1	⊙	0	0	⊙	

Fig. 8.7 Transition table of S-R latch

From the transition table of S-R FLIP-FLOP, we observe that when SR changes from 11 to 00 the circuit will attain either the stable state ⊙ (first row, first column) or ⊙ (second row, first column) depending upon whether S goes to 0 first or R goes to 0 first respectively. Therefore, S = R = 1 must not be applied.

Consider an asynchronous sequential circuit with latches shown in Fig. 8.8

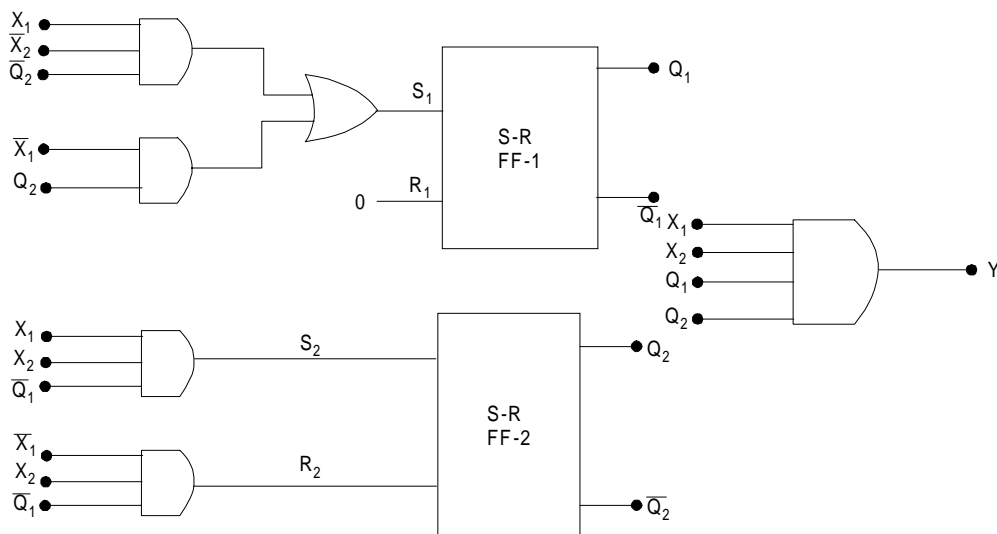


Fig. 8.8 Asynchronous sequential circuit with latches

For FF-1, R<sub>1</sub> = 0 and the excitation equation for S<sub>1</sub> is

$$S_1 = X_1 \bar{X}_2 \bar{Q}_2 + \bar{X}_1 Q_2$$

The next-state equation is

$$Q_1^+ = S_1 + \bar{R}_1 Q_1$$

Substituting the value of S<sub>1</sub> we obtain,

$$Q_1^+ = X_1 \bar{X}_2 \bar{Q}_2 + \bar{X}_1 Q_2 + Q_1$$

Similarly, the excitation equations for FF-2 are

$$S_2 = X_1 X_2 \bar{Q}_1, R_2 = \bar{X}_1 X_2 \bar{Q}_1$$

The next-state equation is

$$Q_2^+ = S_2 + \bar{R}_2 Q_2$$

$$= X_1 X_2 \bar{Q}_1 + \bar{X}_1 X_2 \bar{Q}_1 \cdot Q_2$$

Using next-state equation for FF-1 and FF-2, transition table is obtained as shown in Fig. 8.9.

$X_1 X_2$		$Q_1^+ Q_2^+$			
		00	01	11	10
$Q_1 Q_2$	00	00	00	01	10
	01	11	11	01	01
	11	11	10	11	11
	10	10	10	10	10

Fig. 8.9 Transition table for the circuit of Fig. 8.8

The output function is

$$Y = X_1 X_2 Q_1 Q_2$$

Its flow table is shown in Fig. 8.10.

$X_1 X_2$		$Q_1^+ Q_2^+$			
		00	01	11	10
$Q_1 Q_2$	a	ⓐ, 0	ⓐ, 0	b, 0	d, 0
	b	c, 0	c, 0	ⓑ, 0	ⓑ, 0
	c	ⓒ, 0	d, 0	ⓒ, 1	ⓒ, 0
	d	ⓓ, 0	ⓓ, 0	ⓓ, 0	ⓓ, 0

Fig. 8.10 Flow table for the circuit of Fig. 8.8

From a flow table, transition table can be obtained by assigning binary values to the states. From the transition table, logic equations can be obtained by constructing K-maps for S and R inputs of every latch. For this, the excitation table of S-R latch will be used. Logic circuit can then be designed using the logic equation for S, R inputs of every latch.

**Example.** Design logic circuit using S-R latches for the transition table of Fig. 8.4.

**Solution.** Since, there are two internal states  $Q_1$  and  $Q_2$ , therefore, two S-R latches are required for the design of logic circuit. Let the two latches be  $L_1$  and  $L_2$ . The inputs and outputs of these latches are given as

Latch	Inputs	Outputs
$L_1$	$S_1, R_1$	$Q_1, \bar{Q}_1$
$L_2$	$S_2, R_2$	$Q_2, \bar{Q}_2$

The excitation table of an S-R latch is given in Table 8.2. This is same as for S-R flip-flop.

**Table 8.2 Excitation table of S-R latch**

Present state Q	Next state $Q^+$	Inputs	
		S	R
0	0	0	×
0	1	1	0
1	0	0	1
1	1	×	0

To determine  $S_1$  and  $R_1$  for different values of  $X_1 X_2$ , we make use of  $Q_1$  and  $Q_1^+$  values for every square of transition table. For example, this square in the first row and first column gives  $Q_1 = 0$  and  $Q_1^+ = 0$ . This means, for the present state 0 the circuit gives next state as 0 for  $Q_1$ . Corresponding to this we find the value of  $S_1$  and  $R_1$  using the Table 8.2, which are  $S_1 = 0$  and  $R_1 = X$ .

Thus the entry in the cell corresponding to  $X_1 X_2 = 00$  and  $Q_1 Q_2 = 00$  for K-map of  $S_1$  will be 0 and for K-map of  $R_1$  it will be X. Similarly, K-map entries are determined for  $S_1$  and  $R_1$ .

Following similar procedure, K-maps for  $S_2$  and  $R_2$  are constructed. The K-maps are given in Fig. 8.11.

From the K-map of Fig. 8.11, we obtain logic equations for  $S_1, R_1, S_2,$  and  $R_2$ .

$$S_1 = X_1 \bar{X}_2 + \bar{X}_1 X_2 Q_2$$

$$R_1 = \bar{X}_1 + X_1 X_2 Q_2$$

$$S_2 = \bar{X}_1 X_2 \bar{Q}_2$$

$$R_2 = \bar{X}_1 \bar{X}_2$$

The logic circuit is shown in Fig. 8.12.

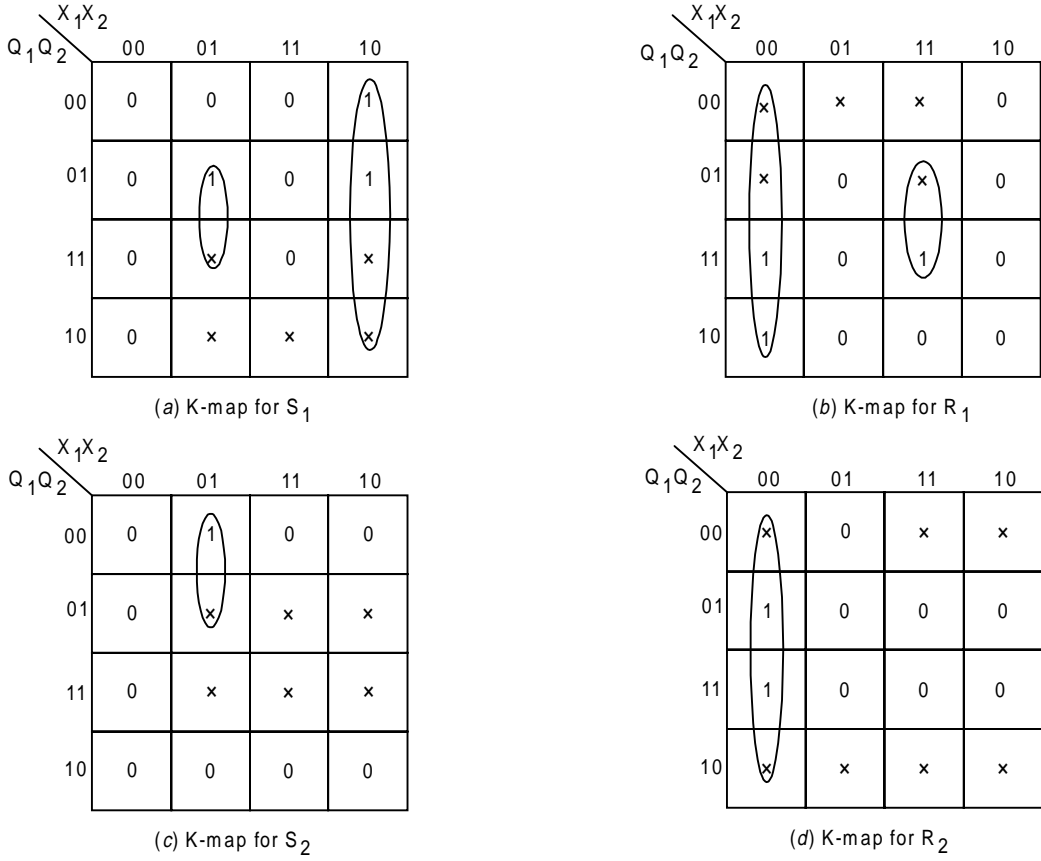


Fig. 8.11

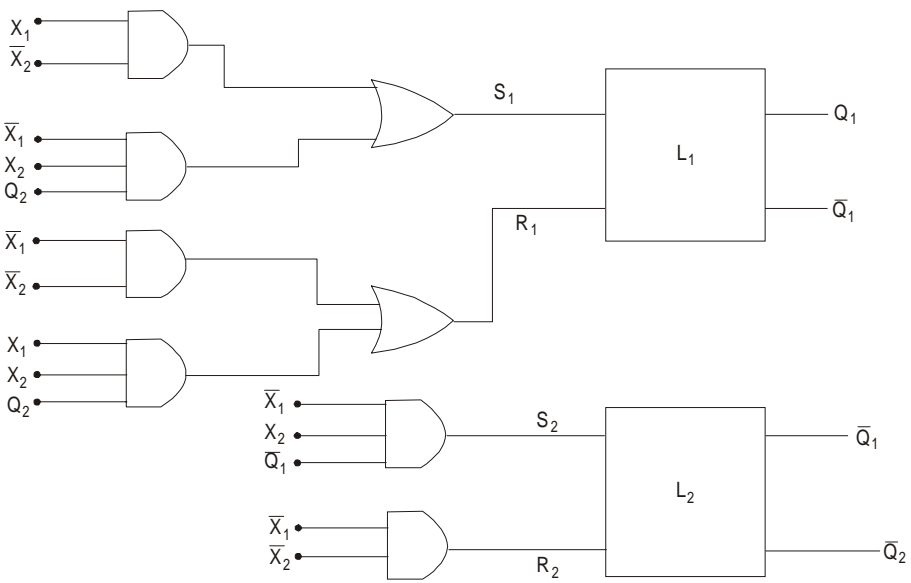


Fig. 8.12

### 8.3.6 Races and Cycles

A *race* condition exists in an asynchronous sequential circuit when more than one state variable change value in response to a change in an input variable. This is caused because of unequal propagation delays in the path of different secondary variables in any practical electronic circuit. Consider a transition table shown in Fig. 8.13. When both the inputs  $X_1$  and  $X_2$  are 0 and the present state is  $Q_1 Q_2 = 00$ , the resulting next state  $Q_1^+ Q_2^+$  will have  $Q_1^+ = 1$  and  $Q_2^+ = 1$  simultaneously if the propagation delays in the paths of  $Q_1$  and  $Q_2$  are equal.

$X_1 X_2$		$Q_1^+ Q_2^+$			
		00	01	11	10
$Q_1 Q_2$	00	11	00	10	01
	01	11	00	11	01
	11	11	00	10	11
	10	11	10	10	11

Fig. 8.13

Since  $Q_1$  and  $Q_2$  both are to change and in general the propagation delays in the paths of  $Q_1$  and  $Q_2$  are not same, therefore, either  $Q_1$  or  $Q_2$  may change first instead of both changing simultaneously. As a consequence of this the circuit will go to either state 01 or to state 10.

If  $Q_2^+$  changes faster than  $Q_1^+$ , the next state will be 01, then 11 (first column, second row) and then to the stable state  $\textcircled{11}$  (first column, third row) will be reached. On the other hand, if  $Q_1^+$  changes faster than  $Q_2^+$ , the next-state will be 10, then 11 (first column, fourth row) and then to the stable state (first column, third row) will be reached. In both the situations, the circuit goes to the same final stable state  $\textcircled{11}$ . This situation, where a change of more than one secondary variable is required is known as a *race*.

There are two types of races: *noncritical race* and *critical race*.

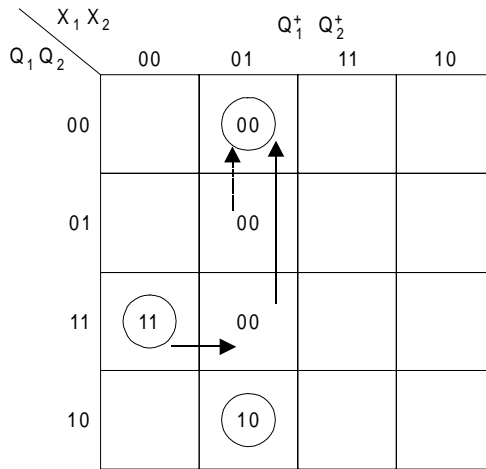
In the case of noncritical race, the final stable state in which the circuit goes does not depend on the sequence in which the variables change. The race discussed above is a noncritical race. In the case of critical race, the final stable state reached by the circuit depends on the sequence in which the secondary variables change. Since the critical race results in different stable states depending on the sequence in which the secondary states change, therefore, it must be avoided.

**Example.** In the transition table of Fig. 8.13, consider the circuit in stable total state 1100. Will there be any race, if the input state changes to 01? If yes, find the type of race.

**Solution.** When the circuit is in stable total state,  $X_1 X_2 = 00$ . Now  $X_2$  changes to 1 while  $X_1 = 0$ . From Fig. 8.13 we see that the required transition is to state 00. If  $Q_1^+$  and  $Q_2^+$  become 00 simultaneously, then the transition will be

$$\textcircled{11} \rightarrow 00 \rightarrow \textcircled{00}$$

These transitions are shown by solid arrows in Fig. 8.14.



**Fig. 8.14**

If  $Q_2^+$  becomes 0 faster than  $Q_1^+$ , the circuit will go to the state 10 and then to  $\textcircled{10}$ , which is a stable state. The transition is

$$\textcircled{11} \rightarrow 10 \rightarrow \textcircled{10}$$

On the other hand, if  $Q_1^+$  becomes 0 faster than  $Q_2^+$ , the transition will be

$$\textcircled{11} \rightarrow 01 \rightarrow 00 \rightarrow \textcircled{00}$$

It is shown by dotted arrow in Fig. 8.13. Thus, we see that the circuit attains different stable states  $\textcircled{00}$  or  $\textcircled{10}$  depending upon the sequence in which the secondary variables change.

Therefore, the race condition exists in this circuit and it is critical race.

Races can be avoided by making a proper binary assignment to the state variables in a flow table. The state variables must be assigned binary numbers in such a way so that only one state variable can change at any one time when a state transition occurs in the flow table. The state transition is directed through a unique sequence of unstable state variable change. This is referred to as a *cycle*. This unique sequence must terminate in a stable state, otherwise the circuit will go from one unstable state to another unstable state making the entire circuit unstable.

### 8.3.7 Pulse-mode Circuits

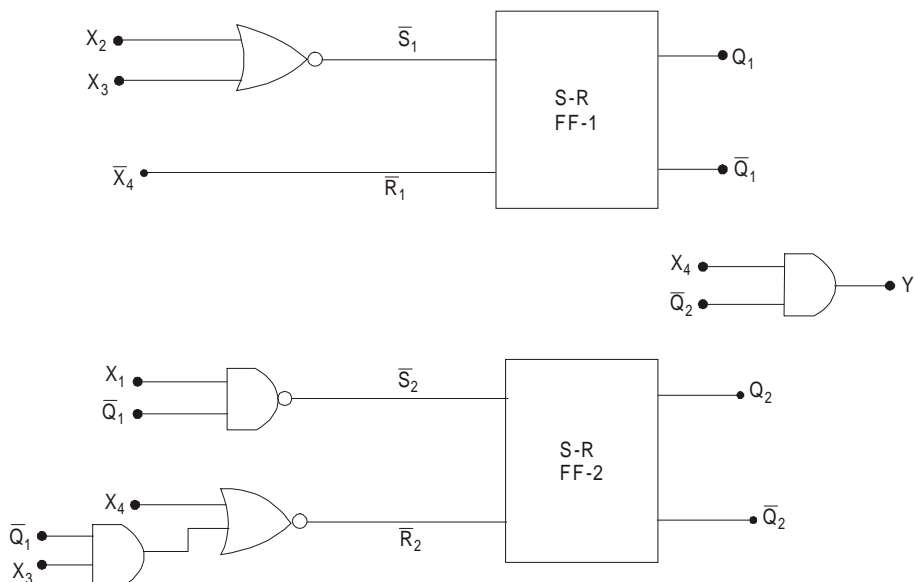
In a pulse-mode asynchronous sequential circuit, an input pulse is permitted to occur only when the circuit is in stable state and there is no pulse present on any other input.

When an input pulse arrives, it triggers the circuit and causes a transition from one stable state to another stable state so as to enable the circuit to receive another input pulse. In this mode of operation critical race can not occur. To keep the circuit stable between two pulses, flip-flops whose outputs are levels, must be used as memory elements.

For the analysis of pulse-mode circuits, the model used for the fundamental-mode circuits is not valid since the circuit is stable when there are no inputs and the absence of a pulse conveys no information. For this a model similar to the one used for synchronous sequential circuits will be convenient to use.

In pulse-mode asynchronous circuits the number of columns in the next-state table is equal to the number of input terminals.

Consider a pulse-mode circuit logic diagram shown in Fig. 8.15. In this circuit there are four input variables  $X_1$ ,  $X_2$ ,  $X_3$ , and  $X_4$ , and  $Y$  is the output variable. It has two states  $Q_1$  and  $Q_2$ .



**Fig. 8.15**

The excitation equations are:

$$\bar{S}_1 = \overline{(X_2 + X_3)} \quad \text{or} \quad S_1 = X_2 + X_3$$

$$\bar{R}_1 = \bar{X}_4 \quad \text{or} \quad R_1 = X_4$$

$$\bar{S}_2 = \overline{\bar{Q}_1 X_1} \quad \text{or} \quad S_2 = \bar{Q}_1 X_1$$

$$\bar{R}_2 = \overline{(X_4 + \bar{Q}_1 X_3)} \quad \text{or} \quad R_2 = X_4 + \bar{Q}_1 X_3$$

The output equation is:  $Y = X_4 \bar{Q}_2$

The next-state equations are obtained by using the excitation equations and the characteristic equation of latch.



These are:

$$Q_1^+ = S_1 + \bar{R}_1 Q_1$$

$$= X_2 + X_3 + \bar{X}_4 Q_1$$

and

$$Q_2^+ = S_2 + \bar{R}_2 Q_2$$

$$= \bar{Q}_1 X_1 + \overline{(X_4 + \bar{Q}_1 X_3)} \cdot Q_2$$

$$= \bar{Q}_1 X_1 + \bar{X}_4 \cdot \overline{(Q_1 X_3)} \cdot Q_2$$

$$= \bar{Q}_1 X_1 + \bar{X}_4 \cdot (Q_1 + \bar{X}_3) \cdot Q_2$$

$$= \bar{Q}_1 X_1 + Q_1 Q_2 \bar{X}_4 + Q_2 \bar{X}_3 \bar{X}_4$$

The transition table is constructed by evaluating the next-state and output for each present state and input value using next-state equations and output equation. The transition table is shown in Fig. 8.16.

Present State $Q_1 Q_2$	Input variables				Next-state value	Output value
	$X_1$	$X_2$	$X_3$	$X_4$		
00	01, 0	10, 0	10, 0	00, 1		
01	01, 0	11, 0	10, 0	00, 0		
11	11, 0	11, 0	11, 0	00, 0		
10	10, 0	10, 0	10, 0	10, 0		

**Fig. 8.16**

It has four rows (one row for each combination of state variables) and four columns (one column for each input variable). Since in pulse-mode circuits only one input variable is permitted to be present at a time, therefore, the columns are for each input variable only and not for the combinations of input variables.

Flow table can be constructed from the transition table and is shown in Fig. 8.17. Here,  $S_0, S_1, S_2,$  and  $S_3$  are the four state variables.

Present State $Q_1 Q_2$	Input variables			
	$X_1$	$X_2$	$X_3$	$X_4$
$S_0$	$S_1, 0$	$S_3, 0$	$S_3, 0$	$S_0, 1$
$S_1$	$S_1, 0$	$S_2, 0$	$S_3, 0$	$S_0, 0$
$S_2$	$S_2, 0$	$S_2, 0$	$S_2, 0$	$S_0, 0$
$S_3$	$S_3, 0$	$S_3, 0$	$S_3, 0$	$S_0, 1$

**Fig. 8.17**

From a flow table a transition table can be constructed by assigning binary values to the states. From a transition table next-state equations can be obtained and the logic diagram can then be obtained.

## 8.4 ASYNCHRONOUS SEQUENTIAL CIRCUIT DESIGN

Design of asynchronous sequential circuits is more difficult than that of synchronous sequential circuits because of the timing problems involved in these circuits. Designing an asynchronous sequential circuit requires obtaining logic diagram for the given design specifications. Usually the design problem is specified in the form of statements of the desired circuit performance precisely specifying the circuit operation for every applicable input sequence.

### 8.4.1 Design Steps

1. Primitive flow table is obtained from the design specifications. When setting up a primitive flow table it is not necessary to be concerned about adding states which may ultimately turn out to be redundant. A sufficient number of states are to be included to completely specify the circuit performance for every allowable input sequence. Outputs are specified only for stable states.
2. Reduce the primitive flow table by eliminating the redundant states, which are likely to be present. These redundant states are eliminated by merging the states. *Merger diagram* is used for this purpose.
3. Binary numbers are assigned to the states in the reduced flow table. The binary state assignment must be made to ensure that the circuit will be free of critical races. The output values are to be chosen for the unstable states with unspecified output entries. These must be chosen in such a way so that momentary false outputs do not occur when the circuit switches from one stable state to another stable state.
4. Transition table is obtained next.
5. From the transition table logic diagram is designed by using the combinational design methods. The logic circuit may be a combinational circuit with feedback or a circuit with *S-R* latches.

The above design steps is illustrated through an example.

**Example.** *The output ( $Y$ ) of an asynchronous sequential circuit must remain 0 as long as one of its two inputs  $X_1$  is 0. While  $X_1 = 1$ , the occurrence of first change in another input  $X_2$  should give  $Y = 1$  as long as  $X_1 = 1$  and becomes 0 where  $X_1$  returns to 0. Construct a primitive flow table.*

**Solution.** This circuit has two inputs  $X_1, X_2$  and one output  $Y$ . For the construction of flow table, the next-state and output are required to be obtained. The flow table is shown in Fig. 8.18.

For  $X_1 X_2 = 00$ , let us take state a. When the circuit has  $X_1 X_2 = 00$  the output is 0 (since  $X_1 = 0$ ) and the circuit is in stable state (a). The next-state and output are shown in the first column, first row of Fig. 8.18. Since only one input is allowed to change at a time, therefore, the next input may be  $X_1 X_2 = 01$  or 10.

If  $X_1 X_2 = 01$ , let us take another state b, correspondingly the second row of the flow table corresponds to state b. when the inputs change from  $X_1 X_2 = 00$  to 01, the circuit is

required to go to stable state (b) and output is 0 (since  $X_1 = 0$ ). Therefore, the entry in the second column, first row will be b, 0 and in the second column, second row will be (b), 0. The output corresponding to unstable state b is taken as 0 so that no momentary false outputs occur when the circuit switches between stable states. On the other hand if  $X_1 X_2 = 10$ , the circuit is required to go to another stable state (c) with output 0. Therefore, the entries in the fourth column, first row and fourth column, third row will be respectively c, 0 and (c), 0.

Present-state	$X_1 X_2$					
	00	01	11	10		
a	(a), 0	b, 0	-, -	c, 0		
b	a, 0	(b), 0	d, 0	-, -		
c	a, 0	-, -	e, -	(c), 0		
d	-, -	b, 0	(d), 0	f, -		
e	-, -	b, -	(e), 1	f, 1		
f	a, -	-, -	e, 1	(f), 1		

**Fig. 8.18 Flow table**

Since both the inputs cannot change simultaneously, therefore, from stable state (a), the circuit cannot go to any specific state corresponding to  $X_1 X_2 = 11$  and accordingly the entry in the third column, first row will be -, -. The dashes represent the unspecified state, output.

Now consider the stable state (b). The inputs  $X_1 X_2$  can change to 00 or 11. If  $X_1 X_2 = 00$ , the circuit will go to state a. Therefore, the entry in the first column, second row will be a, 0. From this unstable state the circuit goes to stable state (a). On the other hand if  $X_1 X_2 = 11$ , then the circuit goes to a new state d. The output corresponding to  $X_1 X_2 = 11$  will be 0 since, there is no change in  $X_2$ , which is already 1. Therefore, the entry in the third column, second row will be d, 0. The fourth row corresponds to state d, and the entry in the third column, fourth row, will be (d), 0. From (b), the circuit is not required to go to any specific state and therefore, the entry in the fourth column, second row will be -, -.

Similarly, now consider stable state (c). The inputs can change to  $X_1 X_2 = 11$  or 00. If  $X_1 X_2 = 11$ , the circuit goes to a new stable state (e) and the output will be 1, since  $X_2$  changes from 0 to 1 while  $X_1 = 1$ . The entry in the third column, third row will be c, -. Output has to change from 0 to 1 from stable state (c) to stable state (e), which may or may not change to 1 for unstable e. The entry in the third column, fifth row will be (e), 1. The entry in the second column third row will be -, - and the entry in the first column, third row will be a, 0 (for  $X_1 X_2 = 00$ ).

In the same manner, we consider the stable (d) and obtain the entries f, - (fourth column, fourth row); (f), 1 (fourth column, sixth row); b, 0 (second column, fourth row) and -, - (first column, fourth row).

Similar procedure applied to (e) and (f), yields the remaining entries of the flow table.

Since, every row in the flow table of Fig. 8.18 contains only one stable state, therefore, this flow table is a primitive flow table.

#### 8.4.2 Reduction of States

The necessity of reducing the number of states has been discussed in chapter 6 and the equivalent states have been defined. When asynchronous sequential circuits are designed, the design process starts from the construction of primitive flow table. A primitive flow table is never completely specified. Some states and outputs are not specified in it as shown in Fig. 8.18 by dashes. Therefore, the concept of equivalent states can not be used for reduction of states. However, incompletely specified states can be combined to reduce the number of states in the flow table. Two incompletely specified states can be combined if they are *compatible*.

Two states are *compatible* if and only if, for every possible input sequence both produce the same output sequence whenever both outputs are specified and their next states are compatible whenever they are specified. The unspecified outputs and states shown as dashes in the flow table have no effect for compatible states.

**Example.** In the primitive flow table of Fig. 8.18, find whether the states a and b are compatible or not. If compatible, find out the merged state.

**Solution.** The rows corresponding to the states a and b are shown in Fig. 8.19. Each column of these two states is examined.

$X_1 X_2$	00	01	11	10
a	(a), 0	b, 0	-, -	c, 0
b	a, 0	(b), 0	d, 0	-, -

Fig. 8.19

**Column-1.** Both the rows have the same state a and the same output 0. a in first row is stable state and in the second row is unstable state.

Since for the same input both the states a and b have the same specified next-state a and the same specified output 0. Therefore, this input condition satisfies the requirements of compatibility.

**Column-2.** The input condition  $X_1 X_2 = 01$  satisfies the requirements of compatibility as discussed for column-1.

**Column-3.** The first row has unspecified next-state and output and the second row has specified state and output. The unspecified state and output may be assigned any desired state and output and therefore, for this input condition also the requirements of compatibility are satisfied.

**Column-4.** The requirements of compatibility are satisfied for the reasons same as applicable to column-3.

Therefore, we conclude that since the next-states and the outputs for all the input combinations are compatible for the two states a and b, the two states are compatible. The merged state will be as shown in Fig. 8.20

		$X_1 X_2$			
		00	01	11	10
a	Ⓐ, 0	Ⓑ, 0	d, 0	c, 0	

**Fig. 8.20**

When the merged state entries are determined a circled entry and an uncircled entry results in a circled entry, since the corresponding state must be stable as shown in Fig. 8.20.

**Example.** In the primitive flow table of Fig. 8.18 find whether the states *a* and *e* are compatible or not. Examine their compatibility if the entries in the fourth column for the states *a* and *e* have same output.

**Solution.** The partial flow table for states *a* and *e* of Fig. 8.18 is shown in Fig. 8.21.

		$X_1 X_2$			
		00	01	11	10
a	Ⓐ, 0	b, 0	-, -	c, 0	
	e	-, -	b, -	Ⓔ, 1	f, 1

**Fig. 8.21**

From this we observe the following

Column-1 compatible

Column-2 compatible

Column-3 compatible

Column-4 not compatible, since the outputs are different.

Therefore, the states *a* and *e* are not compatible.

In case of same output in column-4, the outputs are said to be not conflicting and the states *a* and *e* are compatible if and only if the states *c* and *f* are compatible. This is referred to as *c, f* is implied by *a, b* or *a, b* implies *c, f*.

### 8.4.3 Merger Diagram

A *merger diagram (or graph)* is prepared for a primitive flow table to determine all the possible compatible states (maximal compatible states) and from this a minimal collection of compatibles covering all the states.

A merger graph is constructed following the steps outlined below:

- Each state is represented by a vertex, which means it consists of  $n$  vertices, each of which corresponds to a state of the circuit for an  $n$ -state primitive flow table. Each vertex is labelled with the state name.
- For each pair of compatible states an undirected arc is drawn between the vertices of the two states. No arc is drawn for incompatible states.
- For compatible states implied by other states a broken arc is drawn between the states and the implied pairs are entered in the broken space.

The flow table is required to be examined for all the possible pairs of states. All the pairs are checked and the merger graph is obtained. Thus we see that the merger graph displays all possible pairs of compatible states and their implied pairs. Next it is necessary to check whether the incompatible pair(s) does not invalidate any other implied pair. If any implied pair

is invalidated it is neglected. All the remaining valid compatible pairs form a group of *maximal* compatibles.

The maximal compatible set can be used to construct the reduced flow table by assigning one row to each member of the group. However, the maximal compatibles do not necessarily constitute the set of *minimal* compatibles. The set of minimal compatibles is a smaller collection of compatibles that will satisfy the row merging.

The conditions that must be satisfied for row merging are:

- the set of chosen compatibles must *cover* all the states, and
- the set of chosen compatibles must be *closed*.

The condition of covering requires inclusion of all the states of the primitive flow graph in the set of chosen compatibles. This condition only defines a *lower bound* on the number of states in the minimal set. However, if none of their implied pairs are contained in the set, the set is not sufficient and this is referred to as *closed* condition not being satisfied. Therefore, condition of *closed covering* is essentially required for row merging.

### 8.5 ESSENTIAL HAZARDS

Similar to static and dynamic hazards in a combinational circuits, essential hazards occur in sequential circuits. Essential hazard is a type of hazard that exists only in asynchronous sequential circuits with two or more feedbacks. Essential hazard occurs normally in toggling type circuits. It is an error generally caused by an excessive delay to a feedback variable in response to an input change, leading to a transition to an improper state. For example, an excessive delay through an inverter circuit in comparison to the delay associated with the feedback path many cause essential hazard. Such hazards cannot be eliminated by adding redundant gates as in static hazards. To avoid essential hazard, 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.

Even though an asynchronous sequential circuit (network) is free of critical races and the combinational part of the network is free of static and dynamic hazards, timing problems due to propagation delays may still cause the network to malfunction and go to the wrong state. To better understand, consider for example the network of Fig. 8.22.

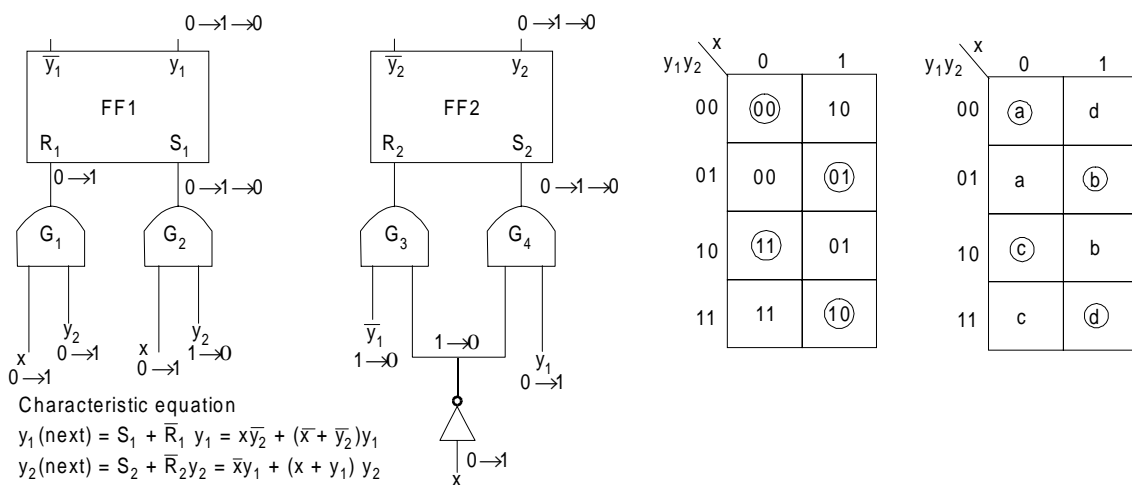


Fig. 8.22 Network with essential hazards.

There is no hazards in the combinational part of the network, and flow table inspection shows that there are no critical races. If we start in state ③ and change  $x$  to 1, the network should go to state ④. Let consider the following possible sequence of events.

- (i)  $x$  change 0 to 1.
- (ii) Gate 3 (G2) output changes 0 to 1.
- (iii) Flip-flop (FF1) output  $y_1$  changes 0 to 1.
- (iv) G4 output changes 0 to 1.
- (v) FF2 output changes 0 to 1.
- (vi) Inverter output  $\bar{x}$  changes 1 to 0.
- (vii) G1 output changes 0 to 1, G2 output changes back to 0, and G4 output changes back to 0.
- (viii) Flip-flop output  $y_1$  changes back to 0.

Though the network should go to stage ④ when change  $x$  to 1 but the final state of the network is ② instead of ④. The malfunction illustrated in example network of figure is referred to as an essential hazard. This came about because the delay in inverter was large than the other delays in the network, so that part of the network having value  $x = 1$  while other part have value  $x = 0$ . The final result was that the network acted as if the input  $x$  had changed three times instead of once so that the network went through the sequence of states  $y_1y_2 = 00, 10, 11, 01$ . Essential hazards can be located by inspection of the flow table. An essential hazard can be defined as follows:

A flow table has an essential hazard starting in stable total state ⑤ for input variable  $x_i$  if and only if the stable total state reached after one change in  $x_i$  different froms the stable total state reached after three changes in  $x_i$ .

If an essential hazard exists in the flow table for total stable state ⑤ and input  $x_i$ , then due to some combination of propagation delays network go to the wrong state when  $x_i$  is changed starting in ⑤ on realization. This occurs because the change in  $x_i$  reaches different parts of the network at different times.

In order to test a flow table for essential hazards it is necessary to test each stable total for each possible input change using the definition of essential hazard given.

Essential hazards can be eliminated by adding delays to the network. For the network show in figure, the essential hazard can be eliminated by adding a sufficiently large delay to the output of FF1, because he change in  $x$  output of FF1 does.

We can summarize the design of an asynchronous network is free of timing problems as:

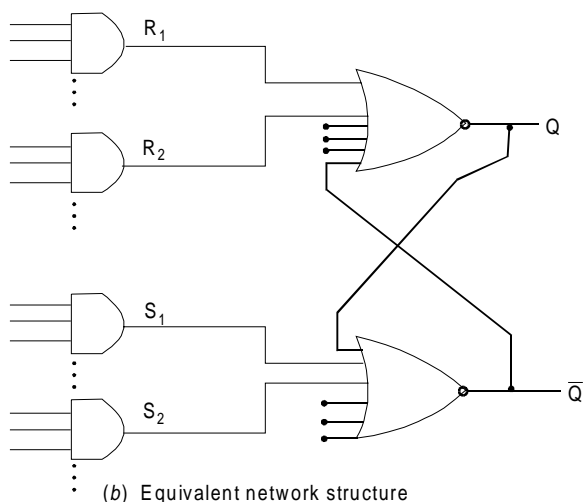
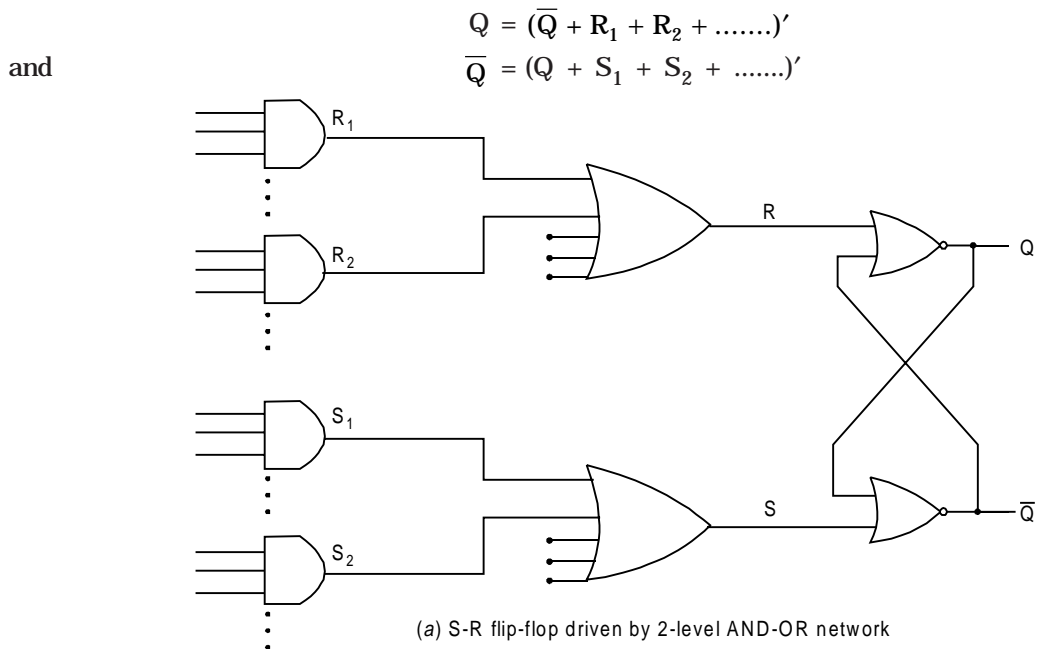
- (i) Make a state assignment which is free of critical races.
- (ii) Design the combinational part of the network so that it is free of hazards (if require by adding redundant gates).
- (iii) Add delays in the feedback paths for the state variables as required to eliminate essential hazards.

## 8.6 HAZARD-FREE REALIZATION USING S-R FLIP-FLOPS

The design of hazard-free asynchronous networks can be simplified using S-R flip-flops. We have already seen in chapter 6 that a momentary 1 applied to the S or R input can set or reset the flip-flop, however a momentary 0 applied to S or R will have no effect on the flip-

plop state. Since a 0-hazard can produce a momentary false 1, the networks realizing S and R must be free of 0-hazards but the S and R networks may contain 1-hazards. A minimum two-level sum of products expression is free of 0-hazards but it may contain 1-hazards. For this reason, the minimum sum of products can be used as a starting point for realizing the S-R flip-flop input equations. Simple factoring or transformation which do not introduce 0-hazards can be applied to the minimum sum-of-products expressions, in the process of realizing S and R.

A typical network structure with the S-R flip-flop driven by 2-level AND-OR networks constructed from cross-coupled NOR gates is shown in Fig. 8.23(a). The Fig. 8.23(b) shows equivalent network structure with multiple input NOR gates. The two structure are equivalent since in both cases.



**Fig. 8.23** Gate structures for S-K flop-flip realization of flow table.



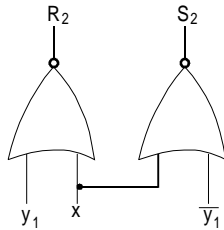
Even if an asynchronous network is realized using S-R flip-flops and S and R networks are free of 0-hazards, essential hazards may still be present. Such essential hazards may be eliminated as discussed previously by adding delays in the feedback paths for the state variables.

An alternative method for eliminating essential hazards involves changing the gate structure of the network. This method can be applied only if wiring delays are negligible and all the gate delays are concentrated at the gate outputs.

As illustrated in previous section, the following sequence of events is needed for an essential hazard to cause a network of malfunction.

- (i) An input variable changes.
- (ii) A state variable changes in response to the input variable change.
- (iii) The effect of the state variable change propagates through the network and initiates another state variable change before.
- (iv) The original input variable change has propagated through the entire network.

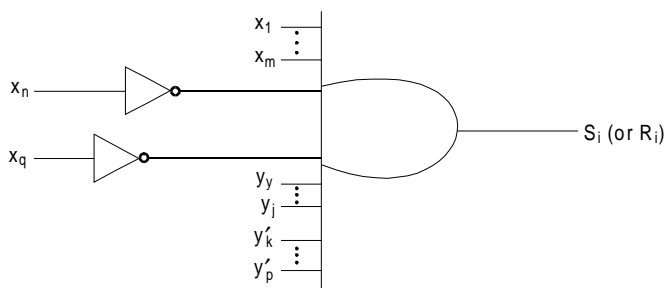
Therefore, in an asynchronous network with S-R flip-flops, we can eliminate the essential hazards by arranging the gate structure so that the effect of any input change will propagate to all flip-flop inputs before any state variable changes can propagate back to the flip-flop inputs. For example, the essential hazard of Fig. 8.24 can be eliminated by replacing the  $R_2$  and  $S_2$  networks with the network of Fig. 8.24.



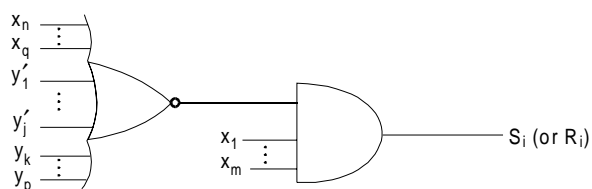
**Fig. 8.24**

Assuming that wiring delays are negligible that the gate delay is concentrated at the gate output any change in  $x$  will propagate to  $R_2$  and  $S_2$  before flip-flop 1 output  $y_1$  can change state and this change in  $y_1$  can propagate to  $R_2$  and  $S_2$ . This eliminates the essential hazard.

In the Fig. 8.23 (b), each AND gate can have inputs of the form shown in Fig. 8.25 (a), where  $x$ 's are external inputs to the circuit, and the  $y$ 's are feedback from flip-flop outputs. If there are essential hazards in the flow table, then the circuit could malfunction due to the inverter delays. By replacing the AND gate with the NOR-AND network of Fig. 8.25 (b), the inverters on the  $x$  variables are eliminated. Therefore by replacing all of the AND gate in Fig. 8.23 with the NOR-AND combinations as indicated in Fig. 8.25, all of the estimates hazards will be eliminated.



(a) AND gate with general inputs



(b) Replacement for (a)

**Fig. 8.25** A gate transformation for elimination of essential hazards.

## 8.7 SOLVED EXAMPLES

**Example 1.** Construct merger diagram for the primitive flow table of Fig. 8.18. Determine maximal compatibles and the minimal set of compatibles.

**Solution:** For construction of merger diagram, every row of the primitive flow table is checked with every other row to determine compatibility of states.

Consider row-1 (state a)

- a, b are compatible
- a, c are compatible
- a, d are compatible if c, f are compatible
- a, e are compatible if c, f are compatible
- a, f are compatible if c, f are compatible

row-2 (state b)

- b, c are compatible if d, e are compatible
- b, d are compatible
- b, e are not compatible (outputs are conflicting)
- b, f are not compatible (outputs are conflicting)

row-3 (state c)

- c, d are compatible if e, d and c, f are compatible

c, e are not compatible (outputs are conflicting)  
 c, f are not compatible (outputs are conflicting)

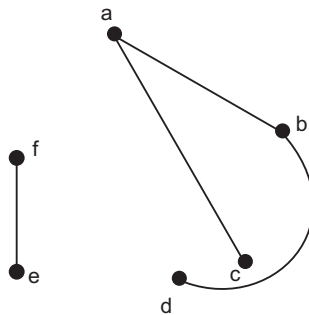
row-4 (state d)

d, e are not compatible (outputs are conflicting)  
 d, f are not compatible (outputs are conflicting)

row-5 (state e)

e, f are compatible

The primitive flow table has six states therefore, there are six vertices in the merger diagram as shown in Fig. 8.26.



**Fig. 8.26 Merger diagram**

Solid arcs are drawn between (a, b), (a, c), (b, d) and (f, e) vertices. Corresponding to these states being compatibles. Since (c, f) and (d, e) are not compatible, therefore, there are no implied pairs available.

From the merger diagram, we get the maximal compatibles:

(a, b), (a, c), (b, d), (e, f)

Since (a, b) is covered by (a, c) and (b, d), therefore, the minimal set is (a, c), (b, d), (e, f)

**Example 2.** Determine the reduced flow table of Fig. 8.19.

**Solution:** From the merger diagram, we have obtained three pairs of compatible states: These compatibles are merged and are represented by

a, c :  $S_0$

b, d :  $S_1$

e, f :  $S_2$

The reduced flow table is shown in Fig. 8.27

		$X_1X_2$			
		00	01	11	10
Present-state	$S_0$	$\textcircled{S_0}, 0$	$S_1, 0$	$S_2, -$	$\textcircled{S_0}, 0$
	$S_1$	$S_0, 0$	$\textcircled{S_1}, 0$	$\textcircled{S_1}, 0$	$S_2, -$
	$S_2$	$S_0, -$	$S_1, -$	$\textcircled{S_2}, 1$	$\textcircled{S_2}, 1$

**Fig. 8.27 Reduced flow table**

**Example 3.** Assign binary states to the reduced flow table of Fig. 8.27. Avoid critical race.

**Solution:** Let us assign the following binary states to S0, S1, and S2 for the reduced flow table of Fig. 8.27

$$S0 \rightarrow 00$$

$$S1 \rightarrow 01$$

$$S2 \rightarrow 11$$

The transition table will be as shown in Fig. 8.28

		$X_1X_2$			
		00	01	11	10
$Q_1Q_2$	00	00, 0	01, 0	11, -	00, 0
	01	00, 0	01, 0	01, 0	11, -
	11	00, -	01, -	11, 1	11, 1
	10				

**Fig. 8.28** Transition table

In the transition table of Fig. 8.28, we observe that race condition occurs in the following cases:

- (i) From stable state 00 to unstable state 11 when  $X_1 X_2$  changes from 10 to 11.
- (ii) From stable state 11 to unstable state 00 when  $X_1 X_2$  changes from 10 to 00.

To avoid critical race, one unstable state 10 is added with the entries 00, -; -, -; 11, -; -, - and the entries in third column, first row is changed from 11, - to 10, - and in first column, third row from 00, - to 10, -.

The modified transition table is given in Fig. 8.29.

		$X_1X_2$			
		00	01	11	10
$Q_1Q_2$	00	00, 0	01, 0	10, -	00, 0
	01	00, 0	01, 0	01, 0	11, -
	11	10, -	01, -	11, 1	11, 1
	10	00, -	-, -	11, -	-, -

**Fig. 8.29** Modified transition table

**Example 4.** Design logic circuit with feedback for the transition table of Fig. 8.29.

**Solution:** The K-maps for  $Q_1^+$ ,  $Q_2^+$ , and Y determined from the transition table are given in Fig. 8.30.

From the K-maps, we obtain,

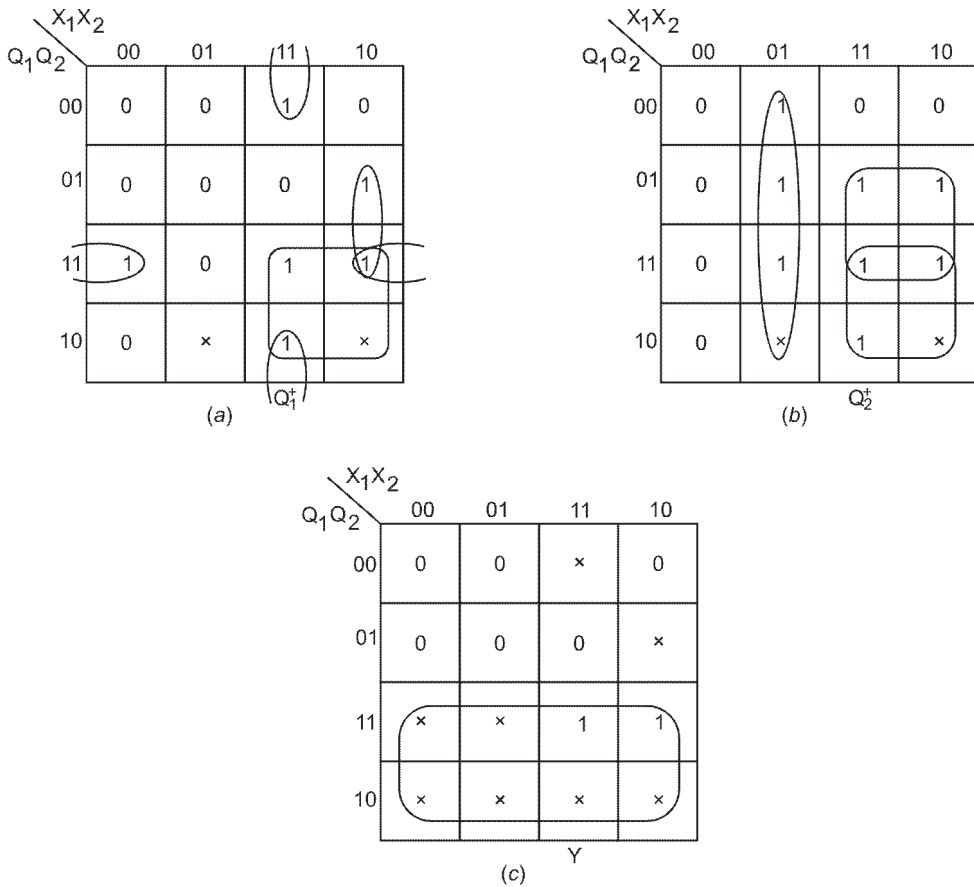
$$Q_1^+ = X_1 X_2 \bar{Q}_2 + \bar{X}_2 Q_1 Q_2 + X_1 \bar{X}_2 Q_2 + X_1 Q_1$$

$$Q_2^+ = \bar{X}_1 X_2 + X_1 Q_2 + X_1 Q_1$$

$$Y = Q_1$$

Logic circuit using gates can be obtained from the above logic equations.

Thus we see that the design steps outlined above can be used to design an asynchronous sequential circuit.



**Fig. 8.30** K-Maps for (a)  $Q_1^+$  (b)  $Q_2^+$  (c) Y

**Example 5.** In the state transition table of Fig. 8.13, if  $X_1 X_2 = 10$  and the circuit is in stable state  $\textcircled{01}$ , find the cycle when  $X_2$  is changed to 1 while  $X_1$  remaining 1.

**Solution:** The circuit is in stable state  $\textcircled{01}$  (fourth column, second row). When  $X_2$  changes to 1, the circuit will go to the state 11 (third column, second row), then to state 10

(third column, third row) and finally to the stable state  $\textcircled{10}$  (third column, fourth row). Thus the cycle is

$$\textcircled{01} \rightarrow 11 \rightarrow 10 \rightarrow \textcircled{10}$$

### 8.8 EXERCISES

- Explain the difference between asynchronous and synchronous sequential circuits.
  - Define fundamental mode of operation.
  - Define pulse mode of operation
  - Explain the difference between stable and unstable states.
  - What is the difference between internal state and total state.
- Describe the design procedure for asynchronous sequential circuits.
- What do you mean by critical and non-critical races? How can they be avoided.?
- Describe cycles in asynchronous sequential circuits.
- Design a JK flip-flop asynchronous sequential circuit that has two inputs and single output. The circuit is required to give an output equal to 1 if and only if the same input variable changes two or more times consecutively.
- Design an asynchronous circuit that has two inputs and single output .The circuit is required to give an output whenever the input sequence 00,10,11 and 01 are received but only in that order.
- Design an asynchronous binary counter with one pulse input and two outputs, capable of counting from zero to three .When the circuit is pulsed after the count has reached three, it should return to zero. The output should provide continuously the count modulo 4.
  - Repeat the problem for level inputs and outputs.
- Find all of the essential hazards in the following flow table. How can table essential hazard which occurs starting in b be eliminated.

		$X_1X_2$			
		00	01	11	10
a	$Q_1Q_2$ 00	$\textcircled{a}$	b	$\textcircled{a}$	d
b	01	a	$\textcircled{b}$	c	–
c	11	–	d	$\textcircled{c}$	d
d	10	a	$\textcircled{d}$	a	$\textcircled{d}$

# ALGORITHMIC STATE MACHINE

---

## 9.0 INTRODUCTION

Finite state machines are a powerful tool for designing sequential circuits, but they are lacking in that they do not explicitly represent the algorithms that compute the transition or output functions, nor is timing information explicitly represented. We can recast the idea of a state machine to include a representation of the algorithms. The result is an *algorithmic state machine*, or ASM. “The ASM chart separates the conceptual phase of a design from the actual circuit implementation.” An algorithmic state machine diagram is similar to a flowchart but with some differences. Square boxes represent the states, diamonds represent decisions, and ovals represent outputs. States can also have outputs, and the outputs associated with a state are listed in the state box. State boxes are labelled with the state name and possibly with a binary code for the state. The basic unit of an ASM is the ASM block. An ASM block contains a single state box, a single entry path, and one or more exit paths to other ASM blocks. Algorithmic state machines capture the timing of state transitions as well.

## 9.1 DESIGN OF DIGITAL SYSTEM

In the earlier chapters we have presented the analysis and design of various types of Digital System for specified task. A close look on all such systems reveal that these systems can be viewed as collection of two subsystems.

- (i) The Data Processing or manipulating subsystem which include the operation such as shifting, Adding, counting, dividing etc.
- (ii) The control subsystem or simply control. This subsystem has to initiate, supervise and sequence the operation in data processing unit.

Usually design of data processor is a fair and simple design. But design of control logic with available resources is a complex and challenging part, perhaps because of timing relations between the event. And in this chapter we are majority concerned with design of control.

The control subsystem is a sequential circuit whose internal states dictate the control command to sequence the operations in data processing unit. The digital circuit used as control subsystem is responsible to generate a time sequence of control signals that initiates operation in data processor, and to determine the next state of control subsystem itself. The task of data processing and control sequence are specified by means of a **hardware algorithm**. An algorithm is a collection of **produces** that tells how to obtain the solution. A **flow chart** is a simple way to represent the sequence of procedures and decision paths for algorithm.

A **hardware algorithm** is a procedure to implement the problem with the available hardware or resource. A flow chart for hardware algorithm translates the word statements

to an information of diagram, that enumerates the sequence of operations along with the necessary condition for their execution.

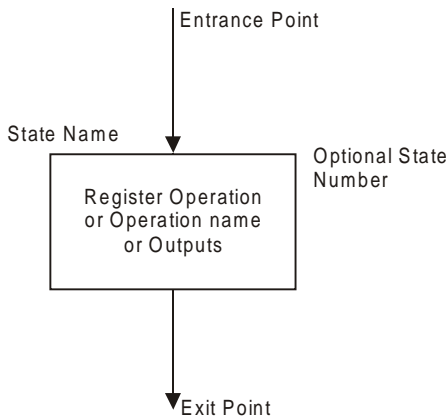
A special flow chart, developed specifically to define the “Digital Hardware Algorithm” is called as an **Algorithmic State Machine** (ASM) chart. In fact, a sequential circuit is alternately called as state machine and forms the basic structure of a digital system. A conventional flow chart describes the sequence of procedural steps and decision paths for an algorithm without concern for their timing relationship. The ASM chart describes the sequence of events as well as the timing relationship between the states of sequential controllers and the events that occur while going from one state to next state. The ASM chart is specifically adapted to accurately specify the control sequence and data processing in digital systems, while considering the constraints of available hardware.

### 9.2 THE ELEMENTS AND STRUCTURE OF THE ASM CHART

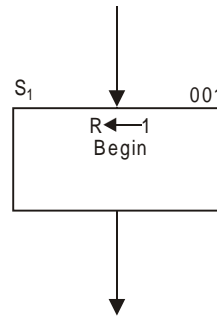
An ASM chart is composed of four elements. These are the “state box”, the “decision box”, the “conditional output box” and “edges”.

#### State Boxes

State boxes describe a state of the ASM. In general an ASM is a sequential system. The state box represents the condition of the system. The symbol for a state box is as follows:



**Fig. 9.1 (a)** State Box



**Fig. 9.1 (b)** Example of State Box

The state box has exactly one entrance point and one exit point. The state box also has a name and is often assigned a number for clarity. Inside the state box we place the names of the system outputs that must be asserted while the system is in that state. We can also place variable assignments in the state box, in order to “remember” the fact that the system has been in that state. This is useful in program design.

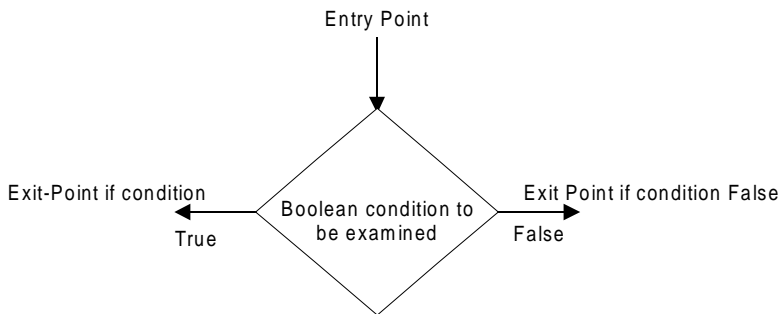
NOTE: Sequential systems are systems with memory; their output depends on their input as well as their history. The historical information that the system stores is called a ‘state’. Combinational systems are the opposite, having no memory. Combinational systems output depends only on present inputs.

#### Decision Boxes

Decision boxes are used to show the examination of a variable and the outcomes of that examination. In this model the outcome of a decision is always either true or false. This



means that there is always exactly one input to a decision box and exactly two exits from the box. The exit points are always labelled “true” or “false” for clarity. When input condition is assigned a binary value, the two exit paths are labelled 1 and 0. 1 in place of ‘True’ and 0 in place of ‘False’. The condition which is being examined is written inside the decision box. The symbol for a decision box is shown here:

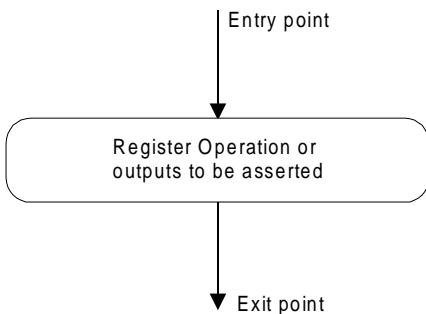


**Fig. 9.2** Decision Box

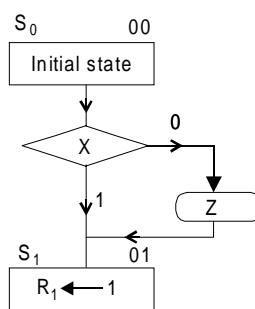
Note that the decision box does not imply a system state. When a decision box is executed the decision is made and the system proceeds immediately to the next item in the ASM chart.

**Conditional Output Boxes**

Conditional output boxes are used to show outputs which are asserted by the system “on the way” from one state to another. The symbol for the conditional output box is shown here in Fig. 9.3(A).



**Fig. 9.3 (a)** Conditional Box



**Fig. 9.3 (b)** Use of Conditional Box.

There is always one entry and one exit point from a conditional output box. Inside the box we write the name of the signal that must be asserted by the system as it passes from one state to another. Input path to conditional box must come from an exit path of decision box.

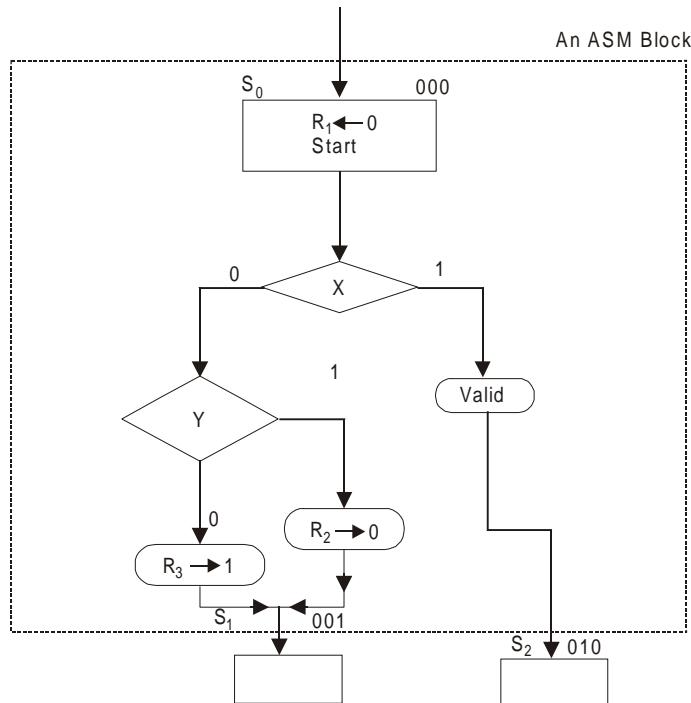
Conditional output boxes do not imply a system state. We can put variable assignments in the state box. Figure 9.3 (b) shows an example using conditional box. The first one in diagram is initial state (Labeled  $S_0$ ) system which attains certain conditions fulfilled before starting the actual process. The control then checks the input X. If  $X = 0$ , then control generates the Z output signal and go to state  $S_1$  otherwise it moves to next state without generating Z.  $R_1 \leftarrow 1$  in state box  $S_1$  is a register operation that loads  $R_1$  by 1.

Edges are used to connect other ASM chart elements together. They indicate the flow of control within the system. The symbol for an edge is as follows:

Note that the edge must always indicate which direction is being taken, by using one or more arrow heads.

### 9.2.1 ASM Block

An ASM block is a structure that contains one state box and all decision boxes and conditional boxes connected to its exit path spanning just before another state box. An ASM block has only one entry path put number of exit paths represented by the structure of decision boxes. Fig. 9.4 shows an ASM block, in ASM chart by dashed lines around it.



**Fig. 9.4** Example of ASM Block-Structure enclosed by dashed line represent an ASM block.

Each ASM block is an ASM chart represents the state of system during one clock pulse. The operations specified by the state box, conditional boxes, and decision boxes are executed during a common clock pulse while the system is in  $S_0$  state. The same clock pulse is also responsible to move the controller to one of the next states,  $S_1$  or  $S_2$  determined by binary status of X and Y. A state box without any decision or conditional boxes constitutes a simple block.

### 9.2.2 Register Operation

A digital system consist of one or more registers for data storage. Thus operation to be performed on data is actually performed on the register that stores the data. A register is a general designation which includes shift registers, counters, storage registers, and single Flip-Flops. A single Flip-flop is identified as 1-bit register. A register is designated by use of one or more capital letters such as A, B, RA, RB,  $R_1$ ,  $R_2$ . In practice most convenient designation is letter R along with numbers such  $R_1$ ,  $R_2$ , ...  $R_n$ .

Register operations can be increment, decrement, shift, rotate, addition, cleaning, copying, data transfer etc. The data transfer to a register from another register or from the result of mathematical operations etc. are shown (or symbolized) by directed arrow whose head is towards target register and tale is towards source register. Fig. 9.5 summarizes the symbolic notations for some of register operations.

<i>Symbolic Notation of Operation</i>	<i>Initial Value of Target Register</i>	<i>Initial Value of Source Register</i>	<i>Value of Target Register After Operation</i>	<i>Description of Operation</i>
$A \leftarrow B$	A = 01010	B = 00000	A = 00000	Copy the content of register B into register A
$R_1 \leftarrow 0$	$R_1 = 11111$	-	$R_1 = 00000$	Clear register $R_1$
$A \leftarrow A + B$	A = 01010	B = 00101	A = 01111	Add the contents of register B to register A and put result in A.
$R \leftarrow R - 1$	R = 01101	-	R = 00100	Decrement register R by 1.
$R \leftarrow R + 1$	R = 00101	-	R = 00110	Increment register R by 1.
"Shift Left A"	A = 10111	-	A = 01110	Shift content of A to left by 1-bit
"Rotate Right A"	A = 10111	-	A = 11011	Rotate content of A to right by 1-bit
$R \leftarrow 1$	R = 01010	-	R = 00001	Set content R to 1

**Fig. 9.5** Symbolic Representation or register operation.

Assume 5-bit register to understand the operations. Note that shift and rotate operation are not same. Shift left means  $MSB \leftarrow MSB-1$ ,  $MSB-1 \leftarrow MSB-2$ , ...,  $LSB + 1 \leftarrow LSB$ ,  $LSB \leftarrow 0$ . If rotate right operation then  $MSB \leftarrow LSB$ ,  $MSB-1 \leftarrow MSB$ , ...  $LSB \leftarrow LSB + 1$ . It is clear that in shift operation loose MSB if left shift or LSB if right shift because as above explained MSB was overwritten by content of MSB-1, and prior to this value of MSB was not saved. And that's why a 0 is inserted at LSB. In rotate operation we don't loose the status of bits. If we rotate left then status of MSB is transferred to LSB and then it is over written by the value of MSB-1.

Equipped with this many knowledge and understanding we are able to draw and understand the simple ASM charts and with analysis and synthesis we can figure out the similarity of ASM charts with that of state diagram.

In the next section (art 9.3) we present some simple examples to give you a feel of ASM chart and its representation.

### 9.2.3 ASM Charts

**Example 9.1. 1-Bit Half Adder:** *The half adder take the two data bits if START input is activated otherwise it remains in initial state. The data bits are read into register  $R_1$  and  $R_2$  and sum and carry bits are maintained in register  $R_3$  and  $R_4$  respectively ASM chart for this task is shown into Fig. 9.6.*

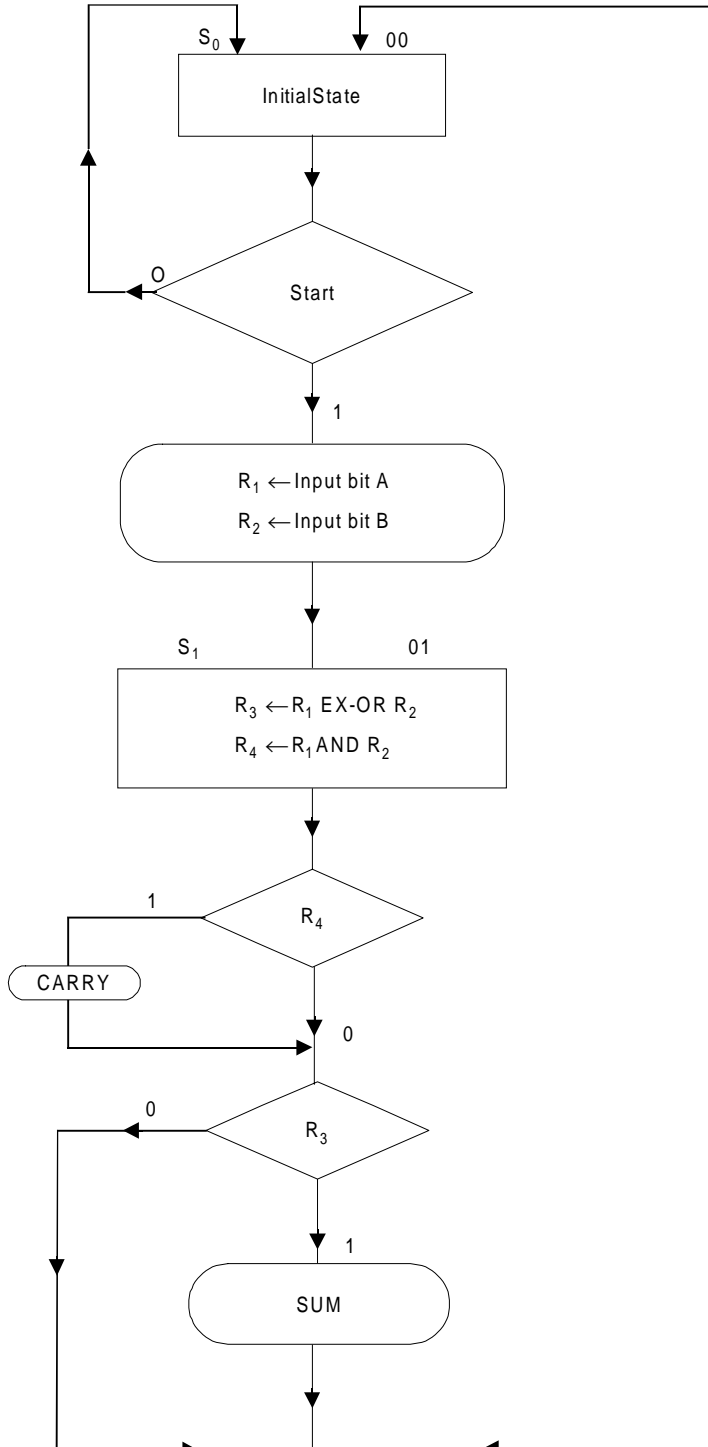


Fig. 9.6 ASM Chart for 1-bit Half Adder.

By observing the ASM chart of Fig. 9.6 we see that there are three state boxes which contributes to three ASM blocks. And we know that the state box and conditional blocks are executed by one common clock pulse corresponding to the state defined by state box in the particular ASM block.

9.2.4 MOD-5 Counter

**Example.** We present counter having one input  $X$  and one output  $Z$ . Counter will have five states, state 0 (i.e.,  $S_0$ ) to state 4 (i.e.,  $S_4$ ) and it moves to next state only and only if input  $X = 1$  at the time of arrival of clock pulse. If  $X = 0$  at this time counter does not moves to next state and maintains its current state. Also when in state  $S_4$  then  $X = 1$  at clock pulse moves the system to next state  $S_0$  i.e., to initial state so that counting can be restarted from 000. The output  $Z$  produces a pulse when  $X = 1$  at 5 clock pulses or when state changes from  $S_4$  to  $S_0$

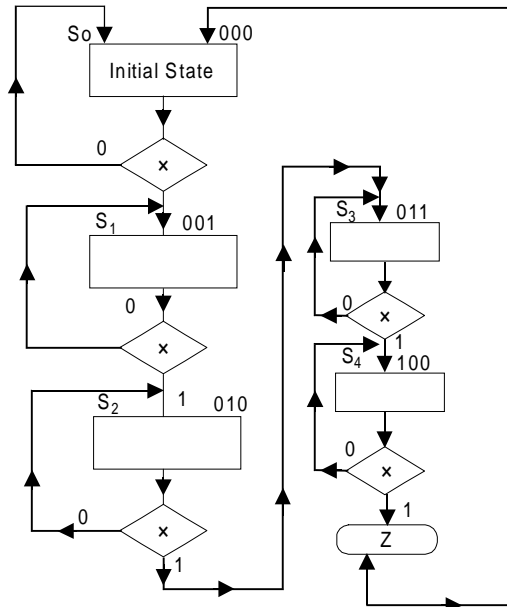


Fig. 9.7 ASM chart for MOD-5 Counter.

Note that in the ASM chart shown in Fig. 9.7 state boxes are blank and does not specify any operation. But the blocks corresponding to these states contains decision boxes which means that only operation to be done in these states are to test the states of input.

Now let us consider the synthesis and we wish to use D flip flop. The 3D-Flip-Flops together are used to assign the 3-bit binary name to states. We know that for D flip-flops excitation input  $D_i$  should be same as next state variable  $Y_i$ . By simply observing the assigned state on ASM chart of Fig. 9.7, we carry out the task.

Let the three outputs of flip-flops are  $Y_2Y_1Y_0$  i.e., the three bit binary name for state.

- (1) First finding the changes in bit  $Y_0$  in ASM chart. When present state is 000 or 010 then the next value of  $Y_0$  has to become 1. Thus

$$D_0 = Y_0 = X[\Sigma(0, 2) + \Sigma\phi(5, 6, 7)]$$

Similarly for  $Y_1$  and  $Y_2$ .

- (2) The next value of  $Y_1$  has to become 1 only for the present states 001 and 010. So

$$D_1 = Y_1 = X[\Sigma(1, 2) + \Sigma\phi(5, 6, 7)]$$

(3) Similarly, next value of  $Y_2$  has to become 1 only for the present state 011. So

$$D_2 = Y_2 = X[\Sigma(3) + \Sigma\phi(5, 6, 7)]$$

(4) Similarly next value of Z has to become 1 only and only for present state 100. So

$$Z = X[\Sigma(4) + \Sigma\phi(5, 6, 7)]$$

Note that state 5, 6, 7 i.e., 101, 110, 111 never occurs and that's why these three states are written  $\Sigma\phi(5, 6, 7)$ .

Thus the synthesis equations can be summarized as

$$D_0 = Y_0 = X.[\Sigma(0, 2) + \Sigma\phi(5, 6, 7)]$$

$$D_1 = Y_1 = X.[\Sigma(1, 2) + \Sigma\phi(5, 6, 7)]$$

$$D_2 = Y_2 = X.[\Sigma(3) + \Sigma\phi(5, 6, 7)]$$

and

$$Z = X.[\Sigma(4) + \Sigma\phi(5, 6, 7)]$$

Here input X is ANDed with all the expressions for the excitations. In this system input X is used to enable the counter. Thus excitation equations can be given as-

$$D_0 = Y_0 = X \bar{Y}_2 \bar{Y}_1 \bar{Y}_0 + X \bar{Y}_2 Y_1 \bar{Y}_0$$

$$D_1 = Y_1 = X \bar{Y}_2 \bar{Y}_1 Y_0 + \bar{Y}_2 Y_1 \bar{Y}_0 X$$

$$D_2 = Y_2 = X \bar{Y}_2 Y_1 Y_0$$

and

$$Z = X Y_2 \bar{Y}_1 \bar{Y}_0$$

Where state  $S_0$  is represented by  $\bar{Y}_2 \bar{Y}_1 \bar{Y}_0$  as its name assigned was 000. In fact if value of state value is 0 then it is represented by  $\bar{Y}_i$  and if it is 1 then use  $Y_i$ .

Similarly the states are represented for  $S_1$  to  $S_4$ .

### 9.3.5 Sequence Detector

**Example.** We now consider a sequence detector to detect "0101" and allowing the overlapping. This example is chosen to illustrate the similarity between state diagram. If we have already drawn a state diagram then drawing the ASM chart is a very easy job. The state diagram to detect 0101 is shown in Fig. 9.8.

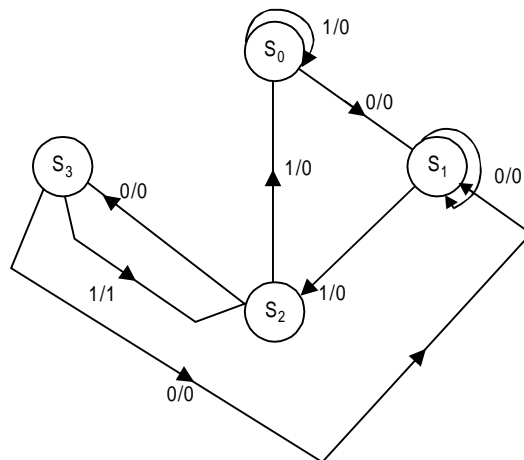
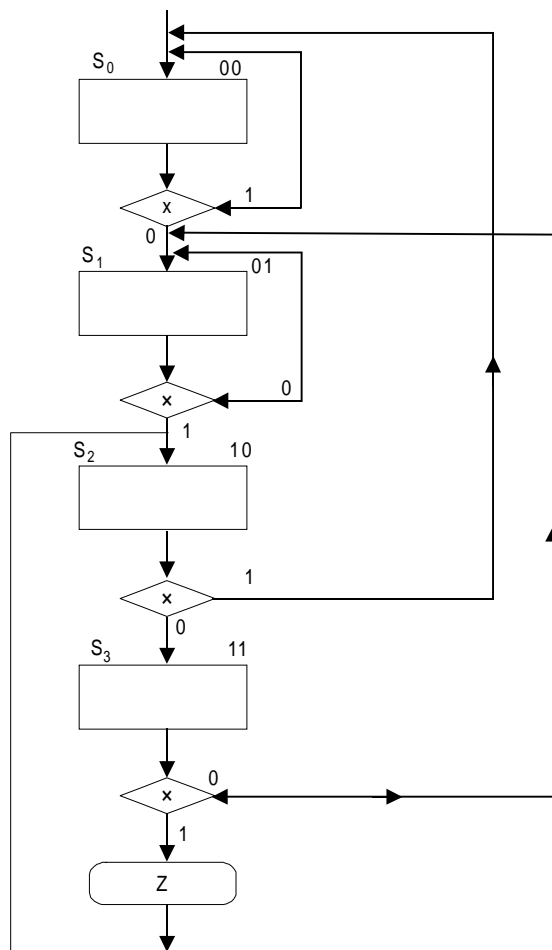


Fig. 9.8 State Diagram of 0101 sequence Detector with overlapping.

Number Marked with segments connecting two states are value of input and output and written as Input/output. If its 1/0 means Input-1 then Output is 0, while in this state.

It is evident from the state diagram that there are four states  $S_0, S_1, S_2, S_3$ . So 2-bit binary code can be associated to these states to identify the particular state. Thus we use two D flip-flops to assign binary number  $Y_1Y_0$  to the states. As earlier indicated use of D Flip Flop makes the excitation table same as transition table because for D flip-flop  $D_i = Y_i$ . In fact we carry out these exercise after drawing ASM chart, but here we did it earlier to reflect the similarity between ASM chart and state graph as major difference between the two is indication of timing relation in the drawing. Below is the ASM chart i.e., Fig. 9.9 for the problem.



**Fig. 9.9** ASM chart for 0101 Sequence Detector.

Note that, as in earlier examples, here also X is input through which the sequence is applied to the system. Z is output which goes high when the intended sequence is detected. Note the similarity between the state diagram and ASM chart. A close inspection of two graphs, shows that for every state of state diagram, there exist one ASM blocks and there is one state box per ASM block in the ASM chart. Thus there are four ASM blocks in Fig. 9.9 each of which contains a decision box and last one contains a conditional box also in

addition to state box. We again assert the fact that all the operations owing to an ASM block are to be completed in the same clock period. We now consider synthesis to find out the equations for excitations.

Here also we use observation (as was done in example 2) to determine that when next state variables  $Y_0$  and  $Y_1$  become 1. Thus

$$D_0 = Y_0 = \bar{X}[\Sigma(0, 2)]$$

$$\therefore D_0 = Y_0 = \bar{X}\bar{Y}_1\bar{Y}_0 + \bar{X}Y_1\bar{Y}_0$$

as the  $S_0$  state = 00 =  $\bar{Y}_1\bar{Y}_0$

$$S_2 \text{ state} = 010 = Y_1\bar{Y}_0$$

If input  $X = 0$  make next state to comes then input =  $\bar{X}$  and if input  $X = 1$  causes the next state then input =  $X$ .

In equation for  $D_0$ ,  $\bar{X}\bar{Y}_1\bar{Y}_0$  shows that next state variable  $Y_0 = 1$  when  $X = 0$  and present state is  $S_0$  (i.e., 00). Similarly  $\bar{X}Y_1\bar{Y}_0$  means next state variable  $Y_0 = 1$  if the input  $X = 0$  while in state  $S_2$  (i.e., 10). See the ASM chart to verify the statements.

Similarly  $D_1 = Y_1 = X\bar{Y}_1Y_0 + \bar{X}Y_1\bar{Y}_0$

and  $Z = XY_1Y_0$

### 9.3 TIMING CONSIDERATIONS

The timing of all the registers and flip-flops is controlled by a master clock generator. The clock pulses are equally applied to the elements (i.e., registers, flip-flops) of both data processing and control subsystems. The input signals are synchronized with the clock as normally they happen to be the output of some other circuit utilizing the same clock. Thus the inputs change the state during an edge transition of clock. In the similar way, the outputs, that are a function of present state and synchronous inputs, will also be synchronous.

We re-insert that major difference between a conventional flow chart and ASM chart is in defining and interpreting the timing relation among the various operations. Let us consider the ASM chart shown in Fig. 9.4. If it would be a conventional flow chart, then the listed operations within the state, decision and conditional boxes are executed sequentially i.e., one after another in time sequence. Alternately saying, at one clock pulse only one of the boxes will be executed, where the box may be a state box or a decision box or a conditional box. Thus a total denial of timing relation among the various activities. In contrast to it, an entire ASM block is treated as one unit. All the activities specified within the block must happen in synchronism with the transition of positive edge of the clock, while the system changes from current state to next state. Here it is assumed that all the flip-flops are positive edge triggered. For illustration purpose consider the ASM chart shown in Fig. 9.4 and Fig. 9.10 shows the transition of control logic between the states.

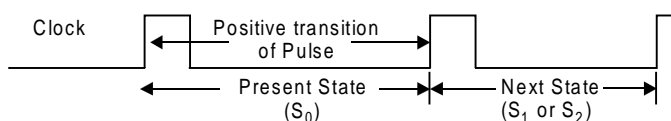


Fig. 9.10 Transition between States.



In order to understand the state transition at the positive edge of clock refer the Fig. 9.4 and 9.10 simultaneously along with the following discussion.

The arrival of first positive transition of clock, transfers the control subsystem into  $S_0$  state. The activities listed in various boxes of ASM block, corresponding to  $S_0$  state can now be executed, as soon as the positive edge of second clock pulse arrives. At the same time depending upon values of inputs  $X$  and  $Y$  the control is transferred to next state which may be either state  $S_1$  or State 2. Referring to the ASM block indicated by dashed line in Fig. 9.4 we can list out operation that occur simultaneously when the positive edge of second clock pulse appears. They are–

Recall that system is  $S_0$  state before second clock pulse

- (1) Register  $R_1$  is cleared.
- (2) If input  $X$  is 1, the output signal VALID is generated and the control enters in  $S_2$  state.
- (3) If input  $X$  is 0, then the control tests the input  $Y$ .
- (4) If input  $Y$  is 0 register  $R_3$  is set to one. If input  $Y$  is 1 register  $R_2$  is cleared. In either the case next state will be  $S_1$  state.

Observe the ASM chart closely (in Fig. 9.4), and we find that next state is decided by the status of input  $X$  only. If  $X = 1$  then next is  $S_2$  state and when  $X = 0$  then whether input  $Y = 0$  or 1 the next state will always be  $S_1$ . Also note that the operation in the data processing subsect and change in state of control subsystem occur at the same time, during the positive transition of same clock pulse. We now consider a design example to demonstrate timing relation between the components of ASM chart.

**Example 9.4.** *Design a digital system having one 4-bit binary counter 'C' whose internal bits are labelled  $C_4 C_3 C_2 C_1$  with  $C_4$  MSB and  $C_1$  as LSB. It has two flip-flops named 'X' and 'Y'. A start signal incrementing the counter 'C' by 1 on arrival of next clock pulse and continues to increment until the operation stops. Given that the counter bits  $C_3$  and  $C_4$  determines the sequence of operation. The system must satisfy following–*

- (1) *Initiate the operation when start signals = 1 by clearing counter 'C' and flip-flop "Y", i.e.,  $C = 0000$  and  $Y = 0$ .*
- (2) *If counter bit  $C_3 = 0$ , it causes E to cleared to 0 i.e  $E = 0$  and the operation proceeds.*
- (3) *If counter bit  $C_3 = 1$ , E is set to 1 i.e.  $E = 1$  and*
  - (a) *if  $C_4 = 0$ , count proceeds.*
  - (b) *if  $C_4 = 1$ , F is set to 1 i.e.  $F = 1$  on next clock pulse and system stops counting.*

**Solution.** ASM chart for the given problem is shown in Fig. 9.11. A close inspection reveals that

When, no operation, system is in initial state  $S_0$ , and keep waiting for start signals 'S'.

When  $S = 1$ , counter  $C = 0000$  and  $Y = 0$  and simultaneously control goes to  $S_1$  state. It means clearing of counter 'C' and flip-flop 'Y' occurs during  $S_0$  state.

The counter is incremented by 1 during state  $S_1$ , on the arrival of every clock pulse. During each clock pulse simultaneously with increment during same transition of clock, one of the three possibility is tested to determine the next state:

- (1) Either  $X$  is cleared and control stays at  $S_1$  ( $C_3 = 0$ ).

or

(2) X is set ( $X = 1$ ) and control maintains  $S_1$  state ( $A_4 A_3 = 10$ ).

or

(3) X is set and control advanced to state  $S_2$  ( $A_4, A_3 = 11$ ).

When in  $S_2$  state flip-flop 'Y' is set to 1 and control move back to its initial state  $S_0$ . The ASM chart consist of three blocks, one external input S, and two status inputs  $S_4$  and  $S_3$ .

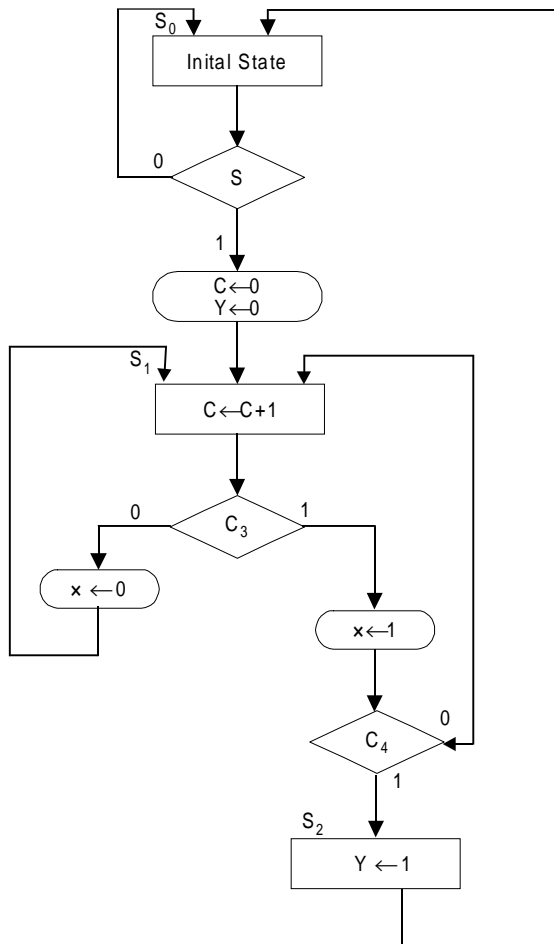


Fig. 9.11 ASM chart for example 9.4.

**Example 9.5.** Design a digital system for weight computation in a given binary word.

**Solution.** The weight of a binary number is defined as the number of 1's contained in binary representation. To solve the problem let the digital system have

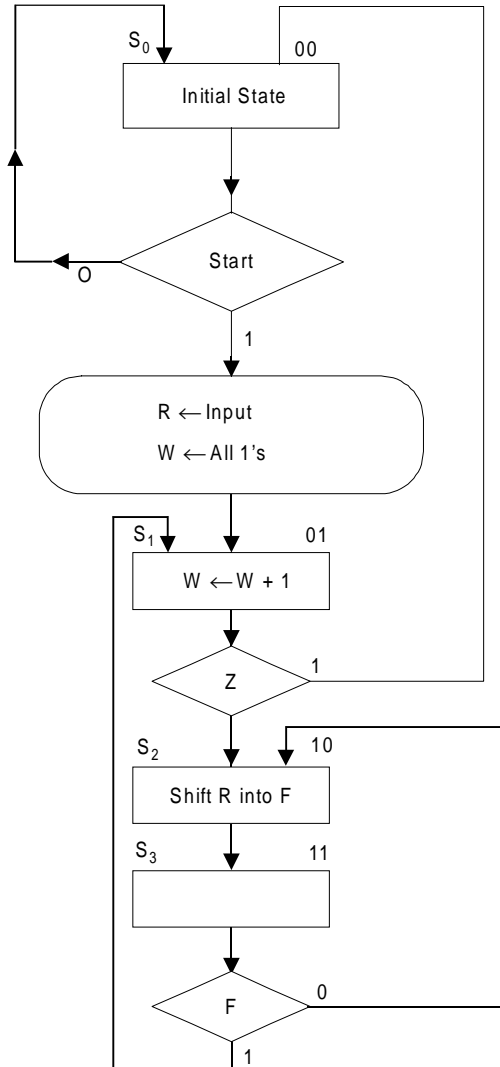
1. R – A register where binary work is stored.
2. W – A register that counts number of 1's in binary number stored in R.
3. F – A flip-flop.

The operation of the system is to shift a single bit of R into F. Then check the output of the F. If it is 1 increment count in W by 1. If it is 0 no increment in W. The moment all the bits are shifted and tested operation stops and W contains the weight of the given word.

The ASM chart for this problem is shown in Fig. 9.12. Note that the system have 3 inputs S, Z and F. Z is used to sense weather all the bits in register R are 0 or not. Z = 1 indicates that register R contains all zeros, and the operation must stop.

Initially machine is in state  $S_0$  and remains is state  $S_0$  until the switch S (*i.e.*, start signal) is made 1. If  $S_0 = 1$  then in  $S_0$  state, the clock pulse causes. Input word to be loaded into R, counter W to have all 1's and machine to transferred to state  $S_1$ .

In state  $S_1$  a clock pulse causes two works simultaneously. First it increments W by 1. If W is incremented for the first time, then the count in W becomes all 0's as initially it was all 1's second it tests Z. If Z = 0 machine goes to state  $S_2$ . If Z = 1 machine goes to state  $S_0$  and count in W is weight of the binary word.



**Fig. 9.12** ASM chart for weight computation.

In state  $S_2$  a bit of register R is shifted into flip-flop F, and machine goes to state  $S_3$ .

In state  $S_3$ , shifted bit in F is tested. If  $f = 0$  the machine goes to state  $S_2$  to shift next bit into F. If  $f = 1$  it goes to state  $S_1$  to increment the count in W.

#### 9.4 DATA PROCESSING UNIT

Once the ASM chart is prepared, the system (or machine) can be designed. The design is splitted in two parts:

- (a) Data Processing Unit
- (b) Control Unit.

The data processing unit contains the element that performs the operations like increment the count, shift a bit etc.

The central unit is the subsystem that is responsible to move the machine from one state to another, according to the conditions specified by ASM chart.

In this section we are concerned with the design of data processing unit only.

**Example 9.6.** *Design data processing unit for binary weight computation, discussed in Examples 9.5.*

**Solution:** Proposed data processing unit is shown in Fig. 9.13. The control sub-system has 3 inputs S, Z, F as discussed in Example 9.5. It has four control signals  $C_0, C_1, C_2, C_3$  corresponding to states  $S_0, S_1, S_2, S_3$  respectively (refer to ASM chart shown in Fig. 9.12). We advise the readers to go through Example 9.5 again.

Next shown in figure is a shift register R. Serial input '0' is a data input. Each time data in R is shifted left this input inserts a 0 at LSB. A HIGH on SHIFT LEFT input shifts the data present in R to left by 1 bit and loads a serial 0 at LSB. A HIGH on LOAD INPUT DATA loads the INPUT DATA into R. This is the binary word whose weight is to be calculated. This word is loaded as parallel data into R.

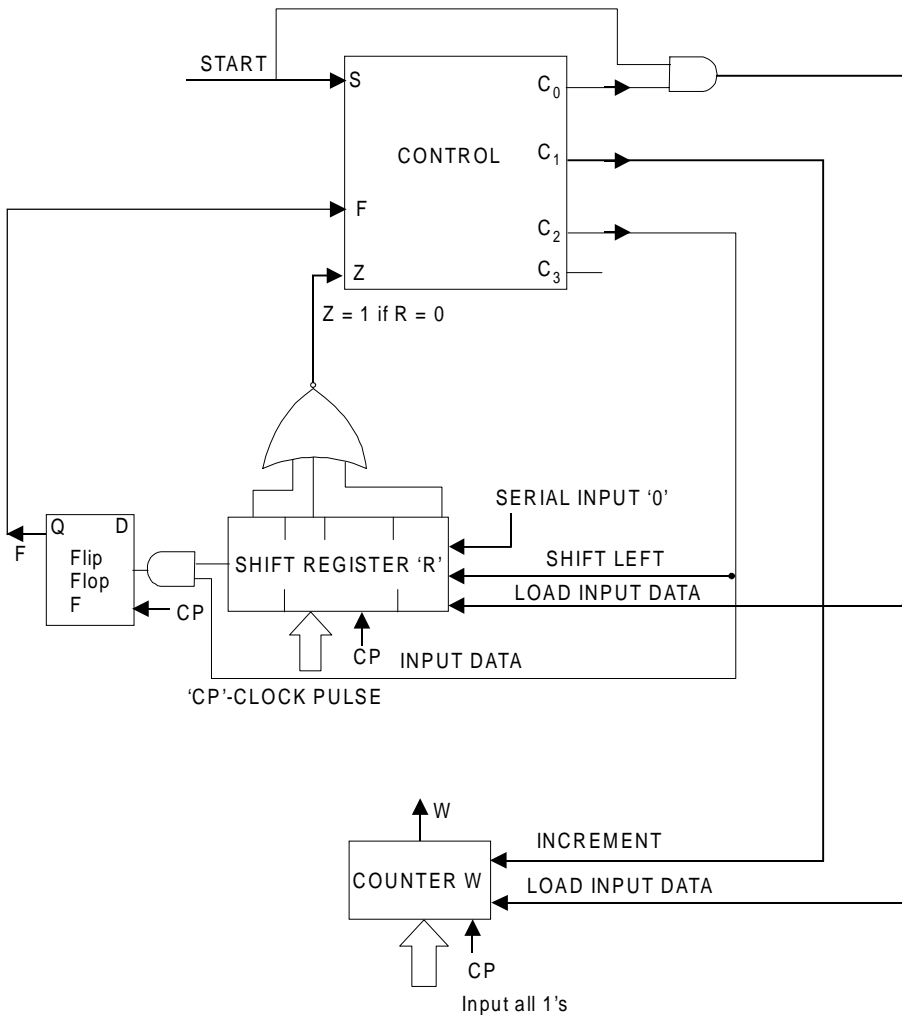
A NOR gate is used to determine whether all the bits of R are 0 or not. All the bits of R are brought to the input of NOR. As soon as all the bits of R become 0 output of NOR goes HIGH. If any of the bit of R is 1 output of NOR remains LOW. Output of NOR is feeded as input Z to controls, where it is checked for 0 or 1.

A flip-flop F is connected a MSB of R. This flip-flop is used to collect each shifted bit from register R. Every time R receives shift left command, its MSB is shifted out and is received in flip-flop F. The output of flip-flop is feeded to controls as input F, where it is checked for 1 or 0.

Last in figure is counter W which is yet another register acting as counter. A HIGH on LOAD INPUT loads all 1's into W. A HIGH INCREMENT increments the count in W by 1.

Initially the system is in state  $S_0$  (refer ASM chart of Fig. 9.12 along with Fig. 9.13). As soon as  $START = 1$ ,  $C_0$  is activated. This causes LOAD INPUT DATA signals of both R and W to go HIGH. Thus binary word is loaded into R and initial count is loaded into W. At the same time the machine moves to state  $S_1$ .

In state  $S_1$  signal  $C_1$  is activated. This causes INCREMENT signal of W to go HIGH and consequently the count in W is incremented by 1. When it is incremented for the first time. All 1's become all 0's. At the same time input Z is tested by controls. If  $Z = 1$  control goes back to state  $S_0$ . If  $Z = 0$  control goes to state  $S_2$ .



**Fig. 9.13** Data processing unit for Binary weight computation.

In state  $S_2$  the signal  $C_2$  is activated. The  $C_2$  causes SHIFT LEFT of R to go HIGH and enables the flip-flop F. Thus the content of R shifted to left by 1-bit. Hence MSB of R is shifted out and is collected by F and at the same time a 0 is inserted at the LSB through serial input. Now the machine moves on to state  $S_3$ .

In state  $S_3$  the output of F is tested by control subsystem. If  $F = 1$  machine should go to state  $S_1$  i.e.,  $C_1$  to be activated next. If  $F = 0$  machine should go to state  $S_2$  i.e.,  $C_2$  to be next. Since all these activities are internal to control subsystem,  $C_3$  is not connected to any element of data processing unit. In fact  $C_3$  is used to activate the signals  $C_1$  or  $C_2$  and is processed internally by control unit.

### 9.5 CONTROL DESIGN

As earlier stated the job of the control subsystem is to move the machine from one state to other state according to inputs given to it. In general variables defined by the “decision boxes” in an ASM chart are treated as inputs to the control subsystem.

There are many methods to obtain a control subsystem according to the ASM charts. Here we consider only two methods:

- (i) Multiplexer Controls
- (ii) PLA controls.

### 9.5.1 Multiplexer Control

In this approach we use the multiplexers to realize the control subsystem. The number of multiplexers depends upon the number of states in ASM chart. For example if there are 4-states then we need 2-bit binary number to specify these states uniquely. So we take two multiplexers, one for each bit of representation. In general, if ' $n$ ' is number of multiplexers then  $2^n \geq$  No. of states.

The type of multiplexer also depends upon the number of states. If there are four states in ASM chart then the multiplexer should be a  $4 \times 1$  multiplexer. Alternately

$$\text{Number of MUX inputs} \geq \text{No. of states}$$

In general design the output of multiplexers denotes the PRESENT STATE variable as these outputs reflect current status of control unit. The inputs to multiplexer represents the NEXT STATE variable. It is because if these inputs are changed output of multiplexers may change and thus we say that the state is changed.

To being with we consider our example of binary weight computation illustrated in Examples 9.5 and 9.6. We urge the readers to go through these to examples carefully before proceeding further.

**Example 9.7.** *Design the control system for binary weight computation, by using multiplexers.*

**Solution.** The ASM chart for binary weight computation is drawn in Fig. 9.12. Referring to the chart we find that there are 4 states. So,

$$2^n \geq 4$$

or

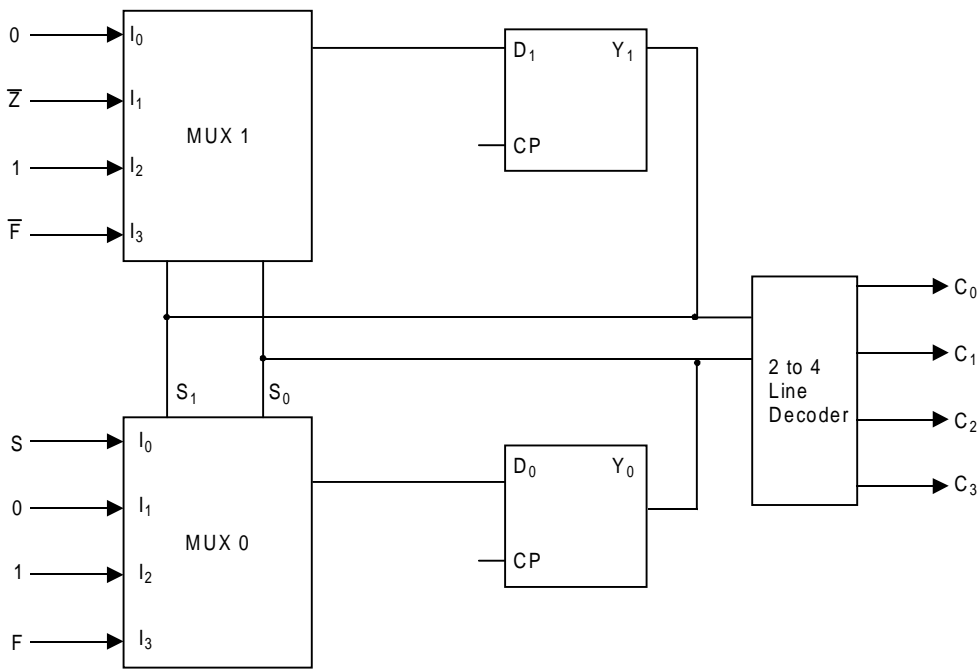
$$n \geq 2$$

So we take 2 multiplexers ( $n = 2$ ) MUX 1 and MUX 0. Since there are 4 states we select 4 input multiplexers *i.e.*,  $4 \times 1$  multiplexers.

After selecting the multiplexers next step is to draw state table, as shown in Fig. 9.14(a). The first 3 columns of the tables shows present states, next state and the inputs that causes the next state. Last column of the table is multiplexer input. As earlier stated multiplexer inputs are next state variables. Thus entries in this columns are made by making observations on inputs and next state. For example, if present state is  $S_0$  *i.e.*, multiplexer output  $Y_1 = 0$  and  $Y_0 = 0$ , then status of switch S decides the next state. If  $S = 0$  the next state is  $S_0$  *i.e.*,  $Y_1 = 0$  and  $Y_0 = 0$ . If  $S = 1$  the next state is  $S_1$  *i.e.*,  $Y_1 = 0$  and  $Y_0 = 1$ . Hence when  $S = 0$ ,  $Y_0 = 0$  and when  $S = 1$ ,  $Y_0 = 1$ ,  $Y_0 = 0$  so we say  $Y_0 = S$ . Since  $Y_1 = 0$  always the first entry in MUX inputs column is 0 S. Consequently input  $I_0$  of MUX 1 must be connected to 0 and input  $I_0$  of MUX 0 must be connected to S. The same is shown in Fig. 9.14(b). Readers are advised to verify all the rows of state table in similar way.

Present State		Inputs			Next State		MUX Inputs	
$Y_1$	$Y_0$	$S$	$Z$	$F$	$Y_1$	$Y_0$	$D_1 = Y_1$ MUX 1	$D_0 = Y_0$ MUX 0
$S_0$	0 0	0	×	×	0	0	0	S
		1	×	×	0	1		
$S_1$	0 1	×	1	×	0	0	$\bar{Z}$	0
		×	0	×	1	0		
$S_2$	1 0	×	×	×	1	1	1	1
$S_3$	1 1	×	×	0	1	0	$\bar{F}$	F
		×	×	1	0	1		

(a) State Table



(b) Logic Diagram

Fig. 9.14 Control Subsystem for binary Weight computation.

Fig. 9.14 (b) shows the complete control design for weight computation. The outputs of multiplexers are fed to a 0 flip-flops, whose outputs  $Y_1$  and  $Y_0$  are brought back to select lines  $S_0$  and  $S_1$  of multiplexers.  $Y_1$  and  $Y_0$  are decoded further by using 2 to 4 line decoder to generate the control signals  $C_0, C_1, C_2, C_3$  corresponding to states  $S_0, S_1, S_2, S_3$  respectively.

To understand the operation let us consider that control is in state  $S_0$  so  $Y_1 = 0$  and  $Y_0 = 0$  i.e.,  $S_1 = S_0 = 0$ . Since  $S_1, S_0 = 0$ , input  $I_0$  of both the multiplexers are selected. As long as  $S = 0$ , both  $Y_1 = Y_0 = 0$  and machine remains in state  $S_0$ . As soon as  $S = 1$  output

of MUX becomes 1 and consequently  $Y_1 = 0$  and  $S_0 = 1$ . Thus signal  $C_1$  is activated and select inputs become  $S_1 = 0$  and  $S_0 = 1$ . Hence inputs  $I_1$  of both multiplexers selected. Note that by activation of  $C_1$  state  $S_1$  has arrived. At the input  $I_1$  of MUX 1,  $\bar{Z}$  is connected whose value is responsible to make  $Y_1 = 0$ ,  $Y_0 = 0$  or  $Y_1 = 1$ ,  $Y_0 = 0$ . Thus input  $Z$  is tested in state 1, which was to be done in  $S_1$  according to the ASM chart shown in Fig. 9.12. Like wise the complete operation can be verified.

### 9.5.2 PLA Control

Use of PLA to realize, control subsystem makes the system more compact and efficient. PLAs have internal AND-OR array *i.e.*, the outputs of PLA represent sum of product. Thus, overall strategy is to prepare on SOP equation for each bit of state representation. For example, if 4-states are there we need two bits of representation. Thus we need two SOP equations. After getting the SOP equations next step is to prepare PLA program table. Such a table is a input-output table according to which PLAs are programmed.

**Example 9.8.** Design a control system for binary weight computation, by using the PLA.

**Solution.** We advise the readers to go through the ASM chart given in Fig. 9.12 and multiplexer control shown in Fig. 9.14.

We now obtain two SOP equation for next state variables  $Y_1$  and  $Y_0$  according to state table given in Fig. 9.14(a). Let  $C_0$ ,  $C_1$ ,  $C_2$ ,  $C_3$  are signals corresponding to states  $S_0$ ,  $S_1$ ,  $S_2$ ,  $S_3$

$$Y_1 = Y_1\bar{Y}_0\bar{Z} + Y_1Y_0 + Y_1Y_0\bar{F}$$

or

$$Y_1 = C_1\bar{Z} + C_2 + C_3\bar{F}$$

as ( $Y_1 Y_0 = 0 1$  means  $C_1$  and  $Y_1 Y_0 = 1 1$  means  $C_3$ )

Similarly

$$\begin{aligned} Y_0 &= \bar{Y}_1\bar{Y}_0S + Y_1\bar{Y}_0 + Y_1Y_0F \\ &= C_0S + C_2 + C_3F \end{aligned}$$

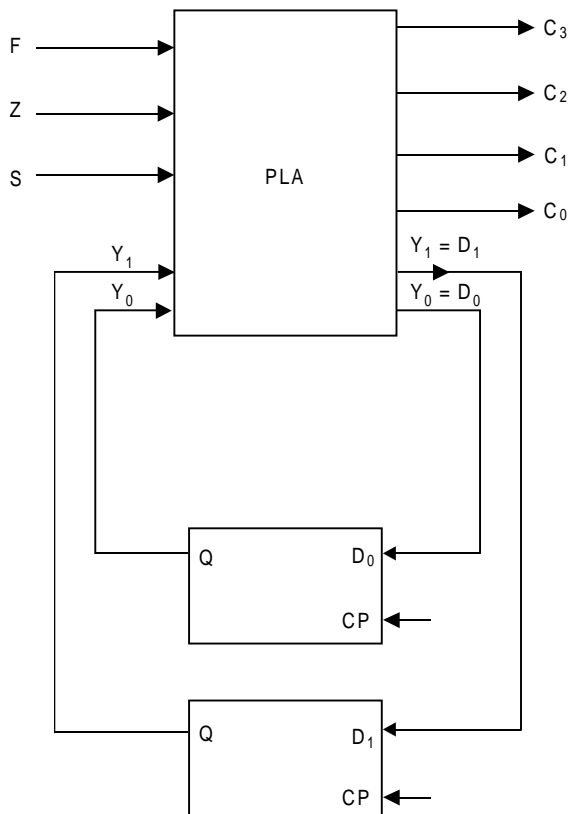
The PLA program table and PLA control block is shown in Fig. 9.15. Let us examine the PLA program table. Note that  $Y_1Y_0$  in the input side of table represents the present state and  $Y_1Y_0$  in the output side represents the next state. Further all entries at the input side is made for product terms and at the output side entries are results of sum of products.

First four rows in the program table are simply showing the values of  $Y_1Y_0$  and corresponding state to be excited. For example if present state is  $Y_1 = 0$  and  $Y_0 = 0$ , then if it is state  $S_0$  and signal  $C_0$  is activated. This is shown in first row. At the output side  $Y_1Y_0$  shows next state. Now observe the third row, which shows that machine is in state  $S_2$  so  $C_2$  is activated. But according to the ASM chart shown in Fig. 9.12, if the machine is in state  $S_2$  it goes to state  $S_3$  without testing any input. Hence at the output side of table we marked  $Y_1 = 1$  and  $Y_0 = 1$ . Note that  $Y_1$  and  $Y_0$  on the output side are filled up according to the two SOP equations obtained in the beginning. In fact the first four rows are used to show what will be the control signal to be activated when machine is in a state.



Product Terms	Inputs					Outputs					
	$Y_1$	$Y_0$	$S$	$Z$	$F$	$Y_1$	$Y_0$	$C_0$	$C_1$	$C_2$	$C_3$
$C_0 = \bar{Y}_1 \bar{Y}_0$	0	0						1	0	0	0
$C_1 = \bar{Y}_1 Y_0$	0	1						0	1	0	0
$C_2 = Y_1 Y_0$	1	0				1	1	0	0	1	0
$C_3 = Y_1 Y_0$	1	1						0	0	0	1
$C_1 \bar{Z} = \bar{Y}_1 Y_0 \bar{Z}$	0	1		0		1	0	0	1	0	0
$C_3 \bar{F} = Y_1 Y_0 \bar{F}$	1	1			0	1	0	0	0	0	1
$C_0 S = \bar{Y}_1 \bar{Y}_0 S$	0	0	1			0	1	1	0	0	0
$C_3 F = Y_1 Y_0 F$	1	1			1	0	1	0	0	0	1

(a) PLA Program Table



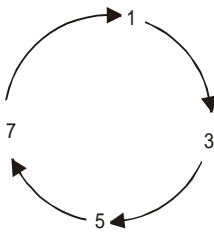
(b) Logic Diagram

Fig. 9.15 Control subsystem for weight computation using PLA.

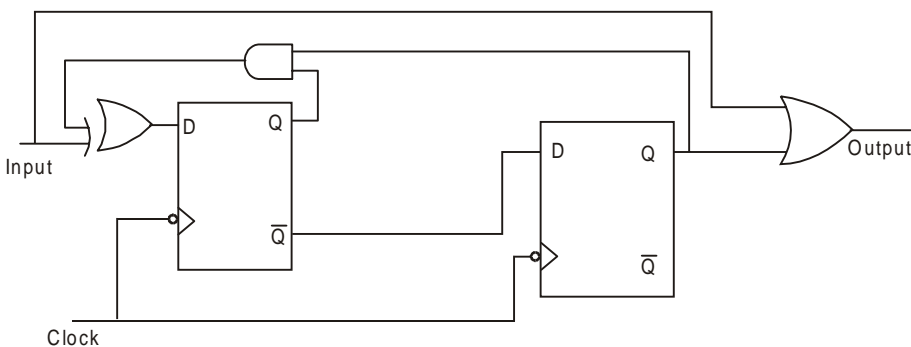
The rest of the four rows in PLA program table shows the input to be tested when machine is in a state and what should be the next state if testing is true. Consider the 7th row having product terms entry  $C_0S = \bar{Y}_1\bar{Y}_0S$ . This tests the input S when machine is in state  $S_0$ . At the input side  $Y_1 = Y_0 = 0$  to show state  $S_0$  and entry  $S = 1$  is status of input S. At the output side in this row  $C_0 = 1$  as machine is in state  $S_0$ . Next  $Y_1 = 0$  and  $Y_0 = 1$  at the output side indicates that since input  $S = 1$ , the machine must go to state  $S_1$  at next clock pulse.

9.6 EXERCISES

1. Draw the ASM chart for a binary multiplier.
2. A binary stream is arriving serially. Stream is such that LSB arrives first and MSB arrives last system requirement is such that the system must output the 2's complement of each incoming bit serially. Draw the ASM chart and design control subsystem and data processing subsystem for this system.
3. Draw the ASM chart to compare two 4-bit binary datas.
4. Draw the ASM chart for 1-bit full adder.
5. Draw the ASM chart for 2-bit binary counter having one enable input.
6. Design a synchronous state machine to generate following sequence of states.



7. Draw the ASM chart and state diagram for the circuit shown.



8. Draw the ASM chart and state diagram for decade counter.
9. Draw the ASM chart and state diagram to convert two digit hexadecimal number into packed BCD number.
10. Draw the ASM chart and state diagram for 1 bit full subtractor.

## SWITCHING ELEMENTS AND IMPLEMENTATION OF LOGIC GATES

---

### 10.0 INTRODUCTION

In earlier chapters we have studied the basic logic gates and seen how they can be realized using switches in ON/OFF fashion (or TRUE/FALSE). The semiconductor devices can be used to replace these switches and can realize these logic functions. A circuit employing semiconductor devices to realized logic functions is called as **digital circuit**. Since semiconductor devices are used to replace the switch, these devices and their switching characteristics are discussed first in this chapter. Infact using semiconductor devices offers several advantages, that will be apparent throughout this chapter.

Advent of semiconductor IC technology in late 1950's and early 1960's made it possible to fabriate large number of circuit components on a small piece of semiconductor, whose area is few mm<sup>2</sup>. Thus a large number of circuits could be integrated on small piece of semiconductor. This small piece of semiconductor is called as CHIP and a chip with circuits fabricated on it is chritened as **Integrated Circuit or IC** in short. Various types of digital circuits are available in form of ICs. In particular, advent of MOS technology made more number of components and logic circuits on one IC. Thus giving rise to very large scale integration which resulted in high capacity memory devices, microprocessor and many other complex circuits available on a small chip.

### 10.1 FUNDAMENTALS OF SEMICONDUCTORS AND SEMICONDUCTOR SWITCHING DEVICES

#### 10.1.1 Semiconductors

By the idea of electrical properties of solids, we know that materials are broadly categorized as conductor, semi conductor, and insulator, on the basis of their conductivity. SEMI CONDUCTORS are those which have electrical conductivity intermediate to that of an conductor and insulator. In fact in a semiconductor the filled energy band, called valance band and the unfilled energy band, called conduction band are separated by a small energy gap, called band gap energy  $E_G$ .

By providing energy from an external source, the charge carriers at filled band can be raised to conduction band. Thus the materials can start conduction.

There are numerous elemental and compound semiconductors are available out of which silicon ad Germanium are two common elemental semiconductors. The silicon is used mostly because energy gap of silicon is a bit larger than the germanium thus providing a good operational mechanism, as we will see later.

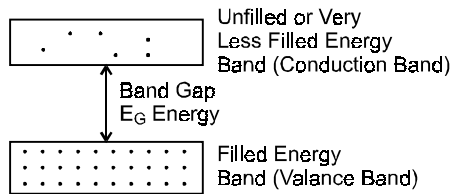


Fig. 10.1 Energy Band Diagram for semiconductors

Both the silicon and germanium atoms have 4 electrons in their outer shell, which forms covalent bands with neighbouring atoms in a semiconductor material. This is shown in Fig. 10.2. By providing energy from an electrical source these covalent bands can be broken and free charge carriers are available for conduction. Many of these covalents bonds are broken even at room temperature so at room temperature semiconductors (shorthand SCs) can behave some what like conductors. A semiconductor in its extreme pure form is called as **intrinsic semiconductor**. At room temperature conductivity of intrinsic semiconductors are poor. Thus to increase the no. of free charge carriers (or conductivity) it usual to add some impurity to intrinsic semiconductors. This dramatically increases the conductivity of semiconductors, and make it suitable for various applications. A semiconductor with impurity added to it is called as **extrinsic semiconductor**. The process of adding impurities to semiconductors is called **doping**. These extrinsic semiconductors are also called as **doped semiconductors**.

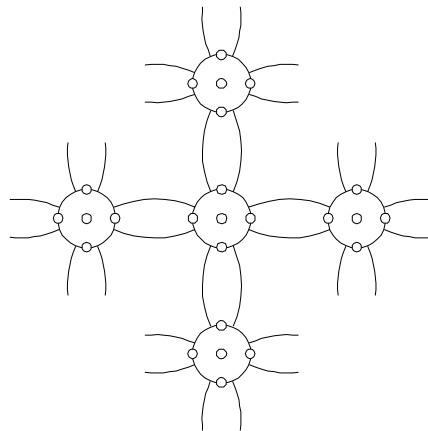


Fig. 10.2 Formation of Covalent Bonds in Semiconductor.

In a semiconductor whenever a covalent bond is broken the electron gets free from its parent atom and leaves a vacancy, thus a +ve charge equal to the magnitude of electron charge is left with the parent atom. Thus the atom is ionized. An electron from neighbouring atom can fill this vacancy, but creates a vacancy there, if this process repeats this vacancy can be moved from one place to another and thus giving a mechanism of conduction. This vacancy is called **hole** which has an equal but opposite charge to that of an electron. In a semiconductor conductor both holes and electrons provide the conduction mechanism. Infact free electrons travels in conduction band where hole movement is due to the movement of electrons in valence band. Both the charge carriers travel in a direction opposite to each other but constitute current in same direction.

In a semiconductor both electrons and holes moves randomly through the semiconductor crystal during which an electron can fill a hole causing free electrons and holes to disappear.

This process is called **Recombination**. In a semiconductor the rate of charge carrier generation (generation of both electrons and holes) is equal to the rate of recombination and remains constant in time.

In an intrinsic semiconductor the number of free electron ( $n$ ) and free holes ( $p$ ) is same *i.e.*,

$$n = p = n_i \quad \dots(10.1)$$

where  $n_i$  is intrinsic concentration of holes or electrons in intrinsic semiconductor. The  $n_i$  can be approximated as

$$n_i^2 = BT^3 e^{-E_G/K_B T} \text{ per cm}^3 \quad \dots(10.2)$$

where  $B$  = material constant =  $5.4 \times 10^{31}$  for silicon

$T$  = Temperature in Kelvin

$E_G$  = Band gap energy of SC = 1.12 eV for silicon

$K_B$  = Boltzman constant =  $1.38 \times 10^{-23}$  joules/kelvin  
=  $8.62 \times 10^{-5}$  eV/K

By using eqn (10.2) for silicon we get that at room temperature *i.e.*, at 27°C we get approximately  $1.51 \times 10^{10}$  free carrier/cm<sup>3</sup>. A silicon crystal has  $5 \times 10^{23}$  atoms/cm<sup>3</sup>. So  $n_i = 1.51 \times 10^{10}$  shows that at room temperature only one of every billion atom is ionized.

In a semiconductor recombination causes a pair of free charge carriers, called electron hole pair (EHP) to disappear and breaking of a covalent gives an EHP. Also breaking of a covalent bond ionizes an atom by freeing an electron, for this reason generation of free charge carriers is called **ionization process**.

In a semiconductor movement of charge carrier can be obtained by two mechanism **diffusion** and **drift**. If there is uniform concentration of charge carriers throughout the crystal of a semiconductor there is not net flow of charges *i.e.*, no current. If in one portion concentration is high and in another portion concentration is low then the charge carriers will diffuse to low concentration region and thus constituting **diffusion current  $I_D$** . If an electric field is applied to semiconductors free electrons and holes are accelerated and acquires a velocity called **drift velocity** and is given as

$$V_{\text{drift}} = \mu E \quad \dots(10.3)$$

where  $\mu$  = mobility of charge carriers in cm<sup>2</sup>/V.sec

$E$  = applied electric field in V/Cm

$V_{\text{drift}}$  = drift velocity in cm/sec

Note that mobility of a hole is lower than that of an electron.

The electric current caused by drift velocity is called as **drift current**.

As earlier stated, to increase the conductivity of a semiconductor doping is done. There are two type of doping is used to increase free charge carriers; depending upon weather we want to increase no. of holes or no. of electrons. In a doped semiconductor if impurity increases no. of free-electrons it is called **n-type** semiconductor and in which case electrons becomes majority carrier and holes become minority carrier. If impurity increases no. of free holes it is called **p-type** semiconductor. To obtain n-type semiconductor a pentavalent impurity (*e.g.*, antimony or phosphorous with 5 electrons in outer orbits) is introduced. Out of 5 valence electrons of impurity atoms four electrons forms covalent bonds with surrounding semiconductor atoms and one electron remains-free. Thus the free electron can be donated

for current conduction. For this reason such impurity atoms are called **donor atoms**. Similarly introducing a trivalent impurity (e.g., boron or medium) gives a p-type semiconductor in which holes are majority carriers. In this case only three electron exists in outer orbits which form covalent bonds with three neighbouring atoms. But this impurity is having a vacant position i.e., hole which is free and can not be used to form fourth covalent bond in lattice structure. Thus a free hole is created. But this impurity can accept an electron to form fourth covalent bond causing hole movement. For this reason trivalent impurities are also called **acceptor impurities**.

“It should be noted that p-type or n-type semiconductors are electrically natural. The majority carriers in each type are neutralized by **bound charges** associated with the impurity atoms, see Fig. 10.3.

### 10.1.2 Semiconductor Diode or PN Junction

When a PN-junction is formed then due to concentration gradient, the charge carriers diffuses on either side of the junction. But soon after crossing the junction the free charge carrier disappear by re-combination. This leaves the ion without neutralizing carrier. The process of diffusion continue until a potential barrier against the flow of charge carrier is formed. This is shown in Fig. 10.3. The PN junction in open circuit is sold to be in equilibrium. Since the recombination occur immediately after the junction, this region is out of or depleted of free charge carrier and thus it is called **depletion region**. It is in this region the potential barrier is built up and the charge carrier must overcome this barrier to cross the junction. Symbol of PN junction diode is shown in Fig. 10.4.

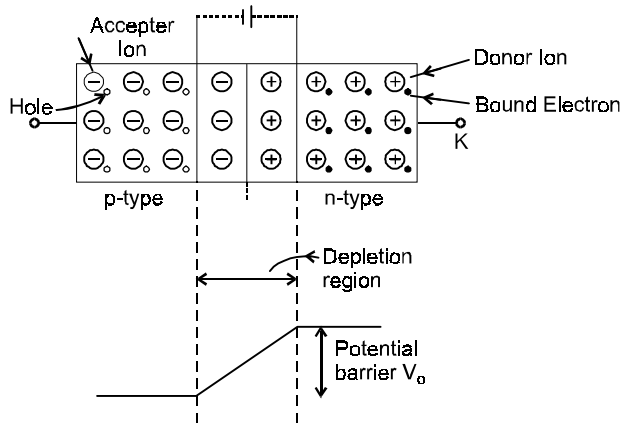


Fig. 10.3 A PN Junction Diode Under Open-Circuit Terminals



Fig. 10.4 Symbol of PN Junction Diode.

When no external voltage applied the inbuilt junction voltage  $V_0$ , shown in Fig. 10.3 is given as

$$V_0 = V_T \log_e \left( \frac{N_A \cdot N_D}{n_i^2} \right) \quad \dots(10.4(a))$$

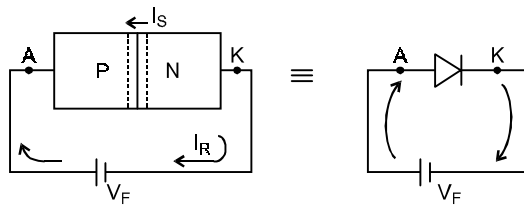
where  $V_T = \frac{K_B \cdot T}{q} = \text{Thermal voltage} \quad \dots(10.4(b))$

$q =$  electronic charge

$N_A, N_D =$  Doping concentration on  $p$ -side and  $n$ -side respectively.

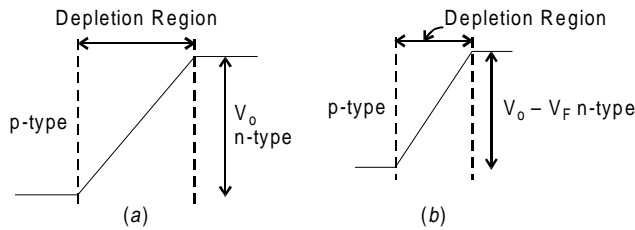
For silicon the junction inbuilt voltage  $V_0$  typically varies between 0.6 to 0.8 V.

Let us connect a dc supply  $V_F$  to the diode such that positive terminal of supply is connected to  $p$ -type and negative is connected to  $n$ -type as shown in Fig. 10.5. This configuration of the diode is **forward biasing**.



**Fig. 10.5** Diode in forward bias

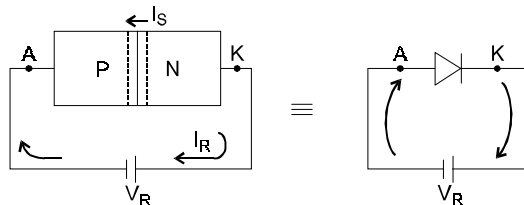
The voltage  $V_F$  is thus called as the forward voltage. As we increase the  $V_F$  the width of the depletion start reducing. Alternately, the potential barrier  $V_0$  reduces as shown in Fig. 10.6(b).



**Fig. 10.6** Potential Barrier at PN Junction  
(a) at equilibrium (open circuit) (b) At forward bias

If we increase  $V_F$  further, situation can soon be reached at which depletion region width is zero. And the moment  $V_F > V_0$ , the charge carriers are swept across the junction under the influence of electric field. Thus current flows through the device in a direction shown in Fig. 10.5. The current flowing through the device in this case is called as forward current.

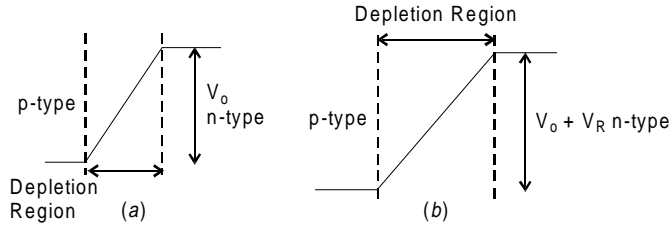
If the batter polarities are reversed *i.e.*, positive terminal is connected to  $n$ -type and negative terminal is connected the  $p$ -type, then we get what called **reverse biasing**. This is shown in Fig. 10.7. Note that battery is now-labelled  $V_R$  (reverse voltage), in order to distinguish it from forward voltage  $V_F$ .



**Fig. 10.7** Diode In Reverse Bias

In this case the electrons of the  $n$ -type are attracted towards the positive plate of battery and holes are attracted towards the negative plate. Consequently the width of depletion

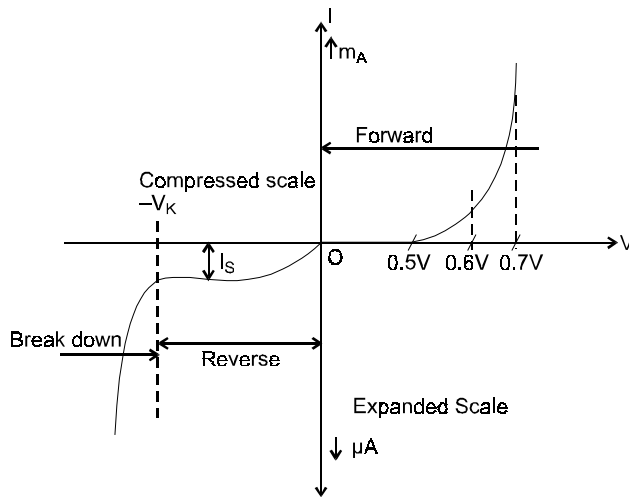
region increases. And increase, in depletion region increases the potential barrier across the junction as shown in Figure 10.8(b).



**Fig. 10.8** Potential Barrier at PN Junction  
 (a) At Equilibrium (open circuit)  
 (b) At reverse bias

In this situation apparently no current flows through the junction. But in reality due to the flow of minority carriers there exist a small reverse current called as **leakage current**.

10.1.2.1 I-V Characteristics of Diodes



**Fig. 10.9** I-V characteristics of a silicon diode with expanded and compressed scale.

As shown in figure the characteristics is divided in 3 parts.

1. The forward biased region  $V > 0$
2. The reverse based region  $V < 0$
3. The breakdown region  $V < -V_K$

Note that scale for  $I < 0$  is expanded and that for  $V < 0$  is compressed.

**Forward Region**

In the forward region  $i - v$  relationship is approximated as

$$I = I_S (e^{V/\eta V_T} - 1) \tag{10.5}$$

where  $I_S$  is a constant for a diode at a given temperature.



$V_T$  is thermal voltage defined by equation 10.4(b).

at  $t = 27^\circ\text{C}$  or  $T = 300^\circ\text{K}$  the thermal voltage

$$V_T \cong 26 \text{ mV} \quad \dots(10.6(a))$$

$\eta$  is a parameter and  $\eta = 1$  for germanium and  $\eta = 2$  for silicon.

The current  $I_S$  in eqn 10.5 is called as SCALE CURRENT. The name is given because  $I_S$  is directly proportional to the cross-sectional area of junction. This means doubling the junction area will double the  $I_S$  which consequently doubles the diode current  $I$  as shown by equation 10.5, for a given forward voltage  $V$ .

An inspection of  $i - v$  characteristics reveals that almost no current flows (*i.e.*, less than 1% or so) for voltages  $V < 0.6 \text{ V}$ . For voltages varying between  $0.6 \text{ V}$  to  $0.8\text{V}$  there flows an appreciable amount of current. Thus we say that  $0.6 \text{ V}$  is a threshold value of voltage that must be applied for conduction. This is called as **cut-in voltage** ( $V_\gamma$ ). Thus for silicon diode cut in voltage is

$$V_\gamma = 0.6 \text{ V} \quad \dots(10.6(b))$$

Since for a fully conducting diode voltage drop lies in a narrow range of  $0.6$  to  $0.8\text{V}$  we can approximate that for silicon diodes

$$V_D = 0.7 \text{ V} \quad \dots(10.6(c))$$

where  $V_D =$  Drop across conducting diode.

In forward region we get appreciable amount of current such that  $I \gg I_S$  so we can approximate equation 10.5 for forward biased region as

$$I = I_S e^{V/\eta V_T} \quad \dots(10.7(a))$$

alternately 
$$V = \eta V_T \log_e \frac{I}{I_S} \quad \dots(10.7(b))$$

**Example 10.1.** Calculate the scale current at room temperature for a  $2 \text{ mA}$  silicon diode in forward biased mode.

**Solution.** By equation 10.6(c) we have drop across conducting diode

$$V_D = 0.7 \text{ V}$$

and at room temperature  $V_T = 26 \text{ mV}$  by equation 10.6(a)

by equation 10.7(a)

$$I_S = I \cdot e^{-V/\eta V_T}$$

for silicon  $\eta = 2$ , given  $I = 2 \text{ mA}$

so 
$$I_S = 2 \times 10^{-3} e^{-(0.7/2 \times 2 \times 26 \times 10^{-3})}$$

or 
$$I_S \cong 2.85 \times 10^{-9} \text{ A } \textit{i.e.}, 1.52.05 \text{ nA at } t = 27^\circ\text{C} \text{ for this diode.}$$

It is worth noting that since both  $I_S$  and  $V_T$  are function of temperature the  $V-I$  characteristics varies with temperature. At a given constant current the drop across diode decreases approximately by  $2.5 \text{ mV}$  by every  $1^\circ\text{C}$  rise in temperature *i.e.*,

$$\frac{dV}{dT} = -2.5 \text{ mV}/^\circ\text{C}$$

**The Reverse Region**

When the applied voltage is reversed and diode voltage  $V$  is made negative we get the reverse bias operation of diode. If  $V$  is negative and is few times larger than  $V_T$  in magnitude we approximate equation 10.5 as

$$I = -I_S \quad \dots(10.8)$$

This means that current in reverse direction is constant in the junction and is equal to  $I_S$ , see Fig. 10.7. Since  $I_S$  is constant it is also called **saturation current**. Practically diodes have quite small reverse current but much larger than  $I_S$  e.g., for  $I_S = 10^{-15}A$ , reverse current  $I_R \cong 10^{-9} A$ . The reverse current also increases with increase of reverse voltage. Major contribution to the reverse current is due to leakage effect. The leakage current is also proportional to the junction area. In general reverse current doubles for every  $10^\circ C$  rise in temperature.

**Breakdown Region**

The breakdown region is reached when reverse voltage exceeds a certain threshold value for a diode, called **breakdown voltage**. This is indicated by the knee of V-I characteristics and that's why it is also called **knee voltage** and is represented by  $V_K$  in Fig. 10.9. In the breakdown region a small increase in reverse voltage results large increase in reverse current. If the dissipated power is limited, the diode breakdown is non destructive.

Also the fact that V-I characteristics in breakdown region is almost a vertical line, enables one to use this region for voltage regulation.

**10.1.2.2 Diode as a Switch**

The P-N junction diode can be operated as a switch in electronics circuits by operating it into forward biased region and reverse biased region.

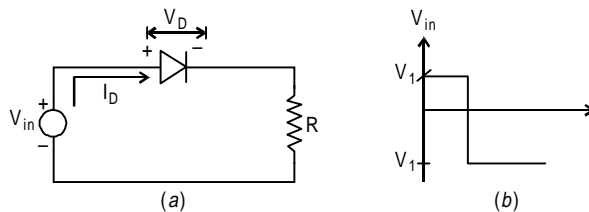
When the diode is operated in forward bias at that time the drop across diode is negligibly small and is almost constant. If the forward current  $I_F$  is limited by a series  $O$  resistor  $R$ , then the forward biased diode can be used as switch in ON condition. This is shown in Fig. 10.10 (a). If the diode is reversed biased then a small current flows through the diode, which is almost constant. This can be used to represent a switch in OFF position.

Now let us apply a voltage  $V_{in}$  as shown in Fig. 10.10(b) and see how the diode circuit shown in Fig. 10.10(a) respond to it.

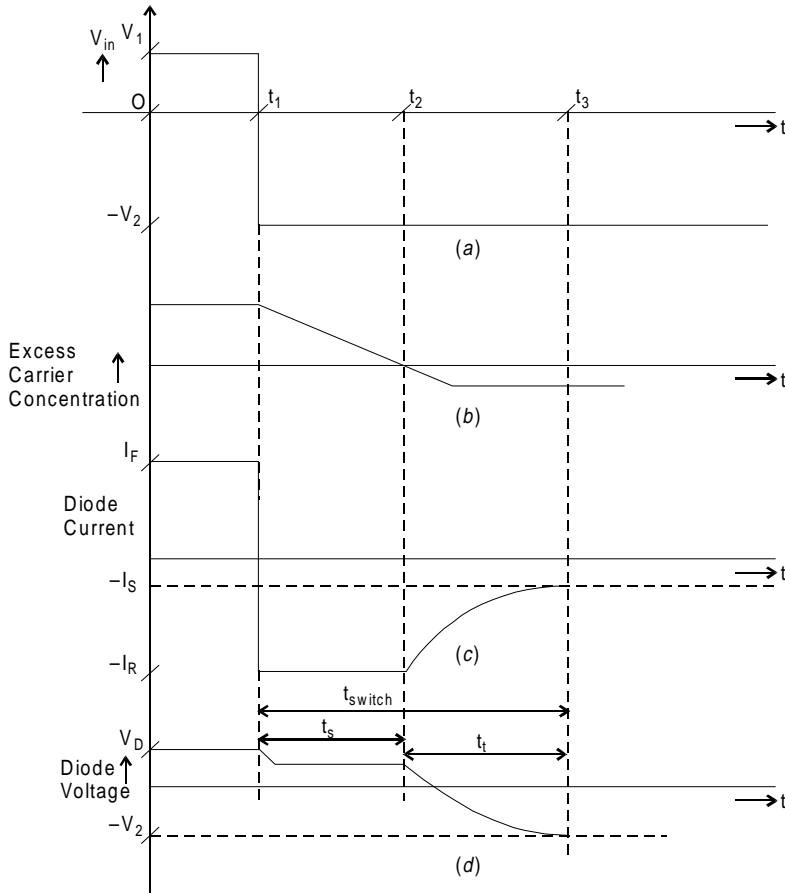
At  $t = 0$   $V_{in} = V_1$  and diodes is in forward biased and the drop across conducting diode is small. This is shown by  $V_D$  in Fig. 10.11(d). The current  $I_F$  flowing in the circuit is approximated as

$$I_F \cong \frac{V_1}{R}$$

This is shown in Fig. 10.11(c).



**Fig. 10.10** Diode circuit with applied voltage in Figure (b).



**Fig. 10.11** Switching waveforms for Fig. 10.10.

Also shown by Fig. 10.11(b) is, when diode is conducting minority charge carriers will accumulated across the junction. This excess concentration will be quite large if diode stays ON for long.

at  $t = t_1$   $V_{in}$  switches to  $-V_2$  but the diode voltage does not changes immediately as shown in Fig. 10.11. This is because during  $t = 0$  to  $t_1$  minority charge carriers were accumulated across the junction. This must be removed to change the state of diode. This takes some time to be done.

“The time required for the removal of excess charge carrier from junction is referred as **storage time ( $t_s$ )**”. In Fig. 10.11 interval  $t_1$  to  $t_2$  is storage time.

Infact to remove excess charge carriers a large reverse current flows till all the excess charges are removed. This is shown in Fig. 10.11(c) in interval  $t_1$  to  $t_2$  where it remains constant and can be approximated as

$$I_R \cong \frac{-V_2}{R}$$

At  $t = t_2$  Excess carriers are removed the current  $I_R$  changes exponentially and goes towards a steady state value of  $-I_S$ . As shown in Fig. 10.11(c) and (d) the diode voltage also

changes exponentially to reach to a steady state value  $-V_2$ . But this takes time and excess charge carrier start accumulating across the junction. Of course this accumulation is very small as was in the case of  $t = 0$  to  $t_1$ .

At  $t = t_3$  Both the  $I_R$  and diode voltage have reached to a steady state value and small amount of excess charge is accumulated.

“Time taken by the diode to reach to a steady state condition after removal of excess carrier concentration is referred as **transition time** ( $t_t$ )”. Interval  $t_2$  to  $t_3$  in figure 10.11 is transition time.

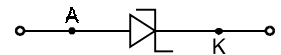
And “the sum of storage time and transition time called as **response time** or **total time delay** of the diode and is oftenty called as **switching time** ( $t_{\text{SWITCH}}$ ) of the diode *i.e.*,

$$t_{\text{SWITCH}} = t_S + t_t \quad \dots(10.9)$$

Since the accumulated excess carrier concentration is large in forward biased it takes a quite longer time to go to OFF state from ON state. Thats why switching speed is measured from delay offered by diode to go to OFF state from ON state.

### 10.1.2.3 Schottky Diode

We have seen in subsection 10.2.2 the switching time is limited by storage time. The switching, time can be improved if storage time is reduced. If a junction is formed by using a SC and a metal then the storage time is reduced dramatically. Such a junction diode is called as SCHOTTKY DIODE, whose symbol is shown in Fig. 10.12. An example is a diode formed using n-type SC and aluminium which has cut-in voltage of 0.35 V.



**Fig. 10.12** Symbol of a schottky diode

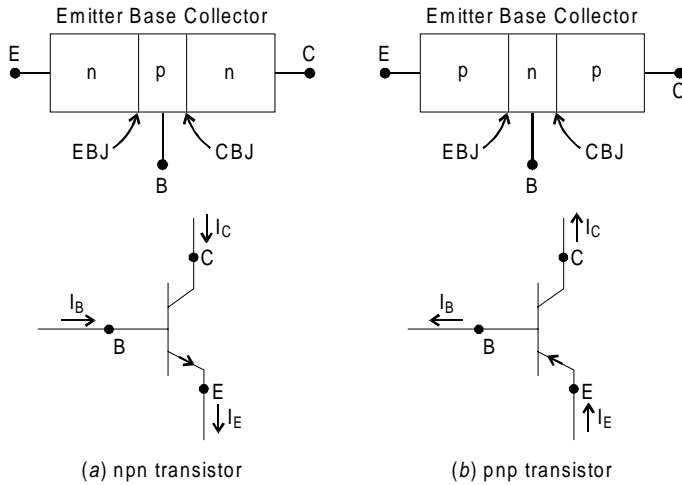
When the diode is forward biased electrons from n-type enters into aluminium where they are not minority carriers as aluminium is a conductor. Thus when junction is reverse biased problem of removing excess minority carriers does not exist. So the SCHOTTKY diodes have negligible storage time and have very high switching speed.

### 10.1.3 Bipolar Junction Transistor (BJTs)

Yet another non linear semiconductor device is a 3-terminal Bipolar junction transistor. The basic principal is to apply a voltage between two terminals to control the current flowing into the third terminal.

A transistor is formed by sandwiching a p-type semiconductor between two n-type semiconductors or by sandwiching an n-type semiconductor between two p-type semiconductors. The former is called *npn* transistor and later is called a *pnp* transistor. A simplified structure with their symbols along with conventional current direction is shown in Fig. 10.13.

An observation of above figure reveals that a transistor is nothing but two PN junction diodes connected back to back. The three regions are named emitter, base and collector which forms two junction the emitter to base junction (EBJ) and collector to base junction (CBJ). To be useful practically emitter is heavily doped, base is lightly doped and width of base is made small *i.e.*, thin base, and collector is made thick offering high resistance.



**Fig. 10.13** The bipolar junction transistor

10.1.3.1 Configuration and Operation of BJTs

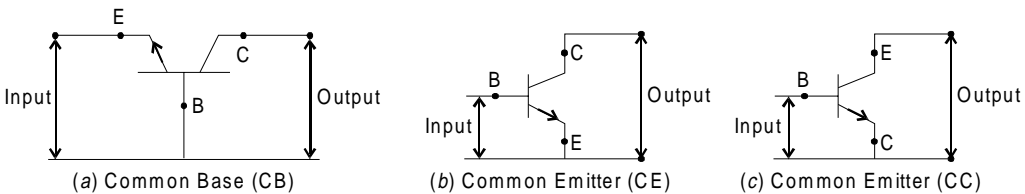
Depending upon the biasing of emitter to base junction (EBJ) and collector to base junction (CBJ) the transistor can be operated in different modes. This is summarised in Table 10.1.

**Table 10.1** Modes of operation of BJT

<i>EBJ Biasing</i>	<i>CBJ Biasing</i>	<i>Mode</i>
REVERSE	REVERSE	CUTOFF
FORWARD	REVERSE	ACTIVE
FORWARD	FORWARD	SATURATION
REVERSE	FORWARD	INVERSE ACTIVE

The Active mode is used for amplification whereas CUTOFF and SATURATION is used for switching application.

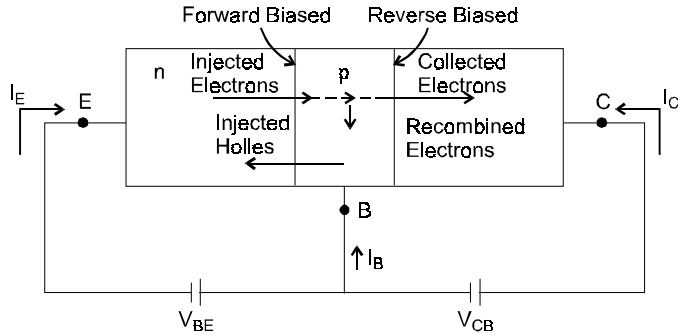
In order to configure the device as two port network one of the three terminals of the device is made common to both the input and output. Thus three different configurations are possible as shown in Fig. 10.13(a). Out of these three configurations the common emitter (CE) configuration is the most used configuration. Note that all the transistors in Fig. 10.13 are *n-p-n* transistors.



**Fig. 10.13(a)** Different configurations of transistors

**OPERATION**

To understand the operation consider the biasing of *npn* transistor, shown in Fig. 10.14. The diagram shows simplified carrier movement due to diffusion only. The EBJ is forward biased by dc supply  $V_{BE}$  and CBJ is reverse biased by supply  $V_{CB}$ .



**Fig. 10.14** Biased npn transistor.

The forward biasing of EBJ causes a current to flow across the junction due to electrons injected to base from emitter and due to holes injected from base to emitter. Flow of both the carriers constitute the current in same direction. Moreover since emitter is much heavily doped and base is lightly doped the emitter current is entirely due to the electrons. Upon entering into the base the electrons injected becomes minority carrier. Some of these electrons will recombine with holes in base and rest can reach to CBJ. The positive collector voltage  $V_{CB}$  causes these electrons to be swept across the CBJ depletion region in to the collector. Thus they get collected to constitute collector current  $I_C$ . Since width of the base region is very thin, recombination in base region is very small. Thus almost all the injected electrons can be collected into collector region. The collector current  $I_C$  is given as

$$I_C = I_S e^{V_{BE}/V_T} \quad \dots(10.09)$$

where

$I_S$  = saturation current or current scale factor

and

$I_S \propto$  junction area EBJ (or device size)

$V_{BE} = V_B - V_E$  = Base to emitter voltage

$V_T = \frac{K_B \cdot T}{q}$  = Thermal voltage = 26 mV at room temperature.

Note that  $I_C$  is independent of base to collector voltage  $V_{CB}$ . This means as long as collector is positive with respect to base (*i.e.*,  $V_{CB} = V_C - V_B > 0$ ) electrons injected from emitter will be swept into collector to make collector current. This is a very important observation because it also *implies that if, "VCB < 0 i.e., CBJ is forward biased then injected electrons may not be swept across the junction and collector current remains unaffected"*. This fact can be utilized for switching. As evident from the figure the base current is made up of two components. One is due to the injected holes into emitter and other is due to holes that must supplied by the battery for every hole disappeared due to the recombination in base region. Both the current adds together to give base current  $I_B$ .

Analysis shows that infact the base current is a fraction of collector current and is represented as

$$I_B = \frac{I_C}{\beta} \quad \dots(10.10)$$

where  $\beta$  is a constant for particular transistor and is called common emitter current gain. The  $\beta$  depends upon width of base and relative dopings of base and emitter region. To achieve high  $\beta$  (which is usually the case) base should be very thin and emitter must be heavily doped with lightly doped base.

The emitter current is sum of base current and collector current as shown in Fig. 10.14. Thus

$$I_E = I_B + I_C \quad \dots(10.11(a))$$

by equation (10.10) we get

$$I_E = \frac{I_C}{\beta} + I_C$$

or 
$$I_E = \frac{\beta + 1}{\beta} I_C \quad \dots(10.11(b))$$

if we define 
$$\alpha = \frac{\beta}{\beta + 1} \quad \dots(10.11(c))$$

then we get 
$$I_C = \alpha I_E \quad \dots(10.11(d))$$

where  $\alpha$  = common base current gain alternatively for  $\beta$

$$\beta = \frac{\alpha}{1 - \alpha} \quad \dots(10.11(e))$$

Since  $\alpha$  is constant and is less than but closer to unity it is a consequence of equation 10.11(e) that a small change in  $\alpha$  will result in large change in  $\beta$ .

Although we have presented the working with a *n-p-n* transistor, the same is applicable for *p-n-p* transistor but with all current directions and voltage polarities reversed as in *p-n-p* transistors majority carriers are holes.

**Example 10.2.** Design the circuit given in 10.15 to have a collector current of 1 mA with collector voltage of 5V. Given that transistor has  $\beta = 50$  and base to emitter drop  $V_{BE} = 0.7$  V.

**Solution.** Given  $\beta = 50$ ,  $I_C = 1$  mA,  $V_C = 5$ V and  $V_{BE} = 0.7$  V

The current  $I_C$  is flowing through the resistor  $R_C$  and is given as

$$I_C = \frac{15 - 5}{R_C}$$

or 
$$R_C = \frac{10}{I_C}$$

at 
$$I_C = 1$$
 mA

we get 
$$R_C = 10$$
 K $\Omega$

we have 
$$I_B = \frac{I_C}{\beta}$$

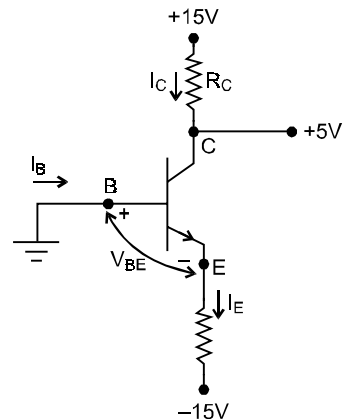
so at  $I_C = 1$  mA 
$$I_B = \frac{1 \times 10^{-3}}{50}$$

or 
$$I_B = 20$$
  $\mu$ A

again we have 
$$V_{BE} = V_B - V_E$$

so 
$$V_E = V_B - V_{BE} = 0.7$$
 V

i.e., 
$$V_E = -0.7$$
 V



**Fig. 10.15** Circuit for Example 10.2.

we have  $\alpha = \frac{\beta}{\beta + 1}$

so at  $1\beta = 50, \alpha \cong 0.98$

we have  $I_C = \alpha I_E$

or  $I_E = \frac{1 \times 10^{-3}}{0.98}$

*i.e.*,  $I_E \cong 1.02 \text{ mA}$

The same value can also be calculated by

$$\begin{aligned} I_E &= I_C + I_B \\ &= 1 \text{ mA} + 20 \text{ } \mu\text{A} = 1.02 \text{ mA} \end{aligned}$$

emitter current  $I_E$  can be given as

$$I_E = \frac{V_E - (-15)}{R_E}$$

or  $R_E = \frac{V_E + 15}{I_E} = \frac{-0.7 + 15}{1.02 \times 10^{-3}}$

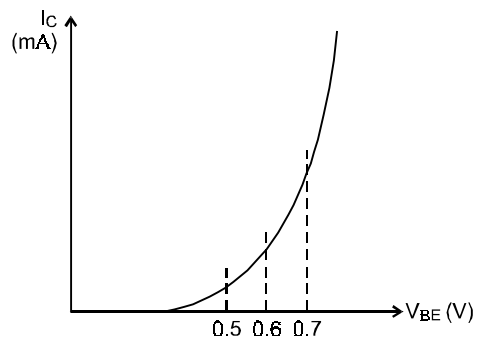
finally  $R_E \cong 14 \text{ K}\Omega$

### 10.1.3.2 BJT Characteristic

The first characteristic we consider is the plot of collector current with respect to base to emitter drop across the junction. In CE configuration the base is used to supply the input and the collector is used to take the output. Thus the plot shown in Fig. 10.16 for a silicon npn transistor displays the transfer characteristics. As shown in figure when the base to emitter drop is below 0.5 V then there flow a small amount of current in the collector junction. The moment drop across base to emitter junction increases 0.5 V the current increases appreciably and at about 0.7 V drop, large collector current flows in the collector circuit. Thus we can regard  $V_{BE} = 0.5 \text{ V}$  as **cut-in voltage** and  $V_{BE} = 0.7 \text{ V}$  as **active voltage** for a *npn* transistors madeup of silicon.

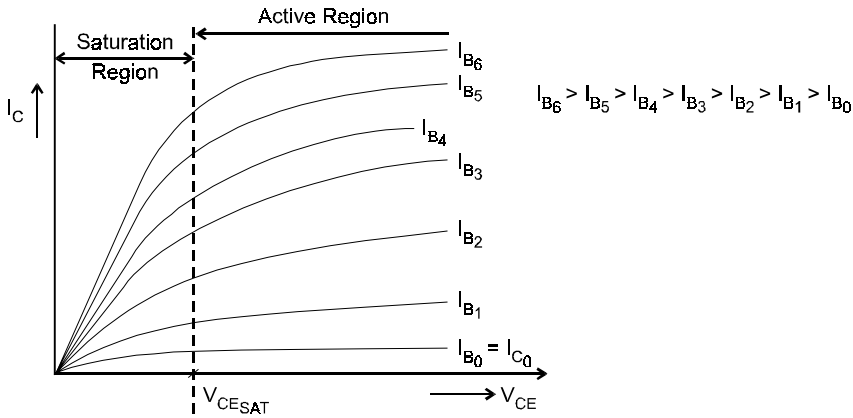
A plot of collector current with respect to collector to emitter voltage drop for a given value of base current is called as the output characteristic of transistor, as shown in Fig. 10.17

The output characteristics can be divided into three regions. The active region, cut off region, and saturation region. The active region operation is what we discussed earlier to describe the operation. This region is used for amplification in which small change in base current results in larger current.



**Fig. 10.16** Transfer characteristics of CE configuration.





**Fig. 10.17** Output Characteristics of CE Configuraiton

In the saturation region both CBJ and EBJ are forward biased. Since the voltage across EBJ and CBJ are very small under forward biased, the collector to emitter voltage also is very small as

$$V_{CE} = V_{BE} - V_{BC} \quad \dots(10.12)$$

In the saturation region the collector current does not change appreciably and can be considered that collector current is approximately independent of base current. In saturation region the collector current may be given as

$$I_{Csat} = \frac{V_{CC} - V_{CEsat}}{R_C} \cong \frac{V_{CC}}{R_C} \quad \dots(10.13(a))$$

because  $V_{C_{sat}}$  is very small and is approximately 0.2 V for silicon *npn* transistor operating well into saturation. a good check of saturation operation is

$$\beta_{min} I_B \geq I_{Csat}$$

alternately 
$$h_{FEmin} I_B \geq I_{Csat} \quad \dots(10.13(b))$$

where  $h_{FE}$  is just an alternate representation for current gain  $\beta$ .

The cut off region approached when both the CBJ and EBJ are reverse biased and no emitter current flows. Even though both the junctions are reverse biased there flows a small amount of current at CBJ called reverse saturation current  $I_{CO}$ . Thus to make  $I_E = 0$ ,  $I_B = -I_{CO}$  as  $I_C = I_{CO}$  at this time. Thus we can say that cutoff is reached when  $I_C = I_{CO}$ ,  $I_B = -I_{CO}$ , thus making  $I_E = 0$  and  $V_{BE} = 0V$  for Si and  $V_{BE} \cong -0.1 V$  for Ge.

Table 10.2 may serve as reference to choose typical values for different mode of operation.

**Table 10.2 Typical npn transistor junction voltages at 25°C**

Semiconductor	$V_{BEcut\ in} (V_\gamma)$	$V_{BE\ ACTIVE}$	$V_{CE\ SAT}$	$V_{BE\ SAT}$	$V_{BE\ CUTOFF}$
Si	.5V	0.7	0.2	0.8	0.0
Ge	0.1	0.2	0.1	0.3	-0.1

**Example 10.3.** For the circuit shown in figure 10.18 determine whether the silicon npn transistor with  $h_{FE} = 100$  is in saturation or not. Use table 10.2 for typical values.

Given  $h_{FE} = 100$

$R_C = 5K$

$R_B = 20K$

**Solution.** Let us first assume that transistor is in saturation and calculate the currents. If calculated values confirm saturation then we say it is in saturation otherwise not. Apply KVL in base circuitry we get.

$$5 - I_B \cdot 20 \times 10^3 - V_{BEsat} = 0$$

so 
$$I_B = \frac{5 - V_{BEsat}}{20 \times 10^3} = \frac{5 - 0.8}{20 \times 10^3}$$

or 
$$I_{Bsat} = 0.21 \text{ mA} \quad \dots(1)$$

Applying KVL to output side we get

$$V_{CC} - I_{Csat}R_C - V_{CEsat} = 0$$

or 
$$I_{Csat} = \frac{V_{CC} - V_{CEsat}}{R_C} = \frac{10 - 0.2}{5 \times 10^3}$$

or 
$$I_{Csat} = 1.96 \text{ mA} \quad \dots(2)$$

by equation 10.13(b)  $h_{FE}I_B \geq I_C$  in saturation.

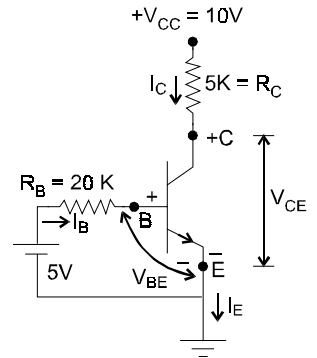
$$h_{FE}I_B = 100 \times 0.21 \times 10^{-3} = 21 \times 10^{-3}$$

Since  $h_{FE}I_B$  ( $21 \times 10^{-3}$ ) is greater than  $I_{Csat}$ , the transistor is in saturation region.

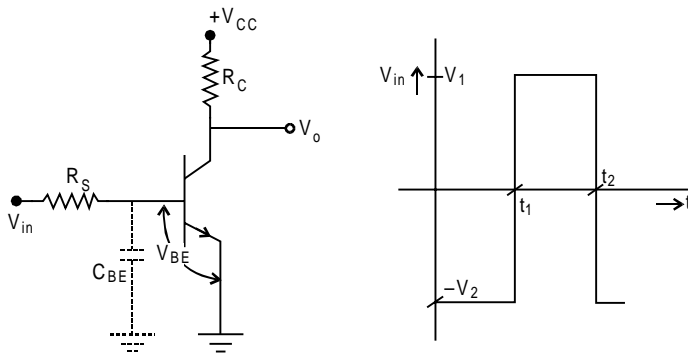
### 10.1.3.3 Transistor as a Switch

To understand the switching characteristics let us consider the transistor as inverter as shown in Fig. 10.19, along with input voltage.

Consider the input for time  $t < t_1$ . Waveform shows that both EBJ and CBJ are reverse biased as  $V_{in} = -V_2$ .



**Fig. 10.18** Circuit of Example 10.3.



(a) Inverter Circuit

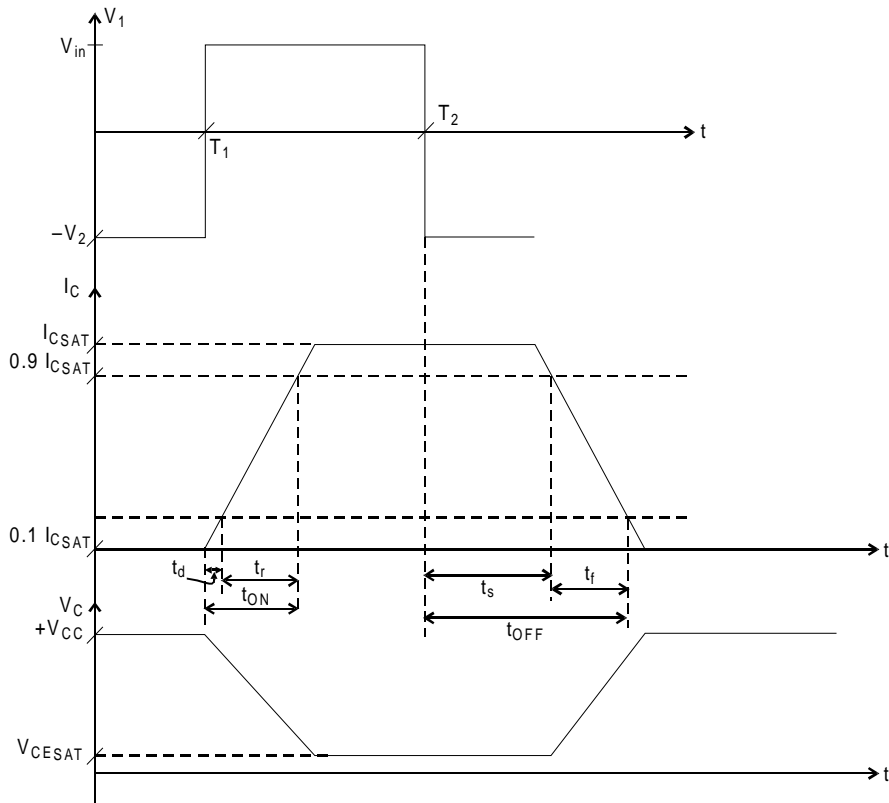
(b) Input to Inverter

**Fig. 10.19** npn transistor as inverter

At this moment almost all the supply voltage  $+V_{CC}$  appear at output and no collector current flows in circuit. This is shown in Fig. 10.20 for  $t < t_1$ . This represents the transistor in OFF state. Note that at this moment transition capacitance  $C_{BE}$  is charged to  $-V_2$ . The moment input changes instantaneously to  $V_1$  the transistor can not change its state instantaneously. The capacitance  $C_{BE}$  starts charging towards  $V_1$  (from  $-V_2$ ). The moment  $V_{BE} > V_\gamma$  transistor starts conducting after some times it enters into saturation. This is shown in Fig. 10.20 for  $t > t_1$ . Here the **delay time** ( $t_d$ ) is the time taken by collector current to rise to 10% of  $I_{C_{sat}}$  from 0. Also shown is **rise time** ( $t_r$ ) is time taken by collector current to reach to 90%  $I_{C_{sat}}$  from 10% of  $I_{C_{sat}}$ . Sum of delay time and rise time is referred as **ON time** ( $t_{ON}$ ) of the transistor. *i.e.*,

$$t_{ON} = t_d + t_r$$

Note that during this time the collector voltage will be reduced to  $V_{CE_{SAT}}$  from  $V_{CC}$ , as shown in figure.



**Fig. 10.20** Switching Characteristics of Tansistor.

At time  $t = t_2$  the input changes abruptly to  $-V_2$  from  $V_1$ . Again the transistor takes some time to change its state. This is mainly due to two facts. First enough charges are accumulated in base due to saturation. Recall that in saturation injected carriers may not all be swept across CBJ. It takes some time to remove these carriers. Second, even before the input goes below cut off many charge carreirs are already injected. These two facts gives rise to delay to switching effect. Thus after certain time transistor becomes OFF and collector

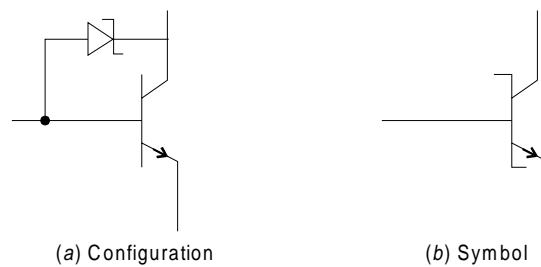
voltage rises to  $+V_{CC}$ . As shown in Fig. 10.20 **storage time** ( $t_s$ ) is the time taken by collector current to reduce to 90% of  $I_{C_{sat}}$  after the input has change its state from positive to negative value. **Fall time** ( $t_f$ ) is the time taken by collector to reduce to 10% of  $I_{C_{sat}}$  from 90% of  $I_{C_{sat}}$ . Sum of storage time and fall time is defined as **OFF time** ( $t_{OFF}$ ) of the transistor *i.e.*,

$$t_{OFF} = t_s + t_f$$

From the figure it is clear that OFF time is larger than the ON time. This is mainly due to the storage time, which is significant due to the saturation region, operation.

#### 10.1.3.4 Schottky Transistor

In previous subsection we pointed out that main victim of switching delay of a transistor. If we connect a schottky diode between base and collector as shown in Fig. 10.21(a), then transistor can be prevented to enter into saturation region. When the transistor is in active region the diode is in reverse bias. When the transistor goes to saturation, the diode conducts. The diode drop (typically 0.4 V) is applied to CBJ and it does not allow the transistor to enter into saturation as  $V_{BC} = V_B - V_C = V_D$ . Very fast switching speed can be obtained when using schottky transistors in diode circuit.



**Fig. 10.21** Schottky Transistor

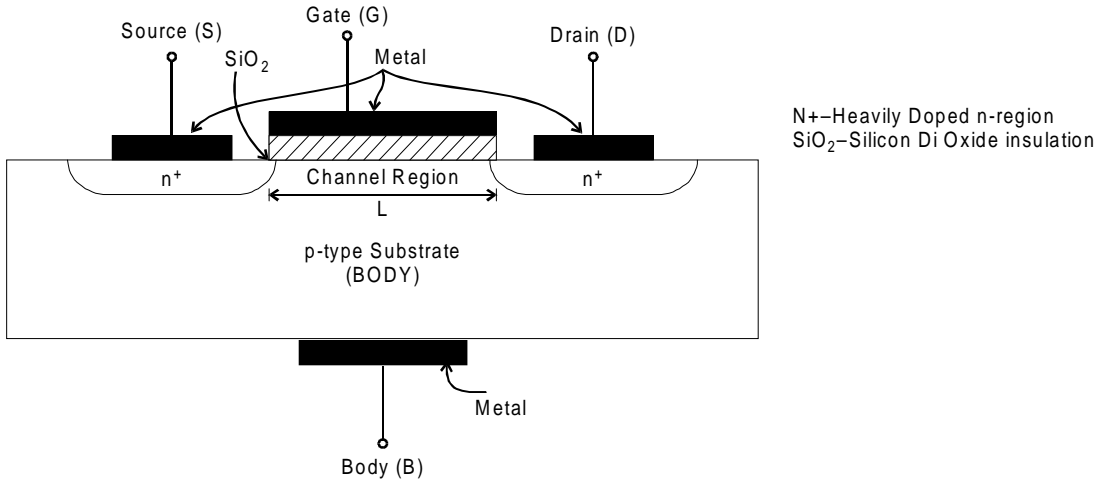
#### 10.1.4 Metal Oxide Semiconductor Field Effect Transistors (MOSFET)

These devices have current control mechanism based on an electric field established by voltage applied to the control terminal. In these devices the current is conducted by only one type of charge carriers, either electrons or holes, and thus called **unipolar transistors**. Important characteristics of this device is that it can be fabricated is very small area as compared to BJTs and for this reason they are most widely used in very large scale integrated (VLSI) circuits. Circuits implemented using MOS devices do not need resistors, diodes, etc. rather they can be implemented using MOS devices only. Generally there are two types of MOS devices called ENHANCEMENT type and depletion type. The enhancement type are widely used.

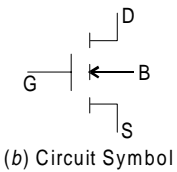
##### Enhancement Type MOSFET

The basic structure and circuit symbol of enhancement type n-channel MOS (or NMOS) transistor is shown in Fig. 10.22. The device is grown on a p-type substrate called body. It contains two heavily doped n-type regions called source and drain. On the top of device a thin layer of  $\text{SiO}_2$  is deposited between sources and drain. A metal is deposited on the top of this layer and the terminal taken from it is called gate. Due to the thin layer of  $\text{SiO}_2$  there exist a small current in gate terminal. It is evident from figure substrate forms two PN junctions with source and drain. In normal operation they are kept reverse biased by connecting the

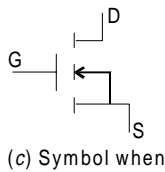
body and source terminal to the ground. Thus the substrate have no effect on operation as it is always shorted with source and device becomes three terminal device. When proper voltages are applied the current flows between source and drain through the region labelled channel which is of length  $L$ . When no voltage applied no current flow through channel.



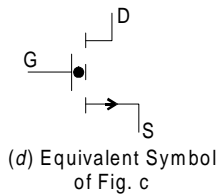
(a) Structure of Enhancement type n-channel MOS (or NMOS) transistor



(b) Circuit Symbol



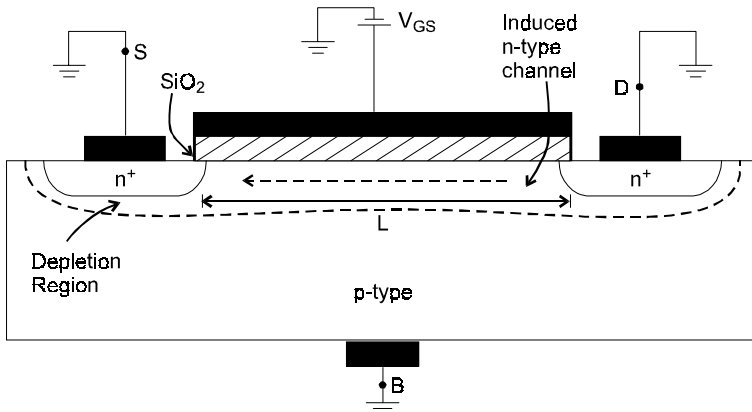
(c) Symbol when B and S shorted



(d) Equivalent Symbol of Fig. c

**Fig. 10.22** Enhancement Type NMOS.

Let the gate is connected to positive voltage as shown in Fig. 10.23, with source and drain grounded. The positive voltage at gate causes the free holes to be repelled from the



**Fig. 10.23** Enhancement type NMOS with Positive Voltage at Gate.

channel region and thus leaving behind a carrier depletion region, as shown in Fig. 10.23. At the same time positive gate voltage attracts electrons from  $n^+$  regions into channel

region. When sufficient number of electrons are accumulated in the channel region, it creates an  $n$ -region connecting drain and source. This is called induced  $n$ -type channel. When a voltage is applied at drain the current can flow through this region. The value of  $V_{GS}$  at which the  $n$ -channel is induced is called as **threshold voltage ( $V_t$ )**. The gate and body of MOSFET form a parallel plate capacitor with  $\text{SiO}_2$  layer acting as dielectric. The electric field developed between the plates of capacitor controls the amount of charge in the channel and thus determines the channel conductivity.

If we apply a small positive voltage  $V_{DS}$  shown in Fig. 10.24 then a current  $I_D$  flows in induced  $n$ -channel from drain to source. Magnitude of  $I_D$  depends upon the density of electrons in induced channel which in turns depends upon the magnitude of  $V_{GS}$ . If  $V_{GS}$  exceeds  $V_t$  density of free electrons increases in induced channel. This means  $V_{GS} > V_t$  enhances the channel and hence the name **enhancement type MOS**. At  $V_{GS} = V_t$  channel is just formed and above  $V_t$  its conductivity is increased. Thus the conductivity of induced channel is proportional to excess voltage  $V_{GS} - V_t$  which is also called as **effective voltage**. It is also evident from the Fig. 10.24 that the current ( $I_S$ ) leaving source terminal is same as the current ( $I_D$ ) entering into drain terminal. Thus only electrons contribute the current in the channel and the device is called **unipolar**. In essence, the voltage  $V_{DS}$  appears as drop across the length of channel. When  $V_{DS}$  is small, say 0.1 V or 0.2V then shape of channel is almost uniform as shown in Fig. 10.25. In this case  $I_D$  Vs  $V_{DS}$  is almost linear and this is shown in Fig. 10.26. If we increase the  $V_{DS}$  then we find that width of induced channel on the drain side reduces where as it remains as it is on source side. Thus the voltage at drain end also reduces by  $V_{GS} - V_{DS}$ , as shown in Fig. 10.25. At this time less room is available for conduction and hence channel resistance increases. This causes a bend in  $I_D$  Vs  $V_{DS}$  curve, shown in figure 10.26. When we make  $V_{DS} = V_{GS} - V_t$  or  $V_{DS} - V_{GS} = V_t$  the width of channel on the drain side is almost zero, and the channel is said to be **pinched OFF**. At this time maximum possible resistance is offered by induced channel. Because the channel width can not be reduced further, see Fig. 10.25. Since beyond this value  $V_{DS}$  resistance is constant the current flowing in the channel is also constant, as shown in Fig. 10.26. The value of  $V_{DS}$  beyond which constant current flows into the channel is referred  $V_{DSSAT}$ . The drain to source voltage at which the pinch off occurs when  $V_{GS} = 0$ , is called as **pinch off voltage ( $V_{PO}$ )**.

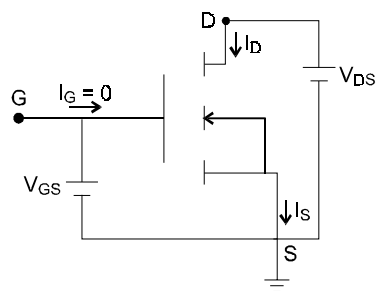


Fig. 10.24 Enhancement type NMOS with  $V_{DS}$  and  $V_{GS}$ .

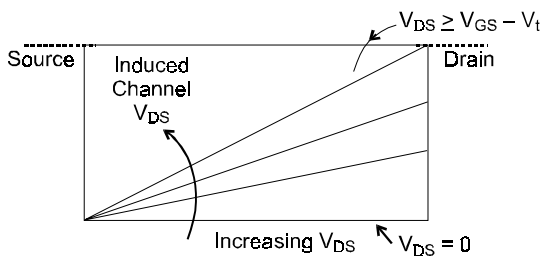
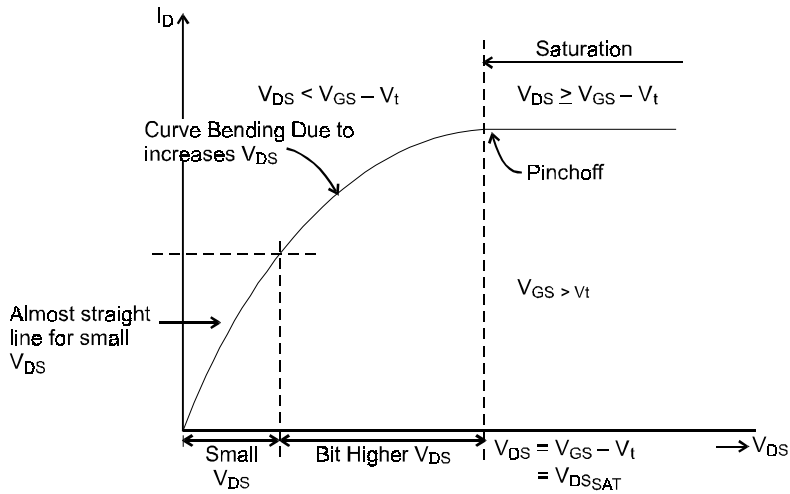
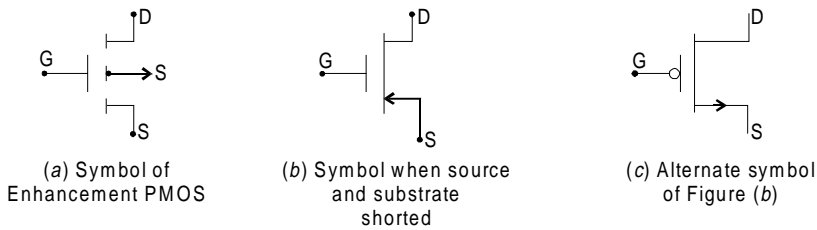


Fig. 10.25 Change in shape of channel with increase in  $V_{DS}$ .



**Fig. 10.26**  $I_D$   $V_S$   $V_{DS}$ .

The **P-channel enhancement type MOSFET** is formed by using *n*-type substrate and *p*<sup>+</sup> type source and drain. In case of PMOS the induced channel is *p*-type region. The circuit symbols of PMOS are shown below.



**Fig. 10.27** Circuit symbols of enhancement PMOS.

Note that in NMOS current flows due to electrons and in PMOS due to holes. Since mobility of electrons is higher than holes, thus NMOS is faster than PMOS. Since mobility of electrons is 3 time greater than holes PMOS devices needs 3 times large area to generate the same amount of current that of NMOS. Thus packing density of NMOS is greater than that of PMOS.

### 10.1.5 The Complementary MOS (CMOS)

The most widely appreciated MOS technology is **COMPLEMENTARY MOS** or CMOS technology. As the name implies the CMOS technology uses both PMOS and NMOS transistors. At present time the CMOS has replaced almost all the NMOs circuit designs. For the fabrication the two transistors are grown on same *p*-type substate and their drain are separated by deposition  $SiO_2$ . An *n*-type well is created in the *p*-type substate to facilitate PMOS its packing density is bit lower than NMOS. But in terms of power dissipation the CMOS performance is dramatically improved. This gives enough motivation for CMOS to be used as powerfull circuit designing tool.

Before we end the discussions an MOS devices we state few more properties for which they superceeds the BJT. For the fabrication point of view, the chip area required by a MOS transistor is just about 5% of that required by BJTs, making high packing density of MOS.

The input capacitance and resistance of MOS devices are very high, thus they can be easily driven by a source as compared to the BJTs. An even attractive feature is that CMOS devices can operate on a wide range of supply voltages, in particular from 3V to 15V. This makes interfacing, a problem discussed later, of CMOS easier with other logics. As earlier mentioned the power dissipation is minimum with CMOS as compared to any other technology.

### 10.2 CHARACTERISTICS OF LOGIC FAMILIES

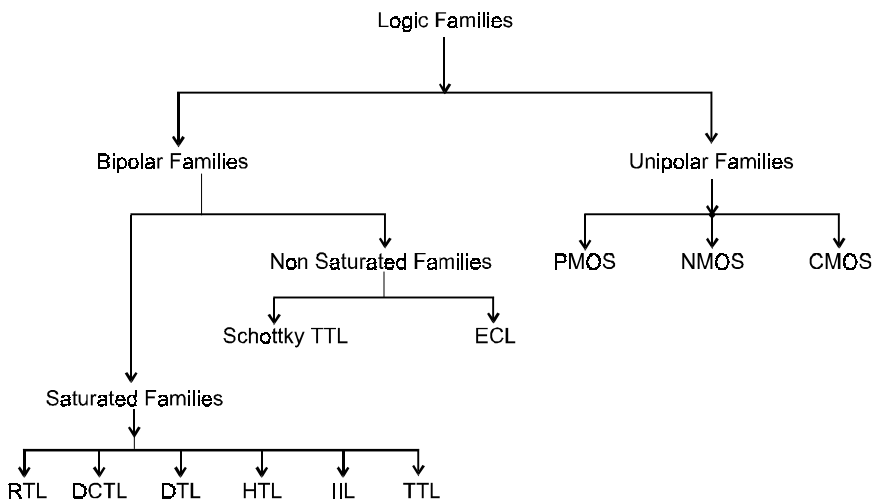
The different types of logic functions can be realized by a variety of digital circuits. Each of these circuits have different form of interconnection between semiconductor devices and the circuit elements. Each form of such circuits can be used to realise a different logic function, such that all the circuitries of this form posses similar characteristics. Thus they can be brought to a single group called a **Digital Logic Family**.

In short a **digital logic family** is a group of compatible logic devices. **Compatibility** means input and output characteristics of the devices match with each other.

#### 10.2.1 Classification of Logic Families

In the Section 10.1 we studied different semiconductor devices. Based upon the devices and their mode of operation used to realize the logic function provides the basis for classification. Mainly the classification is made on the basis of types of charge carriers contributing for current conduction. They are classified as **Bipolar logic families** and **Unipolar logic families**. Diagram shown in Fig. 10.28 summarizes this classification.

The main elements of Bipolar logic families are resistors, diodes (which are also capacitors) and transistors. Where as in Unipolar logic families the only element is the particular type of MOS transistor. In bipolar families the RTL and DTL are obsolete now and in present time TTL and ECL are mostly used. The TTL is further classified as schottky TTL giving attractive range of families suitable for various applications. In unipolar families PMOS is totally obsolete due to their low speed. NMOS is widely used for high packing density and complicated logic devices, like memories and microprocessors. CMOS is an attractive choice for the fabrication of low power consumption devices like calculators but they have a bit slower speed.



**Fig. 10.28** Classification of Logic Families.



Yet another classification of digital ICs can be made on the basis of level of integration of logic gates in a single IC. This is shown by table 10.3.

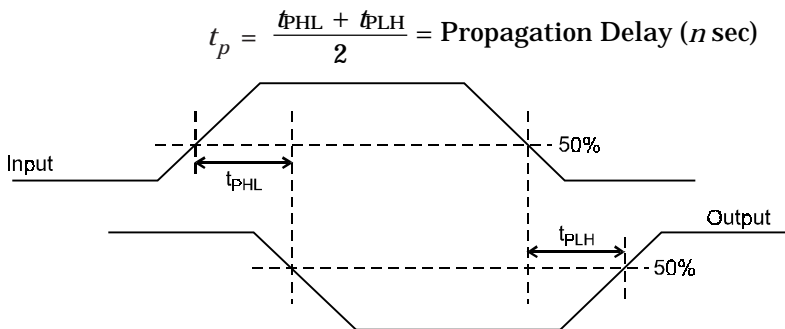
**Table 10.3 Classification of Digital ICs based on Level of Integration**

<i>Integration Scale</i>	<i>Number of Gates Fabricated</i>
SSI – Small Scale Integration	Less than 12 gates
MSI – Medium Scale Integration	$12 < \text{Gates} \leq 100$
LSI – Large Scale Integration	$100 < \text{Gates} \leq 1000$
VLSI – Very Large Scale Integration	$1000 < \text{Gates} \leq 10,000$
ULSI – Ultra Large Scale Integration	Gates $> 10,000$

### 10.2.2 Characteristics of Digital ICs and families

Emergence of various logic families and integration techniques have resulted in widespread use of digital ICs. It is thus necessary to study different performance parameters and characteristics of logic families. They adequately reflects their relative advantages and disadvantages and may serve as a basis to choose a particular family for the application. Infact our study and analysis of different families will be around these parameters only.

- **Propagation Delay** is the maximum time taken by output to change its state in response to input. The propagation delay determines the speed of operation of a gate. In general switching speed is measured when 50% duty cycle time square wave is applied at a input and a square wave is generated at output, as shown in Figure 10.28(a). The times are measured from 50% of voltage levels. The time  $t_{\text{PHL}}$  is delay when output goes LOW from HIGH and  $t_{\text{PLH}}$  is the time delay taken by output to go HIGH from LOW state. The propagation delay  $t_p$  is average of the two times and measured in  $n$  sec.



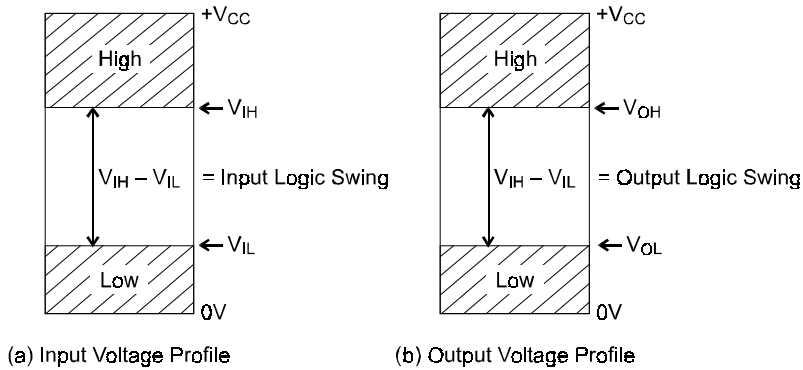
**Fig. 10.28(a)** Input-output waveforms for propagation delay

- **Power Dissipation ( $P_D$ )** is defined as the power dissipated in an IC and measured in mW. It is desired to have low power dissipation to reduce cooling, but it may increase the propagation delays.
- **Speed power product** is defined as the product of propagation delay (in nano seconds) and power dissipation (in mW) and is measured in pico joules. It is also referred as the figure of merit of a digital IC.

$$\begin{aligned} \text{figure of merit} &= \text{Propagation delay} \times \text{Power Dissipation} \\ &= t_p \times P_D = n \text{ sec} \times \text{mW} \\ &= \rho J \end{aligned}$$

- **Fan In** is defined as the number inputs that a logic gate can have
- **Fan Out** is defined as number of similar gates that a logic gate can drive. High fan out is desirable.
- **Input and Output Voltages** various input and output voltage levels are shown in Fig. 10.29.

**High Level Input Voltage ( $V_{IH}$ )** is defined as the minimum input voltage recognized as logic 1.



**Fig. 10.29** Illustration of Voltage Profiles

**Low Level Input Voltage ( $V_{IL}$ )** is defined as maximum value of input voltage recognized as logic 0.

**High Level Output Voltage ( $V_{OH}$ )** is defined as the minimum value of output voltage when output is logic 1.

**Low Level Output Voltage ( $V_{OL}$ )** is defined as the maximum voltage that appears at output when output is logic 0.

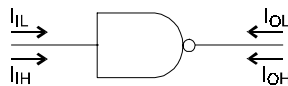
The voltage separation between the two logic states is defined as the *logic swing*.

$$\text{Input logic swing} = V_{IH} - V_{IL}$$

$$\text{Output Logic swing} = V_{OH} - V_{OL}$$

- **Input and Output Currents** The important input and output currents are shown in Fig. 10.30.

**HIGH LEVEL INPUT CURRENT ( $I_{IH}$ )** is defined as the minimum current that must be supplied to a gate corresponding to logic 1 input.



**Fig. 10.30** Illustration of currents in a gate

**Low Level Input Current ( $I_{IL}$ )** is defined as the minimum current that must be supplied to the gate corresponding to logic 0 input.

**High Level Output Current ( $I_{OH}$ )** is defined as the maximum amount of current that a gate can sink when output is logic 1.

**Low Level Output Current ( $I_{OL}$ )** is defined as the maximum amount of current that a gate can sink when output is logic 0.

Why we have shown the output currents as sink current will be clear when discussing the circuits of various families.

- **Supply Currents** it is important to consider the current drawn in **low** and **high** state.

**High State Supply Current ( $I_{CC(1)}$ )** is the current drawn from supply when output of gate is logic 1.

**Low State Supply Current ( $I_{CC(0)}$ )** is the current drawn from supply when output of gate is logic 0.

We can calculate average power dissipation by calculating the dissipation in two states.

The Low state power dissipation  $P(0) = V_{CC} \cdot I_{CC(0)}$

The High state power dissipation  $P(1) = V_{CC} \cdot I_{CC(1)}$

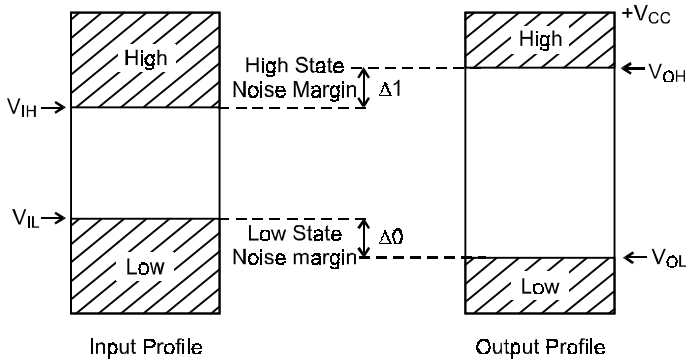
The average power dissipation  $P_{avg} = \frac{P(1) + P(0)}{2}$

This is how we will calculate the power dissipation of various families in next sections.

- **Noise Margin** Unwanted electric signals, called noise, can appear when connecting the logic devices. These noise signals can cause the voltage levels to rise or reduce from intended value. This may cause circuit malfunction. For example if noise causes an input voltage to reduce below the  $V_{IH}$ , then input is not recognized as logic 1 and thus the circuit malfunctions. The ability of a circuit to tolerate the effect of noise is called as **noise immunity**. The amount by which a circuit can tolerate the effect of noise is called **noise margin**. The noise margins are illustrated in Fig. 10.31. Margins shown in the figure are called **dc noise margin**, they are

$$\Delta 1 = V_{OH} - V_{IH} = \text{High state noise margin}$$

$$\Delta 0 = V_{IL} - V_{OL} = \text{Low state noise margin}$$



**Fig. 10.31** Illustration of Noise Margins.

In fact noise is a random signal, considered as ac signal with some amplitude and pulse width. If the pulse width is large compared to propagation delay they can be treated as dc. If the pulse width is less than propagation delay, then pulse width is not enough to change the state of circuit. In this situation the noise pulse amplitude must be large enough to change the state of circuit. Thus the logic circuits have high **ac noise margins**. It is specified in data sheet in form of a curve of noise margin Vs noise pulse width.

- **Operating Temperature** is the range of temperature in which the digital IC can function properly. In general 0 to +70°C is the range for consumers and industrial applications. A range of -55°C to 125°C is accepted for military applications.

### 10.3 IMPLEMENTATION OF LOGIC FAMILIES

So far we have studied the switching characteristics of semiconductor switching devices and logic families. We now turn our attention to the implementation of gates of different families. Here we start with the basic diode logic then move to various families of BJTs and then to MOS family which are most important for fabrication of ICs.

#### 10.3.1 Basic Diode Logic

To start with Diode logic let us first define the two voltage levels corresponding to two logic levels.

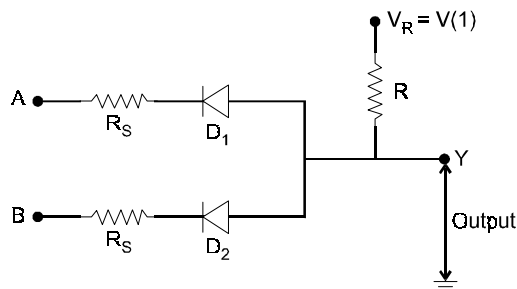
Let  $V(1) = \text{Voltage corresponding to logic 1} = +V_{CC}$   
 $V(0) = \text{Voltage corresponding to logic 0} = \text{Gnd or OV}$

Let all the diodes are silicon diode and have the drop of 0.7 V when in conduction. In all the cases we measure the output voltage with respect to ground, assuming that forward resistance of diode ( $R_f$ ) is very small.

#### DIODE AND GATE

A two terminal diode AND gate is shown in Fig. 10.32 and we assume that the resistance  $R \gg R_S$  so that drop across  $R_S$  can be neglected in calculation.

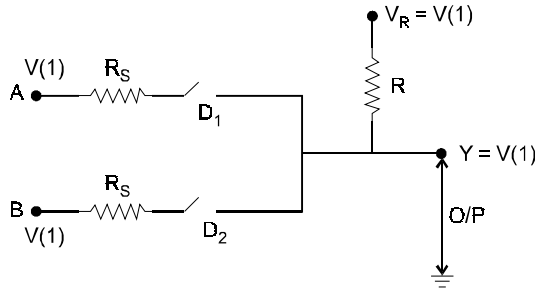
$V_R = \text{Reference Voltage}$   
 $R_S = \text{Source Resistance}$   
 $Y = \text{Output of Gate}$   
 $D_1, D_2 = \text{Identical Diodes}$   
 $A_1 B = \text{Logic inputs}$   
 $R = \text{Pullup Resistor}$



**Fig. 10.32** Diode and Gate

Resistor R is called pullup resistor because in some sense it pulls the output from logic low to logic high.

As evident from the circuit, status of inputs A and B will make the diode conducting (ON) or non conducting (OFF). For simplicity let us first consider that—both the inputs are at logic 1 *i.e.*,  $A = B = V(1)$ . We get Figure 10.32 reduced as in Figure 10.33(a).

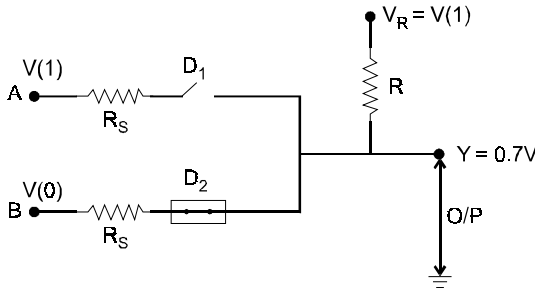


**Fig. 10.33 (a)** And Gate of Fig. 10.32 at  $A = B = V(1)$

Both the diodes are reverse biased and represents open circuit as shown. Since there is no path for current flow through resistor it will not have any drop across it and  $V_R = V(1)$  appears at output. This is obvious because when all inputs to the AND gate are high then output is high. Thus

when  $A = B = V(1) \quad Y = V(1)$

**when  $A = V(1)$  and  $B = V(0)$** , the diode  $D_2$  conducts and  $D_1$  remains in OFF condition. The equivalent circuit in this case is shown below in Figure 10.33(b).



**Fig. 10.33 (b)** And Gate of Fig. 10.32 at  $A = V(1)$  and  $B = V(0)$

Since a current flows through R, through Diode  $D_2$  towards ground ( $V(0)$ ) we get drop across diode  $D_2$  at output so when  $A = V(1)$  and  $B = V(0)$   $Y = V_0 = 0.7 \text{ V} = \text{LOW}$  which confirms that if any of the input of AND gate goes low output goes low.

**When  $A = V(0)$  and  $B = V(1)$**  In this case  $D_1$  conducts and  $D_2$  goes OFF and output again becomes the drop across diode *i.e.*,  $Y = 0.7 \text{ V}$ .

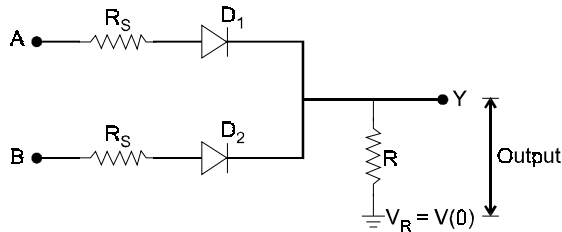
**When  $A = V(0)$  and  $B = V(0)$**  In this case both  $D_1$  and  $D_2$  conducts having a drop of  $0.7 \text{ V}$ . Again this drop appears at output. So the output  $Y = 0.7 \text{ V}$ . All these value of outputs along with logic state is summarized in table 10.4 which confirms AND gate. Note that we define  $0.7 \text{ V}$  as logic low at the output where as  $0 \text{ V}$  as logic low at the input. For logic 1 voltages at input output are same.

**Table 10.4 : Input/output table of AND GATE of figure 10.32**

Inputs		Output	
A	B	Y	Logic State
V(0)	V(0)	0.7 V	LOW
V(0)	V(1)	0.7 V	LOW
V(1)	V(0)	0.7 V	LOW
V(1)	V(1)	V(1)	HIGH

**Diode OR Gate**

A two terminal OR Gate is shown in Fig. 10.34. All the conventions and assumptions are same as Diode and gate discussed previously. Here the position of  $V_R$  is shown below the output Y, just for convenience. It is equally well if you show it above Y with same connections.



**Fig. 10.34** Two terminal diode or gate.

As evident from figure the drop across resistor R will be the output voltage.

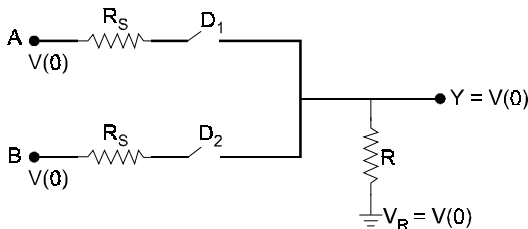
**When both inputs are low** i.e., when  $A = B = V(0)$  both the diodes are OFF and represents switch in open condition as shown in Fig. 10.35 (a). So no current flows through R and the output  $Y = V_R = V(0) = \text{LOW}$ . This is consistent with the definition of OR gate.

**When  $A = V(1)$  and  $B = V(0)$** ,  $D_1$  conducts and  $D_2$  remains OFF as shown in Fig. 10.35(b). Thus a current flows through resistor R starting from input A through  $D_1$  through R to ground. Now applying KVL to conducting path we get

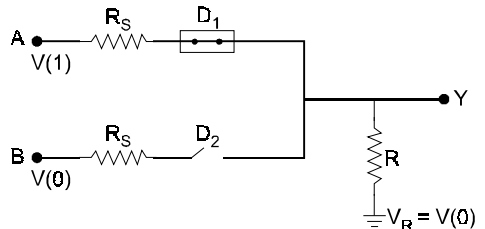
$$V(1) - V_D - Y = Y \quad \text{neglecting } R_S$$

So  $Y = V(1) - V_D$

or  $Y = V(1) - 0.7V = \text{HIGH}$



**Fig. 10.35 (a)** Diode or Gate with  $A = B = V(0)$



**Fig. 10.35 (b)** Diode or Gate with  $A = V(1)$  and  $B = V(0)$

In digital circuits we take  $V(1) = 5 \text{ V}$  generally, so saying  $Y = V(1) - 0.7 \text{ V}$  means  $y = 4.3 \text{ V}$  which is good enough to regard as HIGH output.

Similar situation occurs when  $A = V(0)$  and  $B = V(1)$  and when  $A = V(1)$  and  $B = V(1)$ .

Thus the gate shown in Fig. 10.34 behaves as OR gate. Readers are advised to find out the input/output table for this gate by taking  $V(1) = 5V$ .

### 10.3.2 Resistor Transistor Logic (RTL)

The RTL family is now a days historical but its study and analysis can give good understanding and enough motivation for the other logic families. The Basic RTL gate is a NOR gate as shown in Fig. 10.36. For the logic operation we first assume that logic low input is sufficient to switch off the transistor and logic high is sufficient to switch ON the transistor.

**When both the inputs are high** *i.e.*,  $A = B = V(1)$ , both  $Q_1$  and  $Q_2$  goes to ON state *i.e.*, both enters into saturation. Thus the voltage at collectors is nothing but  $V_{CEsat}$ . So output  $Y = V_{CEsat}$ . Hence, when

$$A = B = V(1) \qquad Y = V_{CEsat} = 0.2 \text{ V} \\ \cong V(0) = \text{LOW}$$

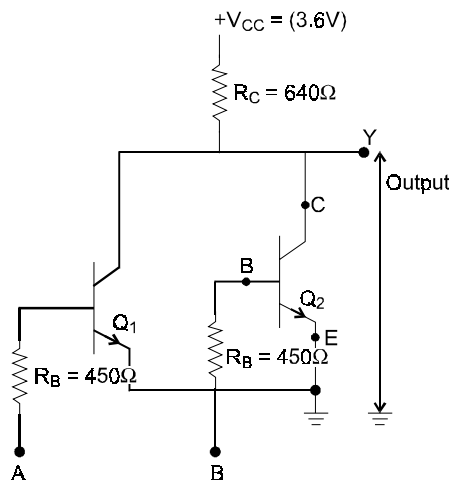
**When both the inputs are low** *i.e.*,  $A = B = V(0)$ , both  $Q_1$  and  $Q_2$  goes to OFF state. Thus no current flow through  $R_C$  and drop across  $R_C$  is zero. So whole  $+V_{CC}$  will appear at output *i.e.*,  $Y = +V_{CC} = 3.6 \text{ V} = \text{HIGH}$ .

Thus when  $A = B = V(0) \qquad Y = +V_{CC} = \text{HIGH}$

**When only one input goes high** *i.e.*,  $A = V(1)$  and  $B = V(0)$  or when  $A = V(0)$  and  $B = V(1)$ , the transistor feeded with high input conducts causing a current to flow through ON transistor. Consequently the transistor enters into saturation. Thus the output voltage becomes  $V_{CEsat}$ . So,

when  $A = V(1) \ B = V(0) \qquad Y = V_{CEsat} = \text{LOW}$   
 $A = V(0) \ B = V(1) \qquad Y = V_{CEsat} = \text{LOW}$

From the above discussion it is clear that output of circuit shown in Fig. 10.36 represents a NOR gate. It is because when all the inputs are low, only then output is high otherwise the output is low.



**Fig. 10.36** The RTL NOR Gate.

To find out the FAN OUT, NOISE MARGINS etc. let us consider the loading of RTL gate as shown in Fig. 10.37. For the ease of analysis we connect the input B of all the load gates are connected to logic LOW and all the A inputs connected to the output of Driver gate.

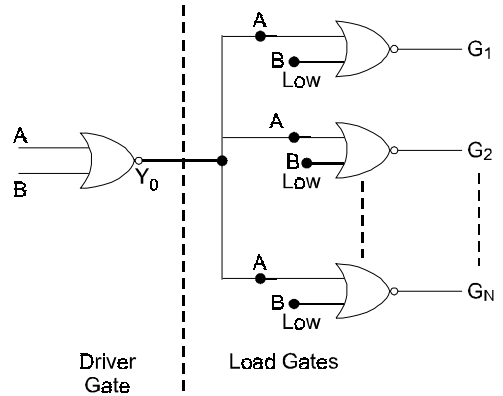
Note that all the load gates and the driver gate are identical with all the parameter assumed same.

Since all the B inputs are low, transistor  $Q_2$  (of Fig. 10.36) represents switch in open condition. So transistor  $Q_2$  may not be drawn in this condition as shown in Figure 10.38(a). To understand the idea let us consider an example.

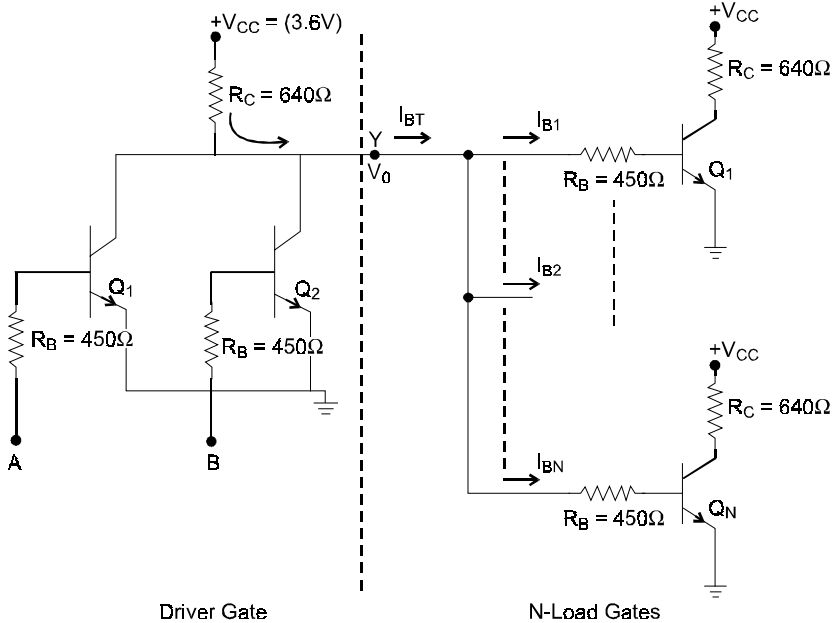
**Example 10.4.** Find out the Noise margin and Fan out of the RTL NOR gate shown in Fig. 10.38(a). Given the transistors have  $h_{FE} = 7$ .

**Solution.** To begin with we take the assumptions and connection explained for Figure 10.37.

To find out FAN OUT let both the inputs of Driver gate are low *i.e.*,  $A = B = V(0)$ . Then both the  $Q_1$  and  $Q_2$  goes to cut off and output of driver goes HIGH. Thus all the load gates are driven by this HIGH output and consequently all the load transistors are driven into saturation. In this situation a current  $I_{BT}$  will be drawn from the  $+V_{CC}$  of driver side, as shown in Fig. 10.38(a). Since all load transistors are identical, they will draw equal amount of base current. Thus,



**Fig. 10.37.** An RTL Gate Driving N-Load Gates.



**Fig. 10.38 (a)** RTL under loading.



and since

$$I_{BT} = I_{B1} + I_{B2} + \dots + I_{BN}$$

we get

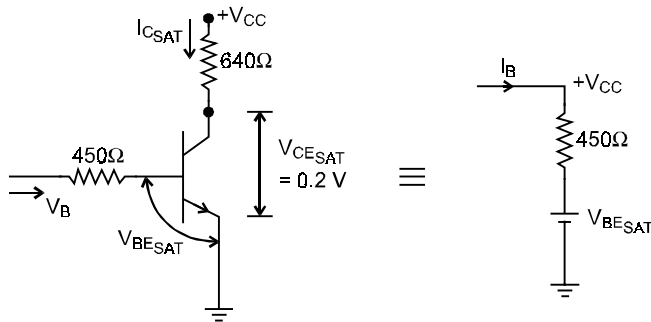
$$I_{B1} = I_{B2} \dots = I_{BN}$$

or

$$I_{BT} = N \cdot I_{B1}$$

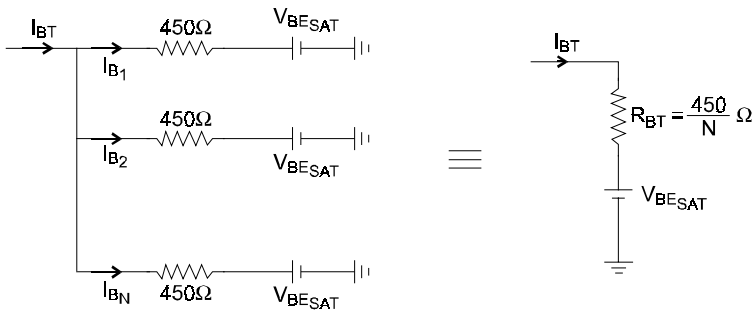
$$I_{B1} = \frac{I_{BT}}{N} \quad \dots(10.14)$$

When a load transistor is driven into saturation it can be represented as shown in Fig. 10.38(b).



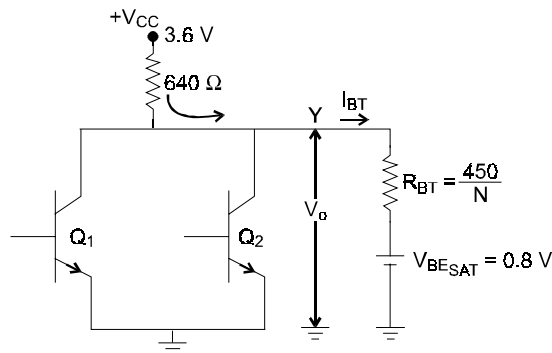
**Fig. 10.38 (b)** Equivalent representation of transistor in saturation

Similarly if all the load transistors are in saturation we get equivalent figure as shown in Fig. 10.38(c).



**Fig. 10.38 (c)** Equivalent circuit for N-load gates in saturation.

Thus we can redraw the Fig. 10.38(a) when both A and B inputs of driver gates are low, as shown in Fig. 10.38(d).



**Fig. 10.38 (d)** Equivalent circuit

Thus the current  $I_{BT}$  can be calculated as

$$I_{BT} = \frac{V_{CC} - V_{BEsat}}{640 + \frac{450}{N}}$$

Now by using equation (10.14) base current to individual transistor can be given as

$$I_{B1} = \frac{I_{BT}}{N}$$

So

$$I_{B1} = \frac{1}{N} \cdot \frac{V_{CC} - V_{BEsat}}{640 + \frac{450}{N}} = \frac{3.6 - 0.8 \text{ V}}{640 N + 450}$$

or

$$I_{B1} = \frac{2.8}{640 N + 450} \quad \dots(10.15)$$

For each of the load transistor in saturation, the current  $I_{CSAT}$  can be calculated as

$$I_{CSAT} = \frac{V_{CC} - V_{CESAT}}{R_C} = \frac{3.6 - 0.2}{640} \quad (\text{see Fig. 10.38(b)})$$

or

$$I_{CSAT} = 5.31 \text{ mA}$$

Now the current  $I_{B1}$  of load gate  $G_1$  as calculated in equation 10.15 must be sufficient to drive the transistor in saturation. In saturation the  $I_{B1}$  and  $I_{CSAT}$  must follow the relation.

$$I_B \cdot h_{fe} \geq I_{CSAT}$$

*i.e.*,

$$\frac{2.8 \times 7}{640 N + 450} \geq 5.31 \times 10^{-3} \quad (\text{given } h_{fe} = 7)$$

for worst case

$$\frac{19.6}{640 N + 450} = 5.31 \times 10^{-3}$$

or

$$19.6 = 3.3984 N + 2.3895$$

or

$$N = 5.064$$

thus taking  $N = 5$  will always drive transistors into saturation. So FAN OUT = 5.

With  $N = 5$  we get  $I_{B1} = 0.767 \text{ mA}$

and

$$I_{BT} = 3.835 \text{ mA}$$

To calculate noise margin let us first find out the output voltages at two states.

When output is in low state  $V_o = V_{CEsat} = 0.2 \text{ V}$ . When this output is applied to load transistor, it will go into cut off. If noise causes it to become  $0.5 \text{ V}$  ( $V_v$  of transistor) then load transistors will conduct and circuit malfunctions. Thus low state noise margin.

$$\Delta 0 = V_v - V_{CEsat} = 0.5 - 0.2 = 0.3 \text{ V}$$

*i.e.*,

$$\Delta 0 = 0.3 \text{ V}$$

when the output of driver goes high all the load transistors goes to saturation and a current  $I_{BT}$  flows. Thus the output voltage will be

$$V_o = V_{R_{BT}} + V_{BE_{sat}} \quad (\text{see Fig. 10.38(d)})$$

$$V_{R_{BT}} = I_{BT} \cdot R_{BT} = 3.835 \times 10^{-3} \times 90 \quad \text{for } N = 5$$

so 
$$V_{R_{BT}} = 0.345 \text{ V} \quad \text{when } N = 5$$

thus 
$$V_o = V_{R_{BT}} + V_{BE_{sat}} = 0.345 + 0.8$$

$$V_o = 1.145 \text{ V} \quad \text{when } N = 5$$

So if this voltage is applied from driver to load, all the load transistors will enter into saturation. Since  $N = 5$  was calculated for  $h_{FE} I_B = I_{C_{sat}}$ , that means any voltage less than 1.145 V may not be able to drive the load transistors into saturation. So we may say that if it is reduced to 1.045 V load transistor may not be in saturation. So high state noise margin  $\Delta 1 \cong 0.1 \text{ V}$ . Infact the high state noise margin depends upon the number of gates being driven. If lesser number of load gates are connected the noise margin would be improved.

The *Propagation Delay* is also affected by number of load gates. When output of driver is LOW all the load transistors are in cutoff and their base to emitter junction appears as capacitor. Since  $N$  load transistors are there,  $N$  such capacitor appears in parallel. If capacitance offered by one load transistor is  $C_p$ , then total capacitance would be

$$C_{TOTAL} = C_i + C_i + \dots$$

$$C_{TOTAL} = NC_i \quad \text{when driver output is LOW}$$

when driver output goes to high from low this capacitor ( $C_{TOTAL}$ ) must be charged to HIGH voltage by a time constant.

$$\tau = R_{eq} \cdot C_{TOTAL}$$

where 
$$R_{eq} = 640 + \frac{450}{N} \quad (\text{see Figure 10.38(d)})$$

so 
$$\tau = \left( 640 + \frac{450}{N} \right) \cdot NC_i$$

$$\text{finally } \tau = (640 N + 450) C_i \quad \dots(10.16)$$

We know that smaller the  $\tau$  faster is the charging of capacitor and hence smaller is the propagation delay.

By equation (10.16) it is clear that larger the value of  $N$  larger the value of  $\tau$  and hence propagation delay is large.

The collector resistor  $R_C$  is called pull up resistor because it pulls the output of gate from low to high state. It is because when transistor is in LOW state the output capacitance  $C_{TOTAL}$  is discharged and for LOW to HIGH transition the output capacitance  $C_{TOTAL}$  must be charged to the Logic 1 voltage, the resistor  $R_C$  provides the charging current path. Thus the name pull up resistor. Since  $R_C$  is passive element, this is called **passive pullup**.

Also note that in Fig. 10.38(a) when driver output is HIGH a current  $I_{BT}$  is supplied from driver side to the load side. Thus the logic is called as the **current source logic**, because the driver is sourcing the current to load.

### 10.3.3 Direct Coupled Transistor Logic (DCTL)

The Basic DCTL gate is a NOR gate shown in Fig. 10.39. Comparing the Fig. 10.39 with Fig. 10.38(a) reveals the fact that DCTL is a RTL gate with no base resistor. Infact output of driver is directly connected to load and hence the name DCTL. Naturally in this case LOW output voltage is  $V_{CE_{sat}} = 0.2 \text{ V}$  and the HIGH output voltage is  $V_{BE_{sat}} = 0.8 \text{ V}$ . But the output logic swing is very small ( $0.8 - 0.2 = 0.6 \text{ V}$ ). Thus the noise margin is poor.

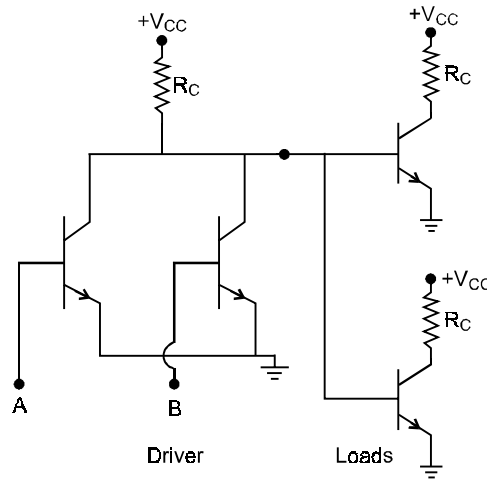


Fig. 10.39 Basic DCTL NOR Gate.

The DCTL gates are simpler than RTL but suffers from a problem called *current hogging*. It occurs when driver output is logic 1. The input characteristic of all the transistors are assumed to be same in theory. But in practice it is not true. The input characteristics of transistor of same batch differ from each other causing variation in  $V_{BE_{sat}}$ . Thus if  $V_{BE_{sat}} = 0.8 \text{ V}$  for one transistor another may have  $0.798 \text{ V}$ , some other may have  $0.796 \text{ V}$ . When Driver's output is logic 1, the transistor having lowest value of  $V_{BE_{sat}}$  (on the load side) will enter first into saturation and will take all the current supplied by the driver side. Thus the other transistors may not get sufficient amount of current to enter into saturation. This is called **current hogging**.

### 10.3.4 Diode Transistor Logic (DTL)

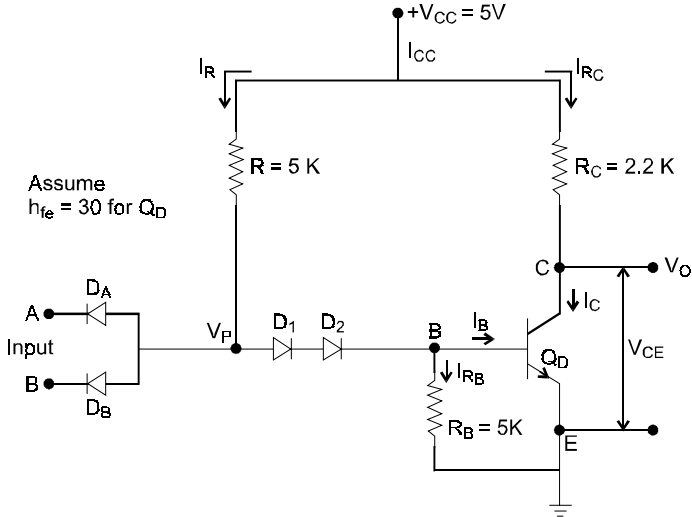
The basic DTL gate is a NAND gate shown in Fig. 10.40. The two inputs of the gate are applied through the diodes  $D_A$  and  $D_B$  which are transmitted to the base of transistor  $Q_D$  through diodes  $D_1$  and  $D_2$ . Output of the gate is measured at the collector of  $Q_D$ .

A close inspection of the circuit reveals that the input diodes along with resistor  $R$  forms a diode AND gate whose output is available at point P. The transistor acts as inverter, thus the output of diode AND gate is inverted to give output of DTL gate as NAND gate.

#### Circuit Operation

Before describing the operation we first define value of  $V(0)$  and  $V(1)$ . The output of gate is said to be HIGH when  $Q_D$  is OFF and  $V_O = V_{CC} = +5\text{V}$ . Thus  $V(1) = +5\text{V}$ . The output of gate is LOW when  $Q_D$  is in saturation so  $V_O = V_{CE_{sat}} = 0.2 \text{ V}$ . Thus  $V(0) = 0.2\text{V}$ . From the

figure it is evident that voltage at point P,  $V_P$  is responsible to drive the transistor in cutoff or in saturation. Let us calculate the voltage at point P, required to turn ON the transistor;



**Fig. 10.40** DTL NAND Gate.

$$V_P(\text{ON}) = (V_v)_{D_1} + (V_v)_{D_2} + (V_v)_{Q_D}$$

but  $V_v = 0.6 \text{ V}$  for Diodes

and  $V_v = 0.5 \text{ V}$  for BJTs

so  $V_P(\text{ON}) = 0.6 + 0.6 + 0.5$

so  $V_P(\text{ON}) = 1.7 \text{ V}$  ... (10.17)

**when  $A = V(0)$   $B = V(1)$** , the diode  $D_A$  is conducting and  $D_B$  is cutoff. So voltage at point P is one diode drop above the  $V(0)$ . So in this case

$$V_P = V(0) + V_{D_A} \text{ and } V_{D_A} = \text{drop across conducting diode } D_A = 0.7$$

thus  $V_P = 0.2 + 0.7$   $V(0) = 0.2\text{V}$  as defined earlier

$$V_P = 0.9\text{V} \text{ ... (10.18)}$$

Since  $V_P < V_P(\text{ON})$ , transistor  $Q_D$  is in OFF state and the output at the gate is  $+V_{CC}$ , i.e.,  $V(1)$ .

Same will be the output when  $A = V(1)$  and  $B = V(0)$  and when  $A = B = V(0)$ .

**when  $A = B = V(1)$** , the diodes  $D_A$  and  $D_B$  will be in OFF state and current  $I_R$  flows through diodes  $D_1$  and  $D_2$ , through base of  $Q_D$ . Consequently the transistor enters in saturation and the output becomes  $V_0 = V_{CE\text{sat}} = 0.2 \text{ V} = V(0)$ . Let us find out the voltage at point P in this case

$$V_P = V_{D_1} + V_{D_2} + V_{BE\text{Sat}}$$

where  $V_{D_1}$  and  $V_{D_2} = 0.7 \text{ V}$  drop across conducting diodes  $D_1$  and  $D_2$  and  $V_{BE\text{Sat}} = 0.8 \text{ V}$

thus  $V_P = 0.7 + 0.7 + 0.8$

so  $V_P = 2.2 \text{ V}$  ... (10.19)

since  $V_P > V_P(\text{ON})$ ,  $Q_D$  is in ON condition and the output is low.

From the above discussion it is clear that output of gate is **low** only when both  $A = B = V(1)$ , otherwise the output is HIGH. Thus the circuit behaves as NAND gates.

**Example 10.5.** Find out the different currents and voltages for the DTL NAND gate shown in Fig. 10.40 in different output states. Also find out the average power dissipation in the gate.

**Solution.** Since the output of gate can be **high** or **low**, our calculation will be different for these two conditions. For simplicity we first consider the output in **high** state.

### 1. When output is high

(a) Inputs  $A = B = V(0)$  or  $A = V(0) B = V(1)$  or  $A = V(1) B = V(0)$

(b) Transistor  $Q_D$  in cutoff so  $V_O = +V_{CC} = 5V = V(1)$

(c) Since  $Q_D$  is OFF  $I_{RC} = 0$ ,  $I_C = 0$ ,  $I_B = 0$ ,  $I_{RB} = 0$ .

(d) Voltage at point P  $V_P = 0.9 V$  by (10.18)

$$(e) \text{ Current } I_R = \frac{V_{CC} - V_P}{R} = \frac{5 - 0.9}{5 \times 10^{-3}} = 0.82 \times 10^{-3}$$

*i.e.*,  $I_R = 0.82 \text{ mA}$  ... (10.21)

(f) Drop across resistor R is  $V_R = V_{CC} - V_P = 4.1V$

(g) Current drawn from supply in HIGH state  $I_{CC}(1)$

$$I_{CC}(1) = I_R + I_{RC} = 0.82 \times 10^{-3} + 0$$

So  $I_{CC}(1) = 0.82 \text{ mA}$  ... (10.21)

(h) HIGH state power dissipation  $P(1) = V_{CC} \cdot I_{CC}(1)$

$$P(1) = 4.1 \text{ mW} \quad \dots (10.22)$$

### (2) When output is low

(a) Inputs  $A = B = V(1) = +5V$

(b) Transistor  $Q_D$  in ON state so  $V_O = V_{CEsat} = 0.2 V$

(c) Voltage at point P  $V_P = 2.2 V$  by 10.19

(d) Drop across resistor R,  $V_R = V_{CC} - V_P = 5 - 2.2$

$$\text{So } V_R = 2.8 V$$

(e) The collector current in saturation  $I_{Csat}$

$$I_{Csat} = I_{RC} = \frac{V_{CC} - V_{CEsat}}{R_C} = \frac{5 - 0.2}{2.2 \times 10^3}$$

so  $I_{RC} = I_{Csat} = 2.18 \text{ mA}$  ... (10.23)

(f) Current through resistor R,  $I_R = \frac{V_{CC} - V_P}{R} = \frac{V_R}{R} = \frac{2.8}{5 \times 10^3}$

$$\text{so } I_R = 0.56 \text{ mA} \quad \dots (10.24)$$

(g) Current through resistor  $R_B$ ,  $I_{RB} = \frac{V_{RB}}{R_B}$

but the drop across  $R_B$  is same as  $V_{BEsat}$ .

$$\text{so } V_{RB} = V_{BEsat} = 0.8 V$$

= drop across base to emitter junction in saturation.

thus  $I_{RB} = \frac{0.8 \text{ V}}{5 \times 10^3}$

so  $I_{RB} = 0.16 \text{ mA}$

(h) The current  $I_B = I_R - I_{RB}$  by KCL at point B

so  $I_B = 0.56 - 0.16$

or  $I_B = 0.4 \text{ mA}$

(i) Check for transistor in saturation. For transistor to be in saturation it must follow

$$h_{FE} I_B \geq I_{C_{sat}}$$

given  $h_{FE} = 30$  for  $Q_D$

so  $h_{FE} I_B = 30 \times 0.4 \times 10^{-3}$

or  $h_{FE} I_B = 12 \text{ mA}$  ... (10.25)

Since this is greater than  $I_{C_{sat}} = 2.18 \text{ mA}$ , the transistor is in saturation.

(j) LOW state supply current  $I_{CC}(0) = I_R + I_{RC}$   
 $= 0.56 \text{ mA} + 2.18 \text{ mA}$

so  $I_{CC}(0) = 2.74 \text{ mA}$  ... (10.26)

(k) LOW state power dissipation  $P(0) = V_{CC} \cdot I_{CC}(0)$   
 $= 5 \times 2.74 \times 10^{-3}$

so  $P(0) = 13.7 \text{ mW}$  ... (10.27)

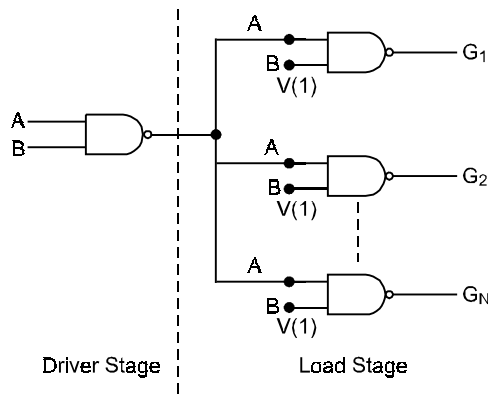
To find out average power dissipation we assume 50% duty cycle at the output i.e., occurrence of LOW and HIGH state is same then, the average power dissipation

$$P_{avg} = \frac{P(1) + P(0)}{2} = \frac{4.1 \text{ mW} + 13.7 \text{ mW}}{2}$$

so  $P_{avg} = 8.9 \text{ mW}$  ... (10.28)

**Loading of DTL Gate**

In the loading of DTL gate we are concerned with finding out FANOUT and noise margin for the DTL NAND gate. We assume that all the load gates are identical and input B of load gates are connected to logic 1, for simplicity. The logic diagram is shown in Fig. 10.41.



**Fig. 10.41** ADTL NAND Gate Driving N Similar Gate.

The equivalent circuit diagram for loading of DTL gate is shown in Fig. 10.42. As evident from Fig. 10.42 when the output  $V_0$  of driver is HIGH then all the  $D_A$  diodes of load gates are reverse biased so no current flows either from Driver to Load or from Load to Driver.

When output of Driver is low then  $D_A$  diodes of load are forward biased and a current  $I_L$  flows from each load gate towards driver gate. Since all load gates identical so they will draw same  $I_L$  (which is  $I_R$  as shown in Fig. 10.42) and hence total load current from load stage to driver state is

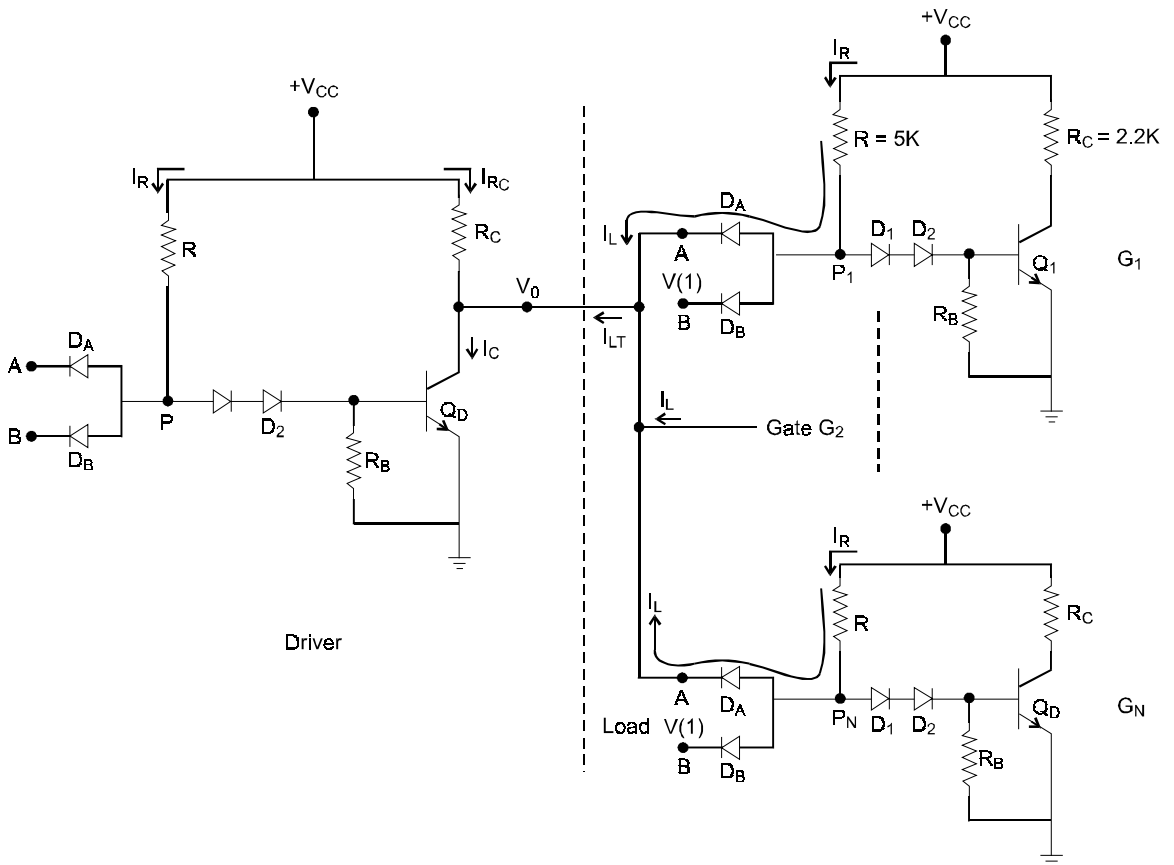
$$I_{LT} = N \cdot I_L \quad \dots(10.29)$$

Also note that when diode  $D_A$  of load gates conduct the output of all load gates are HIGH. In this situation

$$I_L = I_R = 0.82 \text{ mA} \quad \text{see equation (10.20)}$$

$$\text{thus total load current } I_{LT} = N \cdot 0.82 \text{ mA} \quad \dots(10.30)$$

The current  $I_L$  supplied by individual load gate is called *standard load*.



**Fig. 10.42** DTL Gate driving N identical gates. Input B of each load gate is connected to LOGIC 1

Thus when output of Driver is LOW the current  $I_{LT}$  is supplied by load stage to  $Q_D$  in addition to the current  $I_{RC}$ . Thus under loading



$$I_C = I_{RC} + I_{LT}$$

so 
$$I_C = 2.18 \text{ mA} + 0.82 \text{ NmA by equations (10.23) \& (10.29)}$$
 ... (10.31)

But the current  $I_C$  given by equation 10.31 must satisfy  $h_{FE} I_B \geq I_{C_{sat}}$  so that  $Q_D$  remains in saturation and output of driver remains low. Thus we use this fact to determine FAN OUT. So

$$I_{C_{sat}} \leq h_{FE} I_B$$

but  $h_{FE} I_B = 12 \text{ mA}$  when  $Q_D$  is ON, by equation (10.25) then equation (10.31)

$$2.18 + 0.82 N \leq 12$$

Calculating for worst case  $2.18 + 0.82 N = 12$

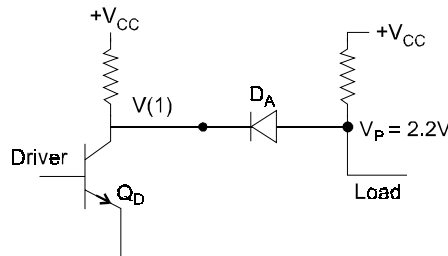
or 
$$N \cong 12$$

but for  $I_{C_{sat}} < h_{FE} I_B$  we must have  $N < 12$ .

so let us select 
$$N = 10$$
 ... (10.32)

**Noise, Margin**

When output of  $Q_D$  is in HIGH state the Diode  $D_A$  of load gates are reverse biased as shown in Fig. 10.43 since  $V_P = 2.2 \text{ V}$  when input diodes are reverse biased, the amount by which  $D_A$  is reverse biased  $= V(1) - V_P = 5 - 2.2 = 2.8 \text{ V}$



**Fig. 10.43** Circuit situation when driver output is high.

Since  $V_v = 0.6V$  a negative spike of at least  $2.8 + 0.6 = 3.4 \text{ V}$  present at the input of load gate will cause malfunction. Thus  $\Delta 0 = 3.4 \text{ V}$ . It is called low state noise margin because HIGH output of driver causes a LOW output of load.

When output of  $Q_D$  is LOW diode  $D_A$  of load is forward biased and voltage  $V_P$  on load side will  $0.9 \text{ V}$ , by equation 10.18. And High state noise margin is given as

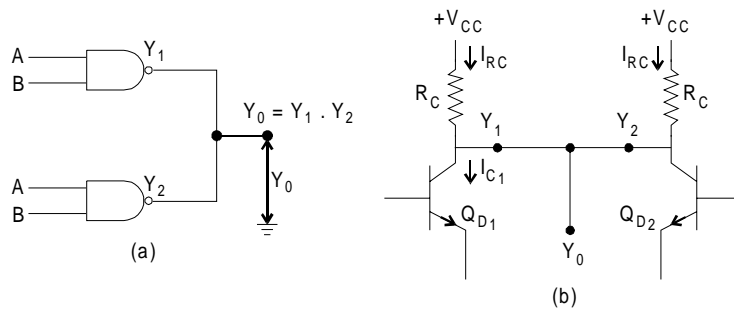
$$\Delta 1 = V_P(\text{ON}) - V_P = 1.7 \text{ V} - 0.9 = 0.8 \text{ V}$$

Thus  $\Delta 1 = 0.8 \text{ V}$ . Thus a positive noise spike of at least  $0.8 \text{ V}$  will cause circuit malfunction.

**Propagation Delay** is approximated by the capacitance present at the output of  $Q_D$  due to load gates. When  $Q_D$  turns ON these output capacitance discharge quickly through  $Q_D$ , thus giving low turn ON delays. But when  $Q_D$  goes OFF the output capacitance has to charge through the register  $R_C$  which takes large time. Thus giving large turn OFF delay. Again register  $R_C$  is **pullup resistor** and this type of pull up is called **passive pullup** as  $R_C$  is passive element.

**Current sink logic** DTL is called current sink-logic because when  $Q_D$  is in ON state there flows a current from Load stage to driver stage, and the  $Q_D$  acts as current sink for these load currents. When  $Q_D$  is in OFF state output is HIGH and input diodes of load gates are reverse biased. Practically there flows a very small amount of current from driver to load side. This current is equal to reverse saturation current of diodes. The driver is said to be sourcing this small current to load. Since sourcing of current is very-very small compared to sinking the overall logic is called current sink logic.

**Wired Logic** when outputs of gates are tied together through a wire, as shown in Fig. 10.44 additional logics performed. These are called wire-logic or implied logic consider Fig. 10.44(a) to find the wired logic.



**Fig. 10.44** Wired-ANDing of DTL NAND gate

when

$$Y_1 = V(0) \text{ and } Y_2 = V(0) \text{ then } Y_0 = V(0)$$

$$Y_1 = V(1) \text{ and } Y_2 = V(1) \text{ then } Y_0 = V(1)$$

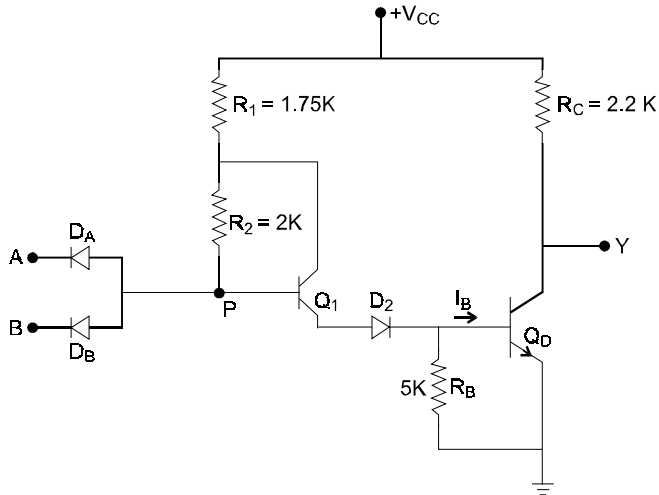
when  $Y_1 = V(1)$  and  $Y_2 = V(0)$ , then the current will flow from  $Y_1$  to  $Y_2$ . Thus if  $Y_0$  is measured with respect to ground we get  $Y_0 = Y_2 = V(0)$ . Thus when one or both of the gate output ( $Y_1$  or  $Y_2$ ) goes low, the wired output is low. Hence the above wired logic is wired-AND logic. The same wired output can be obtained by Fig. 10.44(b). But the wired logic increases the low state power dissipation as if one or both transistors of Fig. 10.44(b) conduct then collector resistance becomes  $R_C/2$ . Thus more current is drawn.

Also the wired logic in DTL decreases to FAN OUT. If only  $Q_{D1}$  (in Fig. 10.44(b)) conducts then the collector current  $I_{C1}$  through  $Q_{D1}$  is  $I_{C1} = I_{RC} + I_{RC} = 2I_{RC}$ . Thus the collector current is doubled than that of normal case. Since  $h_{FE}I_B$  remains constant and collector current is increased, the effective FAN OUT is reduced.

**Modified DTL Nand Gate**

In the analysis of DTL NAND gate we have calculated the FAN OUT by using the equation  $h_{FE}I_B \geq I_{C_{sat}}$ . If some how  $I_B$  is increased the value of  $h_{FE}I_B$  increases and the equation  $h_{FE}I_B \geq I_{C_{sat}}$  can be satisfied for larger no. of load gates. Thus the FAN OUT can be improved. The base current  $I_B$  can be increased if the diode  $D_1$  of Fig. 10.40 is replaced by a transistor  $Q_1$  as shown in Fig. 10.45. The resistor  $R$  is splitted to provide biasing of transistor  $Q_1$ . When  $Q_1$  is conducting the base current  $I_B$  of transistor  $Q_D$  increases and thus value of  $h_{FE}I_B$  increases significantly. As a consequence the FAN OUT of circuit shown in Fig. 10.45 increases significantly as compared to the circuit shown in Fig. 10.40.

In the last we advise the readers to go through the DTL gate once again from beginning as these understandings would be needed when we will discuss the TTL. This is the reason why we have presented DTL exhaustively.

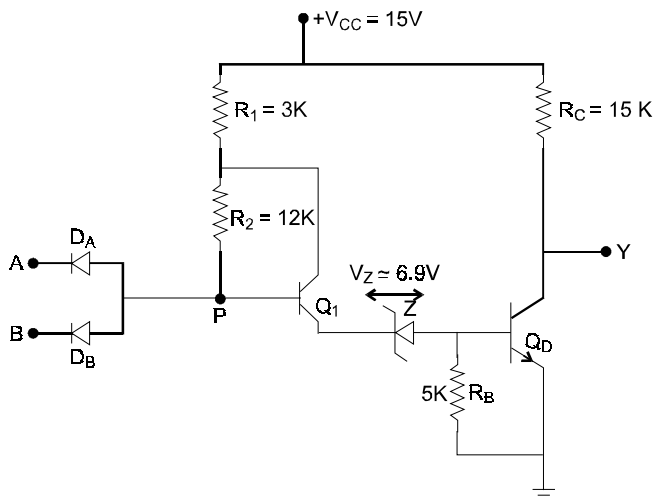


**Fig. 10.45** Modified DTL NAND gate

### 10.3.5 High Threshold Logic (HTL)

In the industrial environments the noise levels are high. These are mainly due to high rating relays, various motors and variety of ON/OFF controls. In general the logic families like RTL, DTL do not offer such a noise margin so that higher noise levels can be tolerated. Infact due to lower noise margin the circuit will malfunction most of the time.

To suit the industrial application, modified DTL gate, shown in Fig. 10.45, is redesigned with higher  $+V_{CC}$  supply and increased value of resistance and diode  $D_2$  is replaced by a zener diode, as shown in Fig. 10.46. The overall circuit provides same current level as that of DTL NAND gate. As evident from the circuit, the noise margins are dramatically improved. But presence of larger resistances severely affects the switching speed, causing the propagation delay in hundreds of nano seconds.



**Fig. 10.46** Basic HTL NAND GATE

### 10.3.6 Transistor Transistor Logic (TTL)

The DTL NAND gate shown in fig. 10.40 has simple and stable circuit design but has the larger propagation delay due to which it was obsolete. The main cause of large propagation delays are diodes present in DTL. There were three main reasons for their slower speed.

1. Inputs are taken through diodes which are slower devices.
2. When output goes to HIGH, the output capacitance charges through collector resistance  $R_C$  which is slow process.
3. When output goes to HIGH transistor  $Q_D$  goes to cut off from saturation  $D_1$  and  $D_2$  are non conducting and excess charge stored in base must be removed through  $R_B$ , which is a slow mechanism.

All these can be avoided if the diodes  $D_A$ ,  $D_B$ ,  $D_1$  and  $D_2$  of DTL gate are replaced by transistor. The modified circuit, called TTL shown in Fig. 10.47, offers a significant improvement in propagation delay. A close inspection of the circuit and its comparison with Fig. 10.40 reveals that it is a DTL gate in which diodes  $D_A$  and  $D_B$  are replaced two E-B junctions of  $Q_1$ , diode  $D_1$  is replaced by B-C junction of  $Q_1$  and diode  $D_2$  is replaced by B-E junction of transistor  $Q_2$ . Thus all the discussion of DTL would be valid with slight modifications. The capacitor  $C_O$  is output capacitance that must be charged to the logic value of output. Before proceeding further let us assume that  $V(0) = 0.2V$  ( $V_{CEsat}$ ) and  $V(1) = 5V$  ( $+V_{CC}$ ).

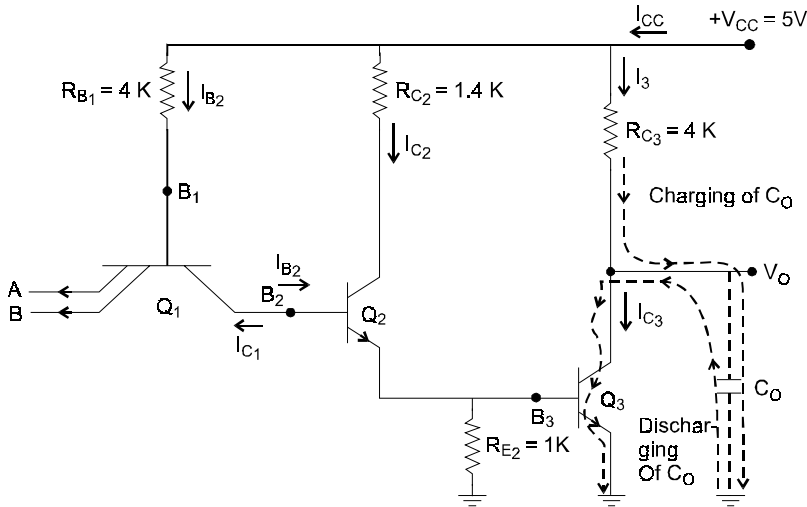


Fig. 10.47 TTL NAND Gate with Passive Pullup.

Let us define a voltage  $V_{B1}(ON)$ , that is minimum voltage required at Base of  $Q_1$  to forward bias B-C junction of  $Q_1$  and to drive  $Q_2$  to  $Q_3$  in conduction. It is given as

$$\begin{aligned}
 V_{B1}(ON) &= (V_{BC})_{CUTIN} + (V_v)_{Q_2} + (V_v)_{Q_3} \\
 &= 0.6 + 0.5 + 0.5
 \end{aligned}$$

so  $V_{B1}(ON) = 1.6 V$  ... (10.33)

**When either or both the inputs are LOW** The B-E junction of  $Q_1$  conducts, thus the voltage at base  $B_1$  is

$$V_{B1} = V(0) + V_{BEACTIVE} = 0.2 + 0.7$$

$$\text{so } V_{B_1} = 0.9 \text{ V} \quad \dots(10.34)$$

Since  $V_{B_1} < V_{B_1}(\text{ON})$ , transistor  $Q_3$  is OFF and the output voltage  $V_o = +V_{CC} = V(1)$ .

**When both the inputs are HIGH** The B-E junction of  $Q_1$  is reverse biased. If we assume that  $Q_2$  and  $Q_3$  are working then the voltage at base  $B_2$  is given as

$$V_{B_2} = (V_{BE\text{sat}})_{Q_2} + (V_{BE\text{sat}})_{Q_3} = 0.8 + 0.8$$

$$\text{or } V_{B_2} = 1.6 \text{ V} \quad \dots(10.35)$$

Since base  $B_1$  is connected to  $+V_{CC}$  through resistor  $R$ , the C-B junction will be forward biased and transistor  $Q_1$  is said to be working in inverse active mode. This causes current  $I_{C_1}$  to flow in reverse direction and thus driving  $Q_2$  and  $Q_3$  in saturation. Thus the output voltage  $V_o = V_{CE\text{sat}} = 0.2 \text{ V} = V(0)$ .

To understand improvement in propagation delay let us consider that both the inputs are HIGH and output of gate is LOW. Thus by equation (10.35),  $V_{B_2} = 1.6 \text{ V}$ . If suddenly any one of the input goes LOW then corresponding drop at base  $B_1$  is  $V_{B_1} = 0.9\text{V}$ . The  $Q_2$  and  $Q_3$  can not turn OFF immediately as excess charge is stored at their base. So at this instant  $V_{B_2} = 1.6$ . This causes reverse biasing of B-C junction of  $Q_1$ , where as B-E junction of  $Q_1$  is already forward biased by LOW input. Thus  $Q_1$  works in ACTIVE region. This is why we have taken  $V_{BE\text{ACTIVE}}$  while calculating equation (10.34). While  $Q_1$  is in active region it conducts a large collector current  $I_{C_1}$  in a direction opposite to  $I_{B_2}$ . This causes very fast removal of excess charge carriers from base of  $Q_2$  and  $Q_3$ . Thus the propagation delay is reduced.

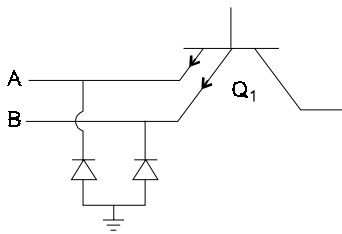
**PASSIVE LULLUP** As evident from figure register  $R_{C_3}$  is **pullup resistor** for the reasons stated earlier. Since  $R_C$  is passive element the circuit is called **passive pullup** circuit. In contrast to this, the resistor  $R_E$  is called **pull down resistor**, as drop across  $R_E$  turns ON the  $Q_3$  and consequently output goes LOW from HIGH state.

NOTE that if in Fig. 10.47, we use single emitter transistor, then the circuit becomes **TTL Inverter or TTL NOT gate**.

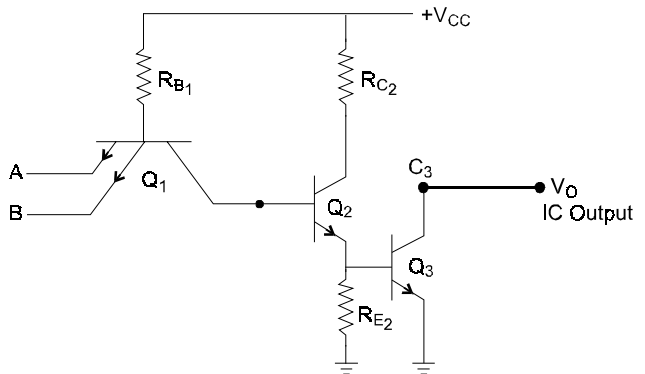
### *Propagation Delay*

Propagation Delay is determined by charging and discharging of output capacitance  $C_O$  (Fig. 10.47) and removal of excess charge carriers from base of  $Q_3$ .  $C_O$  is the capacitance that appears when looking from the output of logic gate. When the output is HIGH,  $C_O$  is charged to  $+V_{CC}$ . When output makes HIGH to LOW transition the  $Q_3$  goes to saturation. Thus  $C_O$  can discharge through ON transistor  $Q_3$ . Since the resistance offered by the transistor  $Q_3$  in saturation very small,  $C_O$  can discharge quickly to  $V(0)$ , through the path shown in Fig. 10.47. When output makes LOW to HIGH transition, the excess charge of base of  $Q_3$  removed by active mode working of  $Q_1$  and  $C_O$  charges to  $V(1)$  (or  $+V_{CC}$ ) through resistor  $R_{C_3}$  as shown in Fig. 10.47. Thus the charging time constant  $\tau = R_{C_3} \cdot C_O$ .  $C_O$  is a bit large, thus charging of  $C_O$  to  $V(1)$  is slower. This limits the switching time of gate. One way to improve is reducing  $R_{C_3}$  but it will increase the collector current. This will increase the power dissipation and reduce the FAN OUT. Another way is to use active pull up arrangement, which is discussed in subsection 10.3.62.

**Clamping (or Protective) Diodes** are used to protect the circuit from damages that can occur due to excess current because of presence negative voltage at input. The input to TTL gates are always positive. Arrangement of clamping diodes is shown in Fig. 10.48. These diodes can clamp up the negative undershoot of approximately up to  $-0.7$  V.



**Fig. 10.48** Clamping diodes at input



**Fig. 10.49** Open collector output of TTL

**Open collector output** When the collector resistance  $R_{C3}$  of Fig. 10.47 is removed and collector of output transistor  $Q_3$  is made available directly as IC pin, the output is called open collector output. However these outputs must be connected to  $+V_{CC}$  through an external resistor, before taking the output.

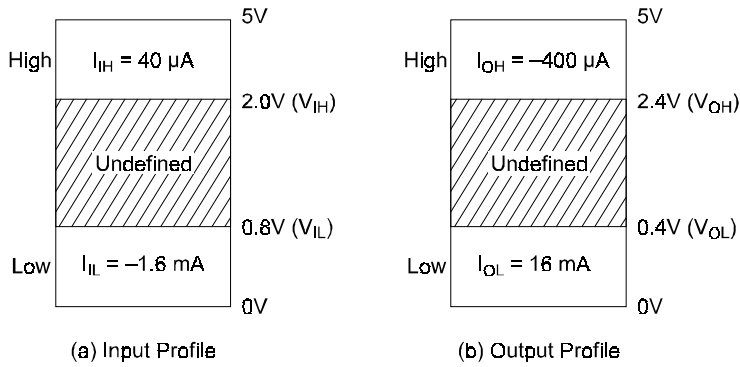
**Wired ANDING** If the output of two TTL gates are wired together then the logic performed is AND. The explanation is same as DTL. For reasons stated for DTL, wiring of TTL gates reduce the FANOUT. To improve the FANOUT open collector output can be used under wired logic condition.

For this purpose, first all the open collectors are tied together and then a single collector resistance is connected to  $+V_{CC}$  from this joint. This ensures that collector resistance for all the gates remains same (which became  $R_C/2$  in DTL and Passive Pullup TTL) and hence the collector current does not increase through output transistor  $Q_3$  of all gates. Thus the FAN OUT can be improved than the FAN OUT obtained by wiring of passive pullup TTL.

**Floating Inputs** of TTL are treated as logic 1 by the circuit. It is because when an input is HIGH the corresponding B-E junction is reverse biased and no current flow through it. Similarly when a input is unconnected *i.e.*, floating, no current flows through it. Thus the floating inputs are treated as logic 1. But floating TTL inputs can easily pickup the noise and circuit can malfunction. Thus it is highly recommended that unused TTL inputs should be tied to  $+V_{CC}$ .

#### 10.3.6.1 Specifications of Standard TTL

The various current and voltage parameters, propagation delays are summarized in Fig. 10.50 and Table 10.5. These parameters are defined in subsection 10.2.2



**Fig. 10.50** Input/Output profile of standard TTL.

**Table 10.5 : Parameters of standard TTL**

Parameter	Typical Value
$t_{PHL}$	15 n sec
$t_{PLH}$	20 n sec
$I_{CC}(1)$	8 mA
$I_{CC}(0)$	22 mA
NOISE MARGIN	0.4 V
FAN OUT	10

**Example 10.6.** Find out the FAN OUT, figure of merit, noise margin average power dissipation for a TTL gate with the help of datas given in Figure 10.50 and Table 10.5.

**Solution.** To calculate FAN OUT consider the Figure 10.51 from figure it is clear that

$$I_{OH} = -NI_{IH} \quad \dots(10.36(a))$$

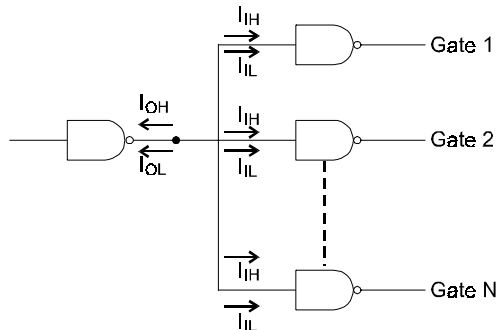
and

$$I_{OL} = -NI_{IL} \quad \dots(10.36(b))$$

by using the datas of figure 10.50 we have

$$I_{OH} = -400 \mu A \quad \text{and} \quad I_{IH} = 40 \mu A$$

$$I_{OL} = 16 \text{ mA} \quad \text{and} \quad I_{IL} = -1.6 \text{ mA}$$



**Fig. 10.51** Loading of TTL

by using equation (10.36(a)) HIGH STATE FAN OUT  $N = -\frac{I_{OH}}{I_{IH}}$

so 
$$N = -\frac{-400\mu\text{A}}{40\mu\text{A}} = 10$$

by using equation (10.36(b)) LOW state FANOUT  $N = -\frac{I_{OL}}{I_{IL}}$

so 
$$N = \frac{-16\text{ mA}}{-1.6\text{ mA}} = 10$$

The FAN OUT of a gate is lowest of HIGH STATE FAN OUT and LOW STATE FAN OUT. Since in TTL, both are same so for standard TTL, FAN OUT = 10.

the propagation delay 
$$t_p = \frac{t_{PHL} + t_{PLH}}{2}$$

$$= \frac{15\text{ n sec} + 20\text{ n sec}}{2} \text{ (by Table 10.5)}$$

so 
$$t_p = 17.5\text{ n sec}$$

High state Noise margin 
$$\Delta 1 = V_{OH} - V_{IH}$$

$$= 2.4 - 2.0 \text{ (by Figure 10.50)}$$

$$= 0.4\text{ V}$$

Low state Noise margin 
$$\Delta V = V_{IL} - V_{OL}$$

$$= 0.8 - 0.4 \text{ (by Figure 10.50)}$$

$$= 0.4\text{ V}$$

The average power dissipation 
$$P_{\text{avg}} = \frac{P(1) + P(0)}{2}$$

$$P(1) = V_{CC} \cdot I_{CC}(1)$$

$$= 5 \times 8 \times 10^{-3} \text{ (by Table 10.5)}$$

so 
$$P(1) = 40\text{ mW}$$

$$P(0) = V_{CC} \cdot I_{CC}(0)$$

$$= 5 \times 22 \times 10^{-3} \text{ (by Table 10.5)}$$

$$= 110\text{ mW}$$

so 
$$P_{\text{avg}} = \frac{40\text{ mW} + 110\text{ mW}}{2}$$

so 
$$P_{\text{avg}} = 75\text{ mW}$$

$$\text{figure of merit} = P_{\text{avg}} \times t_p$$

$$= 75\text{ mW} \times 17.5\text{ n sec} = 1312.5\text{ pJ}$$

### 10.3.6.2 TTL With Active Pullup

In the basic TTL NAND gate circuit (Fig. 10.47) the speed of operation is limited mainly due to the charging of output capacitance  $C_O$ . Since  $C_O$  charges through  $R_{C_3}$  process is time consuming. If the resistance  $R_{C_3}$  is replaced by a transistor  $Q_4$  and a diode  $D$ , as shown in Figure 10.52, then the delays can be reduced. Since the transistor  $Q_4$  (which is an active



device) provides the charging current to output capacitance  $C_O$ , the circuit shown in Fig. 10.52 is called as active pullup circuit, and the output is called **active pullup output**. Also note that transistors  $Q_4$  and  $Q_3$  forms a totem-pole pair, and hence the output is also called as **totem pole output**. Transistor  $Q_2$  acts as phase splitter for totem pole transistor pair. Advantage of active pullup is increased speed without increasing power dissipation. The purpose of using Diode D is to keep transistor  $Q_4$  in cutoff when output of gate is LOGIC 0.

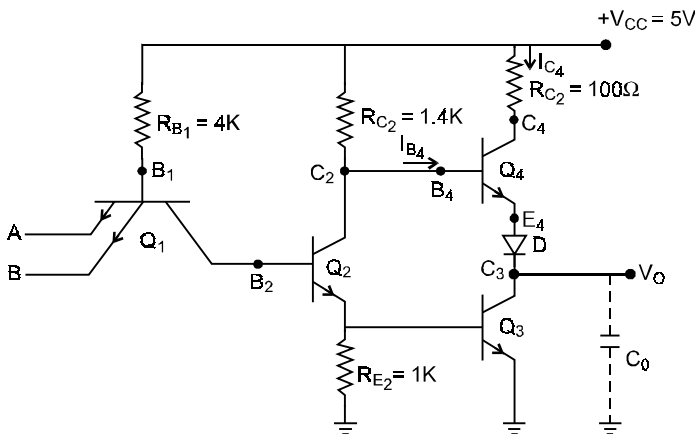


Fig. 10.52 TTL NAND Gate with Active Pullup.

When both the inputs are HIGH transistor  $Q_3$  conducts and the output of gate is LOW i.e.,  $V(0) = V_{CEsat} = 0.2 \text{ V}$ .

When any of input goes LOW then transistor  $Q_4$  conducts and enter into saturation. This causes charging of output capacitance  $C_O$  through  $Q_4$  and diode D. As the  $C_O$  starts charging towards  $V(1)$  i.e., HIGH, the current through  $Q_4$  decreases. The transistor  $Q_4$  goes to cutoff as soon as  $C_O$  charges to  $V(1)$ . Infact under steady state  $Q_4$  just comes out of conduction. Hence the output voltage when output is HIGH can be given as

$$V(1) = V_{CC} - (V_V)_{Q_3} - (V_V)_D$$

$$V(1) = 5 - 0.5 - 0.6$$

so  $V(1) = 3.9 \text{ V}$  ... (10.37)

**Effect of Q4**

Let us see the effect of  $Q_4$  in the circuit. Due to  $Q_4$  the value of  $V(1)$  is reduced to 3.9 V as calculated in equation 10.37. Secondly, the transistor  $Q_4$  causes a large current spike to appear in the circuit when output makes a transition from LOW to HIGH. Thus if frequency of input is high then  $Q_4$  turn ON for larger number of times per second. Hence the average power dissipation is increased if the gate is operated at High frequency.

**Example 10.7.** Explain how diode D of Figure 10.52 keeps the pullup transistor of TTL gate in cutoff when output in LOW state.

**Solution.** When the output of gate is in LOW state  $V_0 = V(0) = 0.2 \text{ V}$ . At this time tansistor  $Q_2$  and  $Q_3$  are in saturation. Thus voltage at collector of  $Q_2$  is given as

$$V_{C_2} = (V_{BEsat})_{Q_3} + (V_{CEsat})_{Q_2} = 0.8 + 0.2$$

so

$$V_{C_2} = 1.0 \text{ V}$$

Since base of  $Q_4$  and collector of  $Q_2$  are connected together so

$$V_{B_4} = V_{C_2} = 1.0 \quad \dots(10.38)$$

Thus the drop across B-E junction of  $Q_4$  and Diode D is given as

$$\begin{aligned} V &= V_{B_4} - V(0) \quad \text{see Fig. 10.52} \\ &= 1.0 - 2.0 \end{aligned}$$

i.e.,

$$V = 0.8 \quad \dots(10.39)$$

is drop between point  $B_4$  and  $C_3$  in Fig. 10.52 for  $Q_4$  and D to conduct cutoff value required

$$\begin{aligned} V_v &= (V_v)_{Q_4} + (V_v)_D = 0.5 + 0.6 \\ V_v &= 1.1 \text{ V} \end{aligned}$$

Since drop between  $B_4$  and  $C_3$   $V < V_v$   $Q_4$  and D are in cut off. If diode is not present then  $V_v = (V_v)_{Q_4} = 0.5V$ , then  $V > V_v$  and transistor  $Q_4$  will be in conduction. Thus presence of diode D keeps  $Q_4$  in cut off when output is LOW.

**Example 10.8.** Explain how transistor  $Q_4$  turns ON during LOW to HIGH transition of output. Calculate the current spike magnitude due to this transition.

**Solution.** Let the output is in steady state LOW. At this moment the transistor  $Q_3$  is in saturation and  $Q_4$  is in cutoff, because all inputs are HIGH. If suddenly any of the input goes LOW then the corresponding B-E junction of  $Q_1$  conducts. As a consequence the transistor  $Q_2$  and  $Q_3$  goes to cutoff. Since  $Q_2$  is going to cutoff its collector voltage rises. Consequently the voltage at base  $B_4$  rises. This causes  $Q_4$  to enter into saturation. But the output cannot change instantaneously, as output capacitance  $C_O$  cannot change its state instantaneously. So the voltage at base  $B_4$  at this instant.

$$\begin{aligned} V_{B_4} &= (V_{BEsat})_{Q_4} + V_D + V(0) \\ &= 0.8 + 0.7 + 0.2 \end{aligned}$$

so

$$V_{B_4} = 1.7 \text{ V} \quad \dots(10.40)$$

Thus

$$\begin{aligned} I_{B_4} &= \frac{V_{CC} - V_{B_4}}{R_{C_2}} \quad \text{no } I_{C_2} \text{ flows as } Q_2 \text{ is in OFF state.} \\ &= \frac{5 - 1.7}{1.4 \times 10^3} \end{aligned}$$

so

$$I_{B_4} = 2.36 \text{ mA} \quad \dots(10.41)$$

the current

$$I_{C_4} = \frac{V_{CC} - (V_{CEsat})_{Q_4} - V_D - V_O}{R_{C_4}} = \frac{5 - 0.2 - 0.7 - 0.2}{100}$$

so

$$I_{C_4} = 39 \text{ mA.}$$

Thus the current drawn from supply during the transition  $I = I_{C_4} + I_{B_4} = 39 \text{ mA} + 2.36 \text{ mA}$

So

$$I = 41.36 \text{ mA} \quad \dots(10.42)$$

Thus high current spike is generated during LOW to HIGH transition of output. This current supply increases the power dissipation and generates noise in power supply distribution by drawing a current spike for short duration.

WIRED LOGIC is not recommended in TTL with active pullup. We have seen in example 10.8, during the LOW the HIGH transition of output high current spike is generated. Performing wired logic with active pullup will increase this spike. This excessive current can damage the whole circuit.

### 10.3.6.3 Schottky TTL

We have seen that the switching speed of TTL gates are limited mainly due to the storage time delay required by a transistor to go to CUTOFF from saturation. Although the transistor  $Q_1$  of basic TTL gate helps fast removal of excess charges but the fact that transistor  $Q_3$  will be driven into saturation makes the device slower. If the schottky transistors are used to create the TTL NAND gate circuit, speed will be high. It is because, as discussed in subsection 10.1.34, the schottky transistors are never driven hard into saturation, rather they are on the verge of saturation. Hence transistors operate in active region. This means there is no excess charge carriers accumulated because transistors are not saturated. Hence speed of operation is improved. The TTL gates employing schottky transistors have propagation delay as low as 2 n sec. Since transistors in this family are not driven in saturation, schottky family is called NON SATURATED LOGIC FAMILY.

### 10.3.6.4 TTL Series

As discussed we have different form of TTL circuits, suited for different applications and requirements. This evolved different series of TTL circuits available in ICs. Some of them are obsolete and some of them are still in use. In particular the 74 XX and 54XX series are popular. The 74XX series is used for general purpose, and consumer applications where as 54XX series is used for military applications. Obviously the temperature range of two types differs. The 74XX has supply range  $5\text{ V} \pm 5\%$  where as 54XX has supply range  $5\text{ V} \pm 10\%$ . For most practical purposes 74XX series are used. The 74XX series is further divided in five categories summarised in Table 10.6.

**Table 10.6 :** Categories of TTL

<i>TTL Series</i>	<i>Letter Prefix</i>	<i>Examples</i>
Standard TTL	74XX	7402, 7404
HIGH Power TTL	74HXX	74H02
LOW Power TTL	74LXX	74L02
Schottky TTL	74SXX	74SO2
Low Power Schottky TTL	74LSXX	74LSO2

Out of the five categories 74LSXX series is most popular. We next present summary of various specifications of TTL ICs in Table 10.7.

**Table 10.7 : Specifications of various TTL families**

Parameters	Series				
	7400	74H00	74LOO	74SOO	74LSOO
$V_{OH}$	2.4 V	2.4 V	2.4 V	2.7 V	2.7V
$V_{OL}$	0.4 V	0.4 V	0.4 V	0.5 V	0.5 V
$V_{IH}$	2.0 V	2.0 V	2.0 V	2.0 V	2.0 V
$V_{IL}$	0.80 V	0.8 V	0.7 V	0.8 V	0.8 V
$I_{OH}$	-400 $\mu$ A	-500 $\mu$ A	-200 $\mu$ A	-1000 $\mu$ A	-400 $\mu$ A
$I_{OL}$	16 mA	20 mA	3.6 mA	20 mA	8 mA
$I_{IH}$	40 $\mu$ A	50 $\mu$ A	10 $\mu$ A	50 $\mu$ A	20 $\mu$ A
$I_{IL}$	-1.6 mA	-2.0 mA	-0.18 mA	-2.0 mA	-0.36 mA
$t_{PHL}$	15 n sec	10 n sec	60 n sec	5 n sec	15 n sec
$t_{PLH}$	22 n sec	10 n sec	60 n sec	4.5 n sec	15 n sec
$I_{CC}$ (1)	8 mA	16.8 mA	0.8 mA	16 mA	1.6 mA
$I_{CC}$ (0)	22 mA	40 mA	2.04 mA	36 mA	4.4 mA
FAN OUT	10	10	20	10	20

Note that specifications of standard TTL were already presented in subsection 10.3.61, but are summarized in Table 10.7 for comparison. Although the data presented in table 10.7 are typical for TTL families, we highly recommend to refer to manufacturers data sheet for exact ratings. Note that from table 10.7 it is very much evident that lower propagation delay comes at the cost of increased power dissipation.

### 10.3.7 Emitter Coupled Logic (ECL)

ECL is another NON Saturated LOGIC FAMILY in which transistors are driven into cutoff and active region. The ECL family offers improved noise performance and is fastest of all logic families. Infact propagation delays as low as 1 n sec are possible with ECL. Basic ECL gate is described in subsection 10.3.72.

Infact the basic TTL, DTL, RTL etc. all have limits on their speed because they are based saturation region operation of transistors. If transistors are driven in active region, then the speed can be dramatically improved. But active region spans to a small voltage width and small temperature variations, component drift can shift the transistors to saturation. Thus stable operation in active region is difficult. A high emitter resistance can solve this problem, but poses another problem. It makes difficult to turn the transistor in active region from cutoff region with small voltage change at input. The problem can be solved if current drawn by the transistors from supply is not switched OFF and ON. The difference amplifier is a circuit in which current drawn from supply is never switched OFF, rather it is shifted from one transistor to another. Also note that using difference amplifier offers high CMRR, means there is effective rejection of noise.

The basic ECL gate is formed around a difference amplifier only. The logic operations are performed by difference amplifier and its outcome is given to output stage. For this reason we first present simplified description of difference amplifier and then move to the basic ECL gate circuit.

10.3.7.1 Difference Amplifier

Basic configuration of a difference amplifier is shown in Fig. 10.53. Voltage at the base of transistor  $Q_2$  is kept constant by supplying a voltage  $V_R$ . The values of resistor  $R_{E1}$ ,  $R_{C2}$  and  $V_R$  are chosen such that  $Q_2$  operates in active region. Whenever an input  $V_i > V_R$  is applied to the base of  $Q_1$ , the transistor  $Q_1$  conducts in active region and  $Q_2$  goes to cutoff. The situation is reversed when  $V_i$  is well below the  $V_R$ . Let us see how it is happening.

**When  $V_i \ll V_R$  i.e.,** when  $V_i < (V_{\nu})_{Q_1}$ . At this time  $Q_2$  start conduction causing a current to flow through the emitter resistance  $R_E$ . The current through emitter resistor  $R_E$  is

$$I_E = I_{E1} + I_{E2}$$

$$= I_{E2} \text{ as } I_{E1} = 0$$

Thus the drop across the emitter resistance  $R_E$  is

$$V_{RE} = V_E = I_{E2}R_E \tag{10.43}$$

Since  $Q_2$  is conducting the drop  $V_E$  is such that  $(V_{BE})_{Q_2} > V_{\nu}$ .

**When  $V_i = V_R$**  As soon as  $V_i$  exceeds  $V_{\nu}$  for  $Q_1$ , it starts conduction and when  $V_i = V_R$  both the transistors conduct same current, through them. This reduces the forward biasing of  $Q_2$ . Because,

$$(V_{BE})_{Q_2} = V_{B_2} - V_{E_2}$$

$$= V_R - V_{E_2}$$

$$(V_{BE})_{Q_2} = V_R - V_E \tag{10.44}$$

and 
$$V_E = (I_{E1} + I_{E2}) R_E \tag{10.45}$$

**When  $V_i > V_R$**   $Q_1$  conducts more heavily and  $Q_2$  conducts less because  $V_E$  increases further when  $V_i > V_R$ . It is because

$$(V_{BE})_{Q_1} = V_{B_1} - V_E$$

but 
$$V_{B_1} = V_i \text{ applied input}$$

so 
$$V_E = V_i - (V_{BE})_{Q_1}$$

$$V_E = V_i - 0.7 \tag{10.46}$$

Thus if  $V_i \uparrow$  it causes  $V_E \uparrow$ .

By equation (10.44)  $(V_{BE})_{Q_2} = V_R - V_E$

So if  $V_E \uparrow$  is causes  $(V_{BE})_{Q_2} \downarrow$

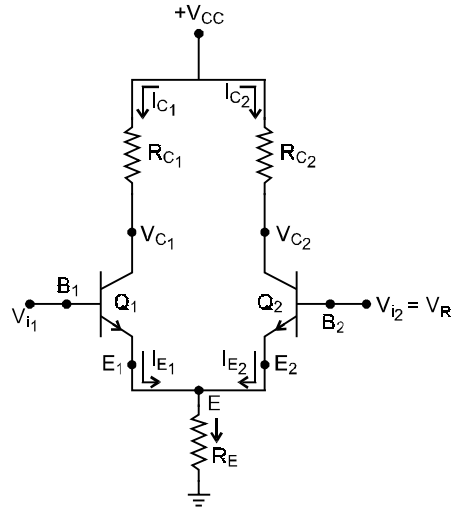


Fig. 10.53 Difference Amplifier.

So we can conclude that when  $V_i \uparrow$  we get  $(V_{BE})_{Q_2} \downarrow$  and if  $V_i$  is sufficiently high  $(V_{BE})_{Q_2} < V_v$ . Consequently  $Q_2$  turn off and only  $Q_1$  remains in conduction. Thus the current  $I_E$  is flowing only due to  $I_{E1}$ . Hence applying  $V_i > V_R$  the current  $I_E$  is shifted from  $Q_2$  to  $Q_1$  without turning OFF the supply, which is the requirement.

10.3.7.2 ECL OR/NOR Gate

The basic ECL gate is shown in Fig. 10.54. As earlier stated it is a non saturated logic family. The transistors used in this family needs slightly high specifications than those we have discussed earlier. Typically they require

$$V_v = 0.7V$$

$$V_{BE\text{ACTIVE}} = 0.75 V$$

$$\text{High } \beta, h_{FE} = 50 \text{ to } 150$$

Circuit Description

The circuit essentially is divided in three parts. They are input stage, difference amplifier and emitter followers or level translators. Note that the  $V_{CC} = 0$ , i.e., grounded and a negative supply is used. Thus the directions of current is conventional. Use of negative voltage offers better noise performance and protects the circuit from accidental grounding.

The input transistors are used to take logic inputs A and B. The two transistors are replaced for transistor  $Q_1$  of Fig. 10.53.

Second stage is difference amplifier, which is heart of the circuit. All the logic operations are performed by this stage only. Also note that emitters of transistors in this stage are connected together and thus the name emitter coupled logic.

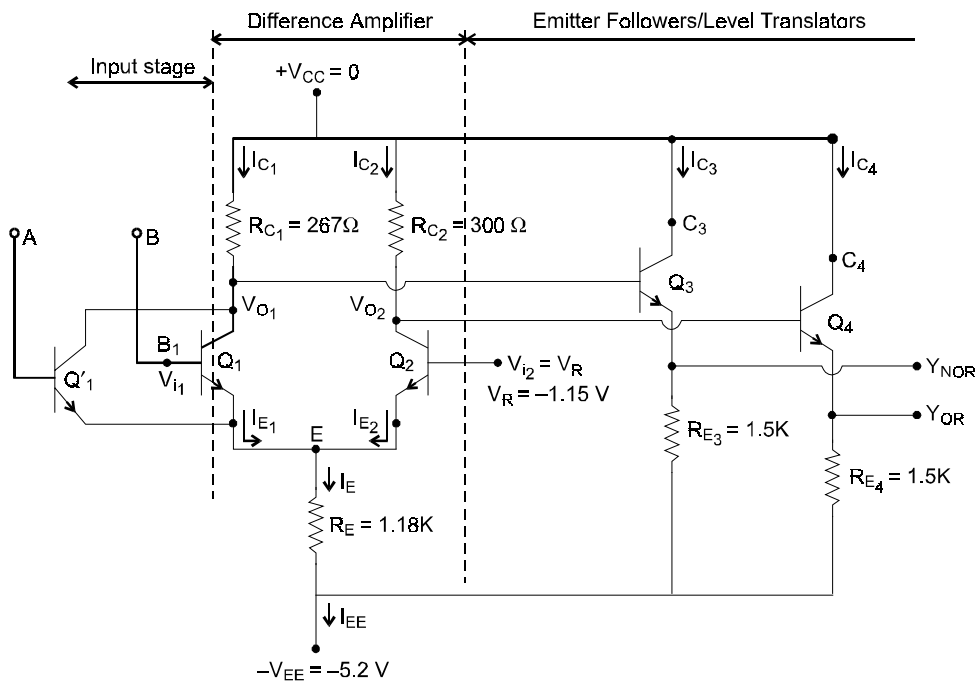


Fig. 10.54 Basic ECL OR/NOR Gate

The use of emitter followers increases the resistance that appears at emitter of difference amplifier, which is desired. It is also use to shift the output voltage such that output level and input voltage levels corresponding to  $V(0)$  and  $V(1)$  are same.

**Circuit Operation**

**When all the inputs are LOW** transistors  $Q_1$  and  $Q'_1$  are OFF and  $Q_2$  is ON. This causes a current  $I_{C2}$  to flow through the resistor  $R_{C3}$ , and consequently collector of  $Q_2$  goes LOW. This is provided at the output through  $Q_4$  at  $Y_{OR}$  output. Since current through  $R_E$  only due to  $Q_2$ , voltage at common collector of input  $V_{O1}$  goes HIGH and this is provided at output  $Y_{NOR}$  through transistor  $Q_3$ . Thus when all inputs are LOW *i.e.*,

$$A = B = \text{LOW} \quad Y_{OR} = \text{LOW} \quad \text{and} \quad Y_{NOR} = \text{HIGH.}$$

**When either or both inputs are HIGH** the transistor  $Q_2$  goes to cutoff and input transistor(s) conduct causing a current  $I_{C1}$  to flow through resistor  $R_{C1}$ . This causes the voltage  $V_{O1}$  to go LOW and this is provided at the output  $Y_{NOR}$  through transistor  $Q_3$ . Since  $Q_2$  is in cutoff voltage  $V_{O2}$  at its collector goes HIGH, which is provided at output  $Y_{OR}$  through transistor  $Q_4$ . Thus

when

$$A = B = \text{HIGH}$$

or

$$A = \text{HIGH and } B = \text{LOW or } A = \text{LOW and } B = \text{HIGH}$$

then

$$Y_{OR} = \text{HIGH and } Y_{NOR} = \text{LOW}$$

The above discussion adequately concludes that the output available at  $Y_{OR} = A + B$  and  $Y_{NOR} = \overline{A + B}$ . Unlike other gates, the ECL gate provides complementary outputs, which is helpful in many of the design problems. The symbol of ECL OR/NOR gate is shown in Fig. 10.55.

**OPEN EMITTER OUTPUT** In high speed digital application the resistors  $R_{E3}$  and  $R_{E4}$  are not connected and the emitters of  $Q_3$  to  $Q_4$  are available directly as the output pin of IC, as shown in Fig. 10.56. Infact in such applications the IC output is used to drive the load line directly. The load line at the other end is terminated by the characteristics impedance of load line.

**10.3.7.3 Circuit Analysis**

In this subsection we are concerned with the calculate of various parameter. The main reason to separate this analysis from previous subsection is to simplify the discussion.

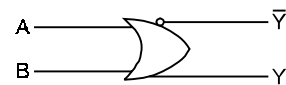
**Let us first consider that all inputs are low**

In this situation all input transistors are cutoff, only  $Q_2$  will conduct. Thus

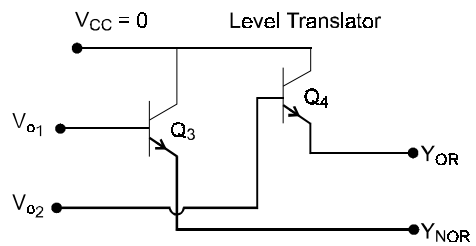
$$I_{E1} = 0$$

$$\begin{aligned} \text{Voltage at point E, } V_E &= V_R - (V_{BE})_{Q_2} \\ &= -1.15 - 0.75 \end{aligned}$$

$$V_E = -1.9 \text{ V} \quad \dots(10.47)$$



**Fig. 10.55** Symbol of ECL OR/NOR Gate



**Fig. 10.56** Open emitter output.

thus current 
$$I_E = \frac{V_E - V_{EE}}{R_E} = \frac{-1.9 - (-5.2)}{1.18 \times 10^3} = 2.7966 \text{ mA}$$

So 
$$I_E \cong 2.8 \text{ mA} \quad \dots(10.48)$$

Since  $Q_2$  in active region  $I_{B2} \ll I_{C2}$  so  $I_{C2} \cong I_{E2} = I_E$

thus 
$$I_{C2} \cong 2.8 \text{ mA}$$

So voltage across collector resistance  $R_{C2}$  is given as

$$V_{RC2} = I_{C2} \cdot R_{C2} = 2.8 \times 10^{-3} \times 300$$

$$V_{RC2} = 0.84 \text{ V}$$

So voltage at collector of  $Q_2$ , is given as

$$V_{O2} = V_{CC} - V_{RC2} = 0 - 0.84$$

So 
$$V_{O2} = -0.84 \text{ V} \quad \dots(10.49)$$

At this time  $Q_4$  is conducting, so

$$Y_{OR} = V_{O2} - (V_{BE})_{Q4} = -0.84 - 0.75$$

So 
$$Y_{OR} = -1.59 \text{ V} \quad \dots(10.50)$$

At this time  $Y_{OR}$  is considered LOW, see circuit operation, so

$$V(O) = -1.59 \text{ V} \quad \dots(10.51)$$

since  $Q_1$  and  $Q_1'$  are in cutoff  $V_{O1} = 0V$ , thus  $V_{C3} = V_{B3} = 0V$ . Thus B-E-junction of  $Q_3$  acts as diode. Thus

$$Y_{NOR} = V_{CC} - (V_{BE})_{Q3} = 0 - 0.75$$

so 
$$Y_{NOR} = -0.75 \text{ V} \quad \dots(10.52)$$

At this time  $Y_{NOR}$  is HIGH, see circuit operation, so

$$V(1) = -0.75 \text{ V} \quad \dots(10.53)$$

At this time 
$$I_{E3} = \frac{Y_{NOR} - V_{EE}}{R_{E3}} = \frac{-0.75 - (-5.2)}{1.5 \times 10^3} = 2.966 \text{ mA}$$

So 
$$I_{E3} \cong 3 \text{ mA} \quad \dots(10.54)$$

**Let input A = HIGH** In this case  $Q_1$  will be conducting and  $Q_2$  will be in cutoff. So

$$V_E = V_i - (V_{BE})_{Q1} = V(1) - 0.75 = -0.75 - 0.75$$

So 
$$V_E = -1.5V \quad \dots(10.55)$$

since 
$$V(1) = -0.75 \text{ V}$$

In this case emitter current  $I_E$  is due to  $Q_1$  only.

So 
$$I_{E1} = I_E = \frac{V_E - V_{EE}}{R_E} = \frac{-1.5 - (-5.2)}{1.18 \times 10^3} = 3.13559 \times 10^{-3}$$

So 
$$I_{E1} = I_E \cong 3.14 \text{ mA} \quad \dots(10.56)$$

$$V_{O1} = V_{CC} - V_{RC1} = 0 - 3.14 \times 10^{-3} \times 267 \text{ as } I_{C1} \cong I_{E1}$$



So  $V_{O1} \cong -0.84 \text{ V}$  ... (10.57)

and  $Y_{\text{NOR}} = V_{O1} - (V_{\text{BE}})_{Q_3} = -0.84 - 0.75$

So  $Y_{\text{NOR}} = -1.59 \text{ V} = V(0)$  ... (10.58)

So same voltage corresponding to LOW output is obtained, which justifies the use of level translators. In the similar way

$$Y_{\text{OR}} = -0.75 \text{ V} = V(1)$$

**Example 10.9.** Find out the Noise margin and average power dissipation by the ECL gate.

**Solution.** To find out noise margin we must find out the base to emitter drop of transistors when they are in cutoff.

When  $Y_{\text{OR}} = \text{LOW}$   $Q_1$  and  $Q_1'$  are in cutoff as  $A = B = V(0)$

$$(V_{\text{BE}})_{Q_1} = V_B - V_E = V(0) - V_E$$

then by equation 10.47 and 10.51

$$\begin{aligned} (V_{\text{BE}})_{Q_1} &= -1.59 - (-1.9) \\ &= 0.31 \text{ V} \end{aligned}$$

So  $(V_{\text{BE}})_{Q_1} \cong 0.3 \text{ V}$

but for ECL transistors are choosen with  $V_v = 0.7 \text{ V}$ .

So a positive noise of  $0.4 \text{ V}$  will make  $Q_1$  in conduction. Thus

$$\Delta O = 0.4 \text{ V} \quad \dots (10.59)$$

Similar situation occurs when  $Y_{\text{OR}} = \text{HIGH}$  and  $Q_2$  is in cutoff. At this time also

$$(V_{\text{BE}})_{Q_2} \cong 0.3 \text{ V}$$

and hence  $\Delta I = 0.4 \text{ V}$  ... (10.60)

Note that, if we take transistors with cutin voltage of  $0.5 \text{ V}$  we will get nose margin of  $0.2 \text{ V}$ . Thus using transistors of high cutin voltages are advantageous.

To find out average power dissipation we must find out the HIGH state and LOW state supply current.

**When  $Y_{\text{OR}} = \text{LOW}$ ,** then by equations (10.48) and (10.54) we get

$$I_E = 2.8 \text{ mA} \text{ and } I_{E3} = 3 \text{ mA}$$

and  $I_{\text{EE}}(O) = I_E + I_{E3} + I_{E4}$

we've  $I_{E4} = \frac{Y_{\text{OR}} - V_{\text{EE}}}{R_{E4}} = \frac{-1.59 - (-5.2)}{1.5 \times 10^3} = 2.4066 \text{ mA}$

So  $I_{E4} \cong 2.41 \text{ mA}$  ... (10.61)

thus  $I_{\text{EE}}(O) = 2.8 \text{ mA} + 3 \text{ mA} + 2.41 \text{ mA}$

or  $I_{\text{EE}}(O) = 8.21 \text{ mA}$  ... (10.62)

when  $Y_{\text{OR}} = \text{HIGH}$  then by equation (10.56)

$$I_E = 3.14 \text{ mA}$$

at this time  $I_{E4} = 3 \text{ mA}$  and  $I_{E3} = 2.41 \text{ mA}$ , as situation for transistors  $Q_3$  and  $Q_4$  are just reversed.

so  $I_{EE}(1) = 3.14 \text{ mA} + 3 \text{ mA} + 2.41 \text{ mA}$   
 or  $I_{EE}(1) = 8.55 \text{ mA}$  ... (10.63)

we've  $P_{avg} = \frac{P(0) + P(1)}{2} = \frac{I_{EE}(0) + I_{EE}(1)}{2} \cdot V_{EE}$   
 $= \frac{8.21 \text{ mA} + 8.55 \text{ mA}}{2} \cdot 5.2 \text{ V} = 43.576 \text{ mW}$

or  $P_{avg} \cong 43.6 \text{ mW}$  ... (10.64)

**FAN OUT** of ECL gates are higher. It is because they offers low output impedance and high input impedance. When all inputs are LOW input transistors offers high input impedance. When any input is HIGH then input impedance of emitter followers appears at input stage, which is already high. Thus input impedance of ECL gate is always High. At the output side  $Q_3$  and  $Q_4$  are always conducting either as diode or transistors. So output impedance is always low. Hence FANOUT is HIGH.

**Wired Logic** performed by the ECL gates are OR. Hence if we simply wire the output of ECL gates we can obtain variety of functions. This is illustrated in Fig. 10.57.

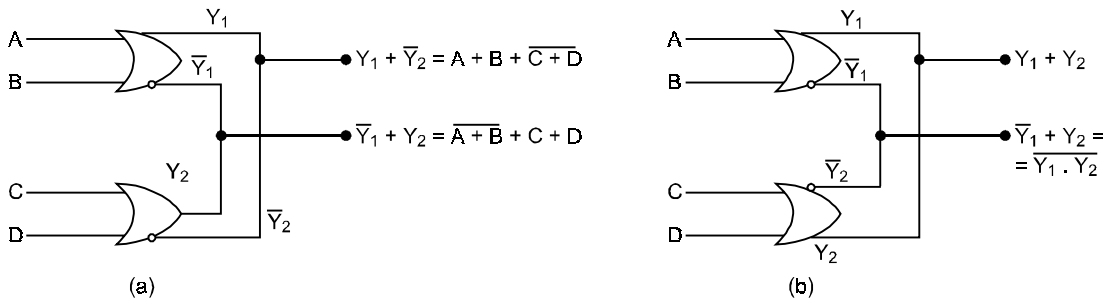


Fig. 10.57 Illustration of wired logic for ECL gate.

Let us see how wired logic comes. Consider the wiring of only  $Y_{OR}$  output as shown in Fig. 10.58(a). Before proceeding further we reinsert that all the voltages are measured with respect to ground.

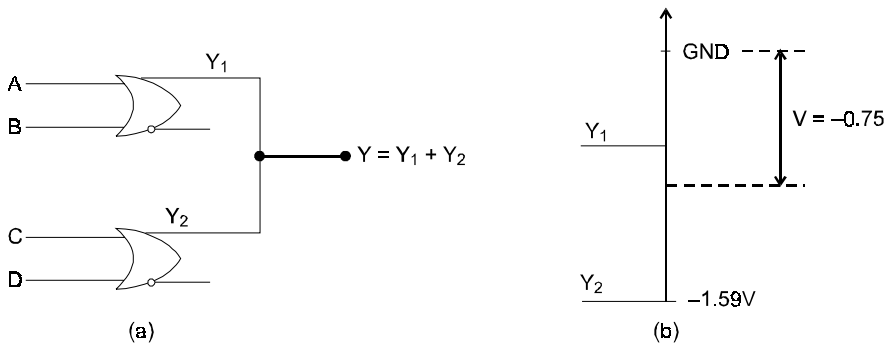


Fig. 10.58 Illustration of wired-or ing in ECL

When both are LOW *i.e.*,  $Y_1 = V(0) = -1.59 \text{ V} = Y_2$  then  $Y = -1.59 \text{ V}$  as  $Y_1$  and  $Y_2$  forms two equipotential points. Hence  $Y = V(0) = \text{LOW}$ .

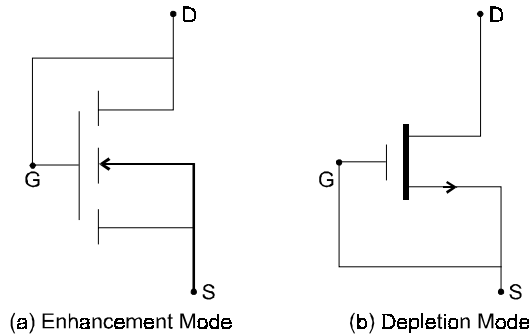
When both  $Y_1$  and  $Y_2$  are HIGH *i.e.*,  $Y_1 = Y_2 = V(1) = -0.75$  then  $Y = -0.75$ . So  $Y = V(1) = \text{HIGH}$ .

When  $Y_1 = V(1) = -0.75$  and  $Y_2 = V(0) = -1.55 \text{ V}$  then when we measure  $Y$  with respect to ground we get  $Y = -0.75 \text{ V}$  as shown in Fig. 10.58(b). Thus  $Y = V(1) = \text{HIGH}$ .

This concludes that wiring two outputs of ECL gates perform wired-OR.

### 10.3.8 MOS Logic

MOS Logic have become popular logic because of their high packing density. As indicated in the beginning the MOS devices, however have larger propagation delays and thus have slower speed than the bipolar logic families. But in present days due to advancements in VLSI technologies, MOS devices can be fabricated even in much smaller areas, which implies that propagation delays will be low. If the present trend continue, MOS families may match the speed of the bipolar families. Another advantage of MOS is, we can use only MOS devices without using any resistor. A MOS device itself can be configured as resistance as shown in Fig. 10.59. When configured as load the MOS devices offers approximately 100 K $\Omega$  resistance. Even though either enhance mode or Depletion mode devices can be as load, preferred are depletion mode load. It is because the depletion mode MOS have maximum current when gate voltage is 0. It is good to use MOS as load resistance because a normal resistor takes area 20 times of area taken by a MOS device.



**Fig. 10.59** NMOS as Load or resistor.

#### 10.3.8.1 NMOS Gates

The various gates are shown below.

**NOT GATE** Basic NMOS NOT gate is shown in figure 10.60. Transistor  $Q_2$  is a depletion mode transistor acting as load. Transistor  $Q_1$  is driver transistor which performs logic operation.

**When  $V_{in} = V(0) = 0$ ,** transistor  $Q_1$  does not conduct and acts as open circuit as shown in figure 10.60(b). Thus  $+V_{DD}$  appears at the output, which is a HIGH voltage. So we get when  $V_{in} = V(0) = \text{LOW}$ ,  $V_o \cong +V_{DD} = V(1)$  *i.e.*,  $Y = \text{HIGH}$ .

**When  $V_{in} = V(1) = +V_{DD}$**  transistor  $Q_1$  start conduction and a current flows through the circuit. In this case transistors  $Q_1$  acts as switch is closed, Condition, as shown in Fig. 10.60 (c). Thus ground voltage or 0V appears at the output which is LOW state. Thus

when  $V_{in} = (1)$ ,  $V_o = 0\text{V} = V(0)$  *i.e.*,  $Y = \text{LOW}$ .

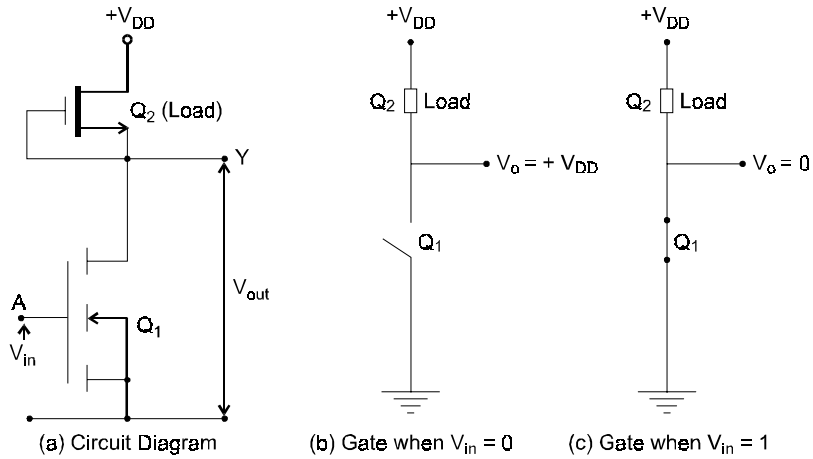


Fig. 10.60 NMOS Inverter with Depletion Load.

From the above discussion it is also clear that the circuit shown in figure 10.60(c) is a NOT gate with the voltage levels  $V(0) = 0\text{ V}$  and  $V(1) = +V_{DD}$ .

**NAND GATE** The NMOS NAND gate is shown in Fig. 10.61. As usual transistor  $Q_3$  is load transistor and  $Q_1$  and  $Q_2$  are driver transistor which performs logic operation. From the circuit we observe that when  $A = B = \text{HIGH} = V(1)$ , then  $Q_1$  and  $Q_2$  conducts and current  $I_{DD}$  can flow in the circuit. In this situation  $Q_1$  and  $Q_2$  acts as switch in closed condition.

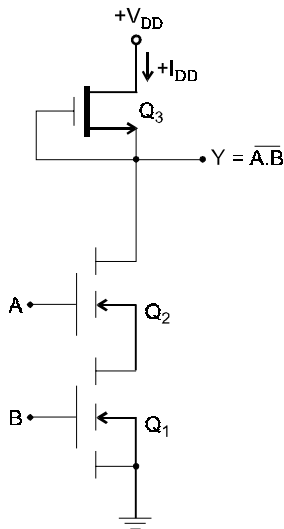


Fig. 10.61 NMOS Nand gate with depletion load.

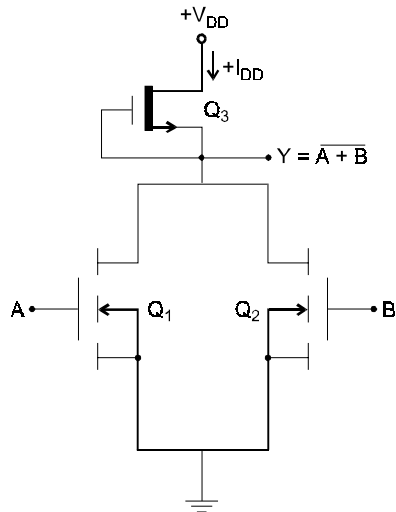


Fig. 10.62 NMOS Nor gate with depletion load.

Thus  $Y = V(0) = 0\text{V}$  when  $A = B = V(1)$ . If  $A = V(0)$  and  $B = V(1)$ , then  $Q_2$  acts as switch in open condition. Consequently  $I_{DD}$  can not flow in the circuit and  $Y = +V_{DD} = V(1)$ . Similar situation occurs when  $B$  is LOW and when both  $A$  and  $B$  are LOW. Thus the circuit gives  $Y = \text{LOW}$  when both  $A$  and  $B$  are HIGH otherwise  $Y = \text{HIGH}$ . This clearly indicates that the circuit in Fig. 10.61 acts as NAND gate. Also note that the power dissipation will be low in this gate as current is drawn from  $+V_{DD}$  only when both the inputs are HIGH. In all other situations no current is drawn.

**NOR GATE** The NMOS NOR gate is shown in Fig. 10.62. Transistor  $Q_3$  is load transistor and  $Q_1$  and  $Q_2$  are drivers which perform logic operation. It is evident from circuit that when both  $A = B = V(0)$ ,  $Q_2$  and  $Q_3$  does not conduct and current  $I_{DD}$  does not flow. Thus  $Y = +V_{DD} = V(1) = \text{HIGH}$ . If any of the inputs goes HIGH current  $I_{DD}$  flows and consequently  $Y = 0V = V(0) = \text{LOW}$ . Thus the output is HIGH only when all inputs are LOW other-wise output is LOW, which is clearly a NOR gate. Note that power dissipation in NOR gate is higher than NAND gate. It is because the current  $I_{DD}$  does not flows only when both inputs are LOW. In all other input conditions current will flow in the circuit, and hence power dissipation is higher than NMOS NAND gate.

10.3.8.2 CMOS Gates

A logic circuit that combines p-channel and n-channel MOS transistors on the same chip is known as complementary MOS or CMOS circuit. Major advantage of CMOS is very low power dissipation. Since technology has improved such that a very small chip area is required to fabricated MOS devices the CMOS can now operate on faster speed. Indeed the CMOS circuits have already replaced TTL in many practical applications. CMOS can operate over a supply range of 3.3 V to 15V. Lowpower CMOS circuits have emerged with supply of 3.3 V (as power dissipation is proportional to  $V_{DD}^2$ ), but lower power supply reduces the noise immunity.

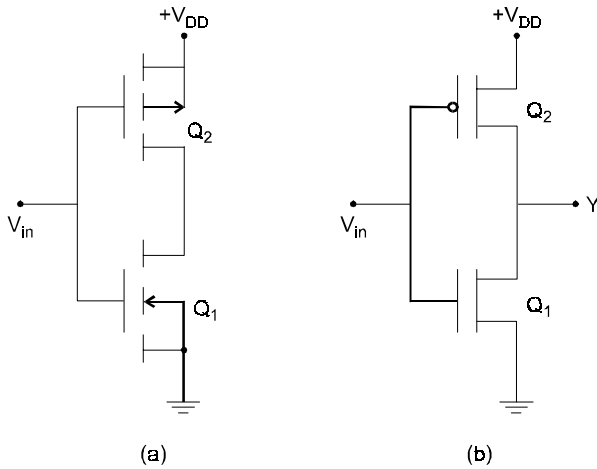


Fig. 10.63 (a) CMOS Inverter (b) Simplified circuit schematic.

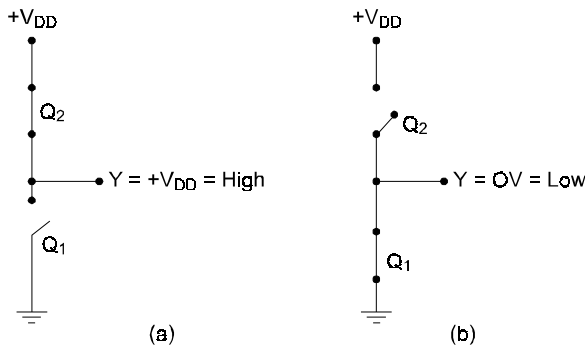


Fig. 10.64 CMOS inverter equivalent circuits.

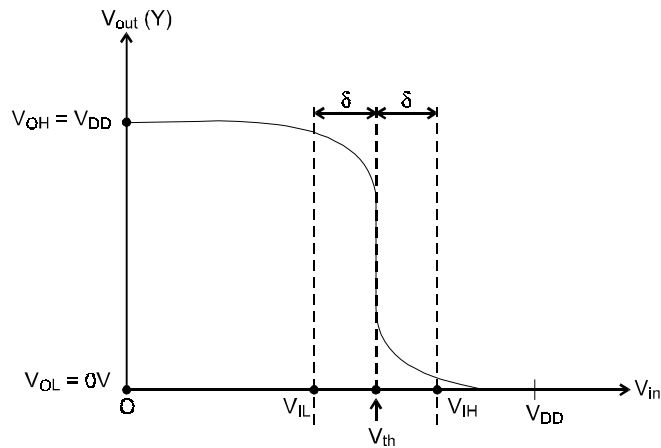


Fig. 10.65 Voltage transfer characteristics of CMOS Inverter.

**CMOS INVERTER** The basic CMOS gate is a NOT gate, called CMOS inverter shown in Fig. 10.63. Transistor  $Q_1$  is an enhancement mode NMOS transistor and transistor  $Q_2$  is enhancement mode PMOS transistor. The circuit operation is simple.

**When  $V_{in} = V(0)$**  transistor  $Q_1$  acts as open circuit and transistor  $Q_2$  acts as close switch. Thus the supply voltage  $+V_{DD}$  is connected at output. Hence  $Y = +V_{DD} = V(1)$ . This is shown in Figure 10.64(a).

**When  $V_{in} = V(1)$**  transistor  $Q_2$  acts as open circuit and  $Q_1$  acts close switch. Thus the ground is connected at the output. Hence  $Y = 0V = V(0)$ . This is shown in figure 10.64(b).

From the above discussion it is clear that circuit of Figure 10.63 acts as inverter.

**Static Power Dissipation** is the dissipation calculated when the output is static *i.e.*, output is steady state HIGH and LOW. From the above discussion it is clear that when  $Y = \text{HIGH}$ ,  $Q_1$  is open circuit so no current flows in the circuit. Similarly when  $Y = \text{LOW}$ ,  $Q_2$  is open circuit so no current flows in the circuit. Thus in either the case no current flows in circuit (as leakage currents are negligibly small). Thus zero static current flows and consequently zero static power dissipation.

**Transfer characteristics** of CMOS inverters is shown in figure 10.65. It is the curve of input voltage Vs output voltage which explains the operation of CMOS inverter in detail. The curve indicates that output voltage makes a sharp transition when input voltage passes through a value  $V_{th}$ . The voltage  $V_{th}$  is called threshold voltage around which the change in output takes place. The voltage  $V_{th}$  is such that if  $V_{in} < V_{th} - \delta$  (or  $V_{IL}$ ) the output  $Y = +V_{DD}$  and if  $V_{in} > V_{th} + \delta$  (or  $V_{IH}$ ) the output is  $Y = 0V$ . Clearly the value of  $\delta$  is very small. This does mean that the input signal need not be exactly equal to  $+V_{DD}$  or  $0V$  to produce correct output. There is a room for some error, called noise. The LOW state and HIGH state noise margins are shown in figure, and calculated as below

	$\Delta 1 = V_{OH} - V_{IH}$	
For figure	$V_{OH} = +V_{DD}$ and $V_{IH} = V_{th} + \delta$	
So	$\Delta 1 = V_{DD} - (V_{th} + \delta)$	
or	$\Delta 1 = V_{DD} - V_{th} - \delta$	...(10.66)
Similarly	$\Delta 0 = V_{IL} - V_{OL}$	

$$= V_{th} - \delta - O \text{ from figure}$$

or 
$$\Delta O = V_{th} - \delta \quad \dots(10.67)$$

This clearly indicates high noise immunity of logic gates.

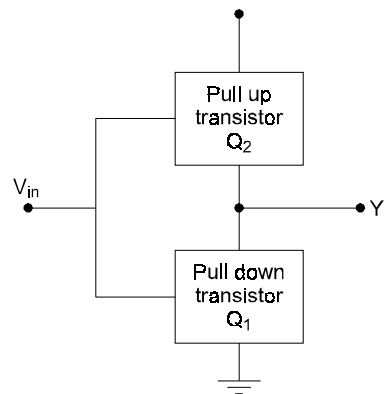
**Dynamic Power Dissipation** is the dissipation that occurs when the gate is changing its output state *i.e.*, when output is going HIGH to LOW or LOW to HIGH. As indicated by transfer characteristics, around the  $V_{th}$  there is situation when both  $Q_1$  and  $Q_2$  are conducting for a short while. It is because the devices can not change its state instantaneously. This produces a short current pulse in the circuit with every change of state of output. Consequently there is power dissipation, called **dynamic power dissipation**. Thus in a CMOS circuit power dissipation depends upon the frequency of operation and increase linearly if frequency is increased. If  $f$  is the frequency of switching and CMOS gate is driving a load capacitance  $C$  then the dynamic power dissipation is given as

$$P_D = f.C.V_{DD}^2 \quad \dots(10.68)$$

where  $V_{DD}$  is supply voltage.

**Propagation Delay** is mainly due to the high capacitance present at the input and output. When CMOS gate is driving  $N$  loads, the input capacitances of all  $N$  gates appears in parallel at the output of driver gate. Thus net output capacitance at drivers output is  $C_{OUT} = C = C_O + NC_i$  where  $C_O$  is capacitance of driver when looking back from output and  $C_i$  is the input capacitance of load gates. Thus a large output capacitance is present whose charging and discharging is slower and hence the lower speed of operation. One way to improve it is to increase supply voltage, but this will increase the dynamic power dissipation and hence the average dissipation.

**Pullup and Pull down** In the circuit shown in Figure 10.63, we note that  $Y = \text{HIGH}$  when  $Q_2$  conducts and  $Q_1$  open circuit. Hence  $Q_2$  can be regarded as pullup transistor. When  $Y = \text{LOW}$ ,  $Q_1$  conducts and can be regarded as pulldown transistor. Thus the circuit shown in Fig. 10.63 can be redrawn as fig. 10.66. Infact Fig. 10.66 shows basic structure of any CMOS gate. Usually NMOS are used to form pull down network and PMOS are used to form pullup network. Connecting the two networks in a fashion shown in Fig. 10.66 can give variety of logic circuits. We use same methodology to draw NAND gates and NOR gates using CMOS.



**Fig. 10.66** Structure of CMOS Inverter.

**NAND and NOR GATE** Basic CMOS NAND gate and NOR gate circuits are shown in Figure 10.67 and 10.68. In both the cases  $Q_1$  and  $Q_2$  forms pull down network and  $Q_3$  and  $Q_4$  forms pull down network.

In NAND gate  $Y = \text{OV} = \text{LOW}$  only when both  $Q_1$  and  $Q_2$  conducts *i.e.*, when both  $A = B = V(1)$ . In all other cases of input either or both the  $Q_1$  and  $Q_2$  are open so  $Y = +V_{DD} = \text{HIGH}$ .

In NOR gate  $Y = +V_{DD} = V(1)$  only when both  $Q_3$  and  $Q_4$  conduct and both  $Q_1$  and  $Q_2$  are open *i.e.*, at  $A = B = V(0) = \text{OV}$ . In all other cases either or both the  $Q_1$  and  $Q_2$  conducts so  $Y = \text{OV} = V(0) = \text{LOW}$ .

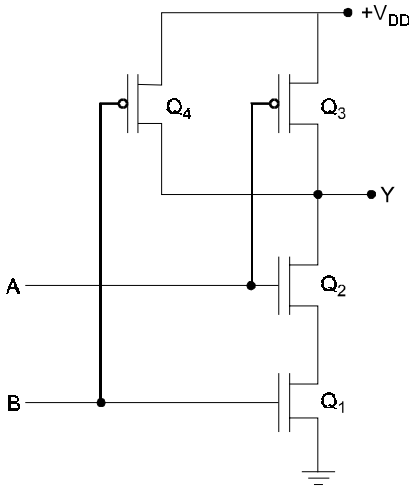


Fig. 10.67 CMOS NAND Gate.

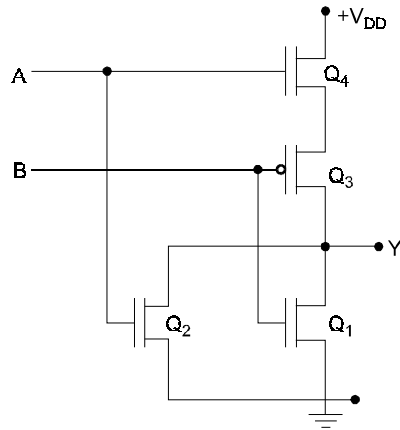


Fig. 10.68 CMOS NOR Gate.

**Example 10.10.** A CMOS inverter is driving an identical inverter at a supply of 5 V with the input fed with square wave of 1MHz. If the input and output capacitances are 40 fF and 20 fF respectively then find out the dynamic power dissipation in the gate ?

**Solution.** We've  $C_i = 40\text{fF}$  and  $C_o = 20\text{fF}$ , then at the output of Driver  $C = C_i + C_o = 40\text{fF} + 20\text{fF} = 60\text{fF}$

Thus driver is said to be driving a capacitance of 60 fF. By equation 10.68 the dynamic power dissipation.

$$P_D = fCV_{DD}^2 = 1 \times 10^6 \times 60 \times 10^{-15} \times (5)^2$$

$$P_D = 1.5\mu\text{W}, \text{ a considerably low dissipation.}$$

but if frequency is increased to 100 MHz,  $P_D = 150\mu\text{W}$ . Hence average power dissipation increases with frequency of operation.

### 10.3.8.3 CMOS Series

Popular CMOS ICs come from 40XX, 54CXX, and 74CXX series. The 40XX series refers to original CMOS design but they lack compatibility with TTL, which is badly needed in practice. The 54C series is used for military purpose and 74C series is used for general purpose electronics. Letter C is used to discriminate the two series from 74XX and 54XX series of TTL. Advantage of 54CXX/74CXX is their pin to pin, function to function compatibility with TTL. For example 7404 is a TTL inverter and 74C04 is a CMOS hex inverter. Both the ICs have inputs, outputs and supply at same pin number. Also a 54CXX series device can replace a 74CXX device, but it is rarely done due to higher cost of 54CXX devices. Table 10.8 summaries the different CMOS series and their compatibility with TTL.

Table 10.8 : Different CMOS series

CMOS Series	Series Prefix	Example	Compatibility With TTL
Standard CMOS series	40XX	4000 4009	No compatibility
Compatible CMOS series	74CXX	74C00 74C04	Pin to Pin and Function to function



CMOS Series	Series Prefix	Example	Compatibility With TTL
High speed Compatible CMOS series	74HCXX	74HC00 74HC04	Pin to Pin and Function to function
High speed electrically compatible CMOS series	74HCTXX	74HCT04	Pin to Pin Function to Function Electrically compatible

Note that 74HCTXX series is electrically compatible means the current and voltage parameters are matched with TTL and at the same time pin to pin compatibility is available with TTL ICs. Various parameters for 74CXXCMOS series are summarized in Fig. 10.69 and Table 10.9, assuming supply voltage of 5 V.

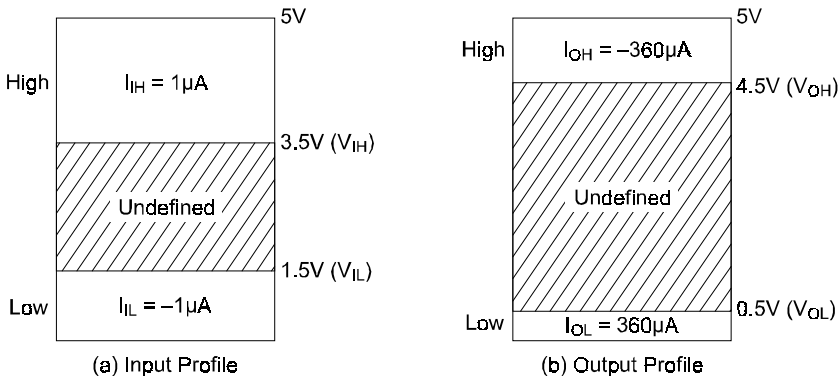


Fig. 10.69. Input/output profile of 74CXX series with supply 5V.

Table 10.9 : Parameters of 74CXX series with supply 5V.

Parameters	Values
$t_{PHL}$	60 n sec
$t_{PLH}$	45 n sec
Static Power Dissipation	10 nW

### 10.3.9 Three State Logic (TSL)

In a logic circuit we have two output states LOW and HIGH. Logic circuits have been designed in which output can assume another state called High impedance state or High. Z in short. Logics with three states of output are called tristate logic (TSL) or three state logic. When the output is in third state, the output is said to be disabled *i.e.*, neither HIGH nor LOW. In a microprocessor based system outputs of many logic devices has to be connected to a common **bus** which in turns may be driving number of other logic devices. Such connections cause a number of problem–

1. If the output is not in HIGH state it does not necessarily mean LOW state output.
2. Accidentally more than one logic device may drive the bus and can cause **bus contention**, which may damage the bus.
3. Active pullup outputs cannot be directly connected, because it introduces very high current spike. This causes overheating of IC and consequently IC may be damaged.

- Connecting open collector output through a common external resistor causes problem of loading and speed.

To overcome such problems TSL are used, in which outputs are in High-Z, and they deliver a HIGH or LOW to bus only when they are made to come out of 3rd state through a control input.

The basic TSL circuit is a inverter called TSL inverter, and shown in Fig. 10.70. A close inspection of circuit reveals that it is a TTL NAND gate with some modification. Readers are advised to glance over the operation of TTL NAND gate.

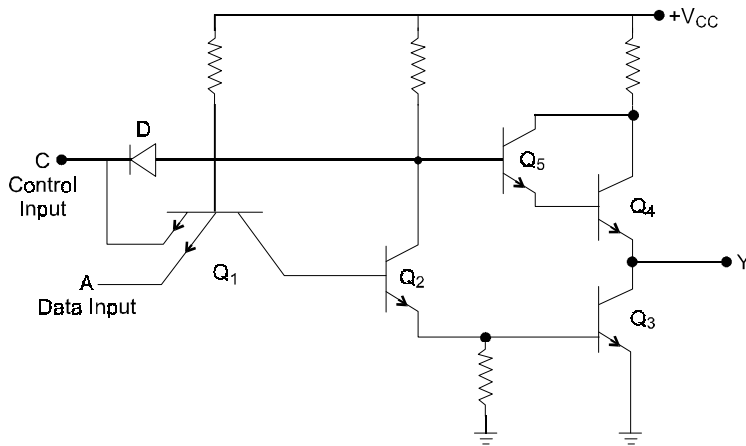


Fig. 10.70 TSL Inverter

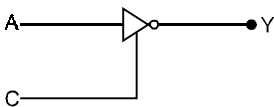


Fig. 10.71 Logic symbol of TSL inverter

Table 10.10 : Truth Table of TSL Inverter

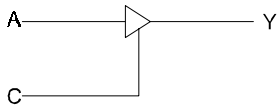
C	A	y
0	X	HIGH-Z
1	A	$\bar{A}$

**When  $C = V(0) = 0V$**  diode D and corresponding B-E junction of  $Q_1$  conducts. Recall that if any of B-E junction of  $Q_1$  conducts, transistors  $Q_2$  and  $Q_3$  does not conduct and acts as open circuit. When diode D is conducting, the diode drop  $V_D = 0.7 V$  appears at the base of  $Q_5$  which is not sufficient to turn ON both  $Q_4$  and  $Q_5$ , thus they act as open circuit. The result is very high impedance appeared at output. Thus the output is in High impedance or 3<sup>rd</sup> state.

**When  $C = V(1) = +V_{CC}$**  The diode is open circuit and corresponding B-E junction of  $Q_1$  is open circuit. The result is that the circuit behaves as 2-input TTL NAND gate whose one of the input is HIGH. Thus the gate acts as inverter.

The logic symbol of a TSL inverter and its truth table is shown in Figure 10.71 and Table 10.10, respectively.

Another useful TSL circuit is **three state buffer** in which output is same as input. Such buffers are commonly used to enhance the driving capability of a logic device. The logic symbol and truth table of a three state buffer is shown in Fig. 10.72 and Table 10.11, respectively.



**Fig. 10.72** Logic symbol of three state Buffer

**Table 10.11 : Truth Table of three state Buffer**

C	A	Y
0	X	HIGH-Z
1	A	A

### 10.4 INTERFACING OF LOGIC GATES

In the last section we have discussed various forms of logic circuits, called logic families. At present time TTL and CMOS are the only used families. In a system many a times it is needed to **interface**. CMOS and TTL gates. By interfacing we mean the method of connecting the output of driver to input of load gate such that their exist a compatibility. In nutshell by interfacing we have to match electrical parameters of two gates, so that logic STATE produced by driver can be correctly recognized by load gate. For interfacing following equations must be satisfied.

$$(V_{OH})_{DRIVER} \geq (V_{IH})_{LOAD} \quad \dots(10.69)$$

$$(V_{OL})_{DRIVER} \leq (V_{IL})_{LOAD} \quad \dots(10.70)$$

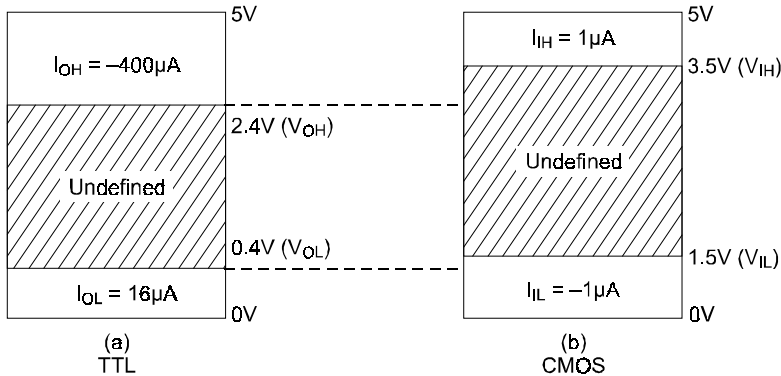
$$-(I_{OH})_{DRIVER} \geq N \cdot (I_{IH})_{LOAD} \quad \dots(10.71)$$

$$(I_{OL})_{DRIVER} \geq -N \cdot (I_{IL})_{LOAD} \quad \dots(10.72)$$

Note that negative sign in current equations means the direction of current opposite to the one assumed in Fig. 10.30.

#### 10.4.1 TTL to CMOS Interface

As stated earlier, our strategy is to match electrical parameters, such that equations (10.69) through (10.74) are satisfied. For this purpose we reproduce the output profile of TTL and input profile of CMOS in Fig. 10.73, assuming that CMOS is also operating at  $+V_{DD} = 5V$ .



**Fig. 10.73** (a) TTL output Profile (b) CMOS Input Profile

From the figure it is clear that HIGH state output of TTL may not be treated as HIGH input always. Because  $V_{OH}$  of TTL is lower than  $V_{IH}$  of CMOS, so equation (10.69) is not satisfied. As shown by figure equation 10.70 is well satisfied and equations (10.71) and (10.72) can be satisfied for reasonable value of N. Thus overall connection problem is to raise  $V_{OH}$  of TTL above 3.5 V. For this purpose a simple method is to use external pullup resistor of 3.3 K $\Omega$ , as shown in Fig. 10.74. The pullup resistor raise the HIGH state voltage to about

+5V and has no effect on LOW state output. When output is in HIGH state, the output capacitance is charged 2.4 V through pullup transistor. As soon as pullup transistor stops conduction external pullup resistor R charges the output capacitance to 5 V. The value 3.3 KΩ is suited in most application and is a compromise between speed and power dissipation. Clearly TTL will sink current through R when output is LOW, thus

$$I_{\text{sink}} = \frac{5 \text{ V}}{3.3 \text{ K}} = 1.515 \text{ mA}$$

The minimum value of R is determined by maximum amount of sink current *i.e.*,

$$R_{\text{min}} = \frac{5 \text{ V}}{I_{\text{OL}}} = \frac{5}{16 \text{ mA}} = 312.5 \Omega$$

Nearest standard value is 330 Ω. Thus Rmin = 330 Ω can do the job: This is used in very high speed requirements, but it increases power dissipation.

### 10.4.2 CMOS to TTL Interface

To illustrate this we consider CMOS driving a low power TTL gate of 74LXX. The respective profiles are reproduced in Fig. 10.75 from Table 10.7 and Fig. 10.69.

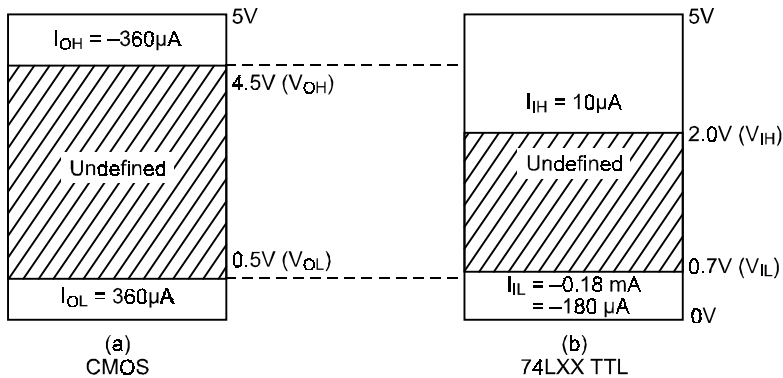


Fig. 10.75 (a) Output profile of CMOS (b) Input profile of 74LXX TTL

It is evident from Figure 10.75 that equations (10.69), (10.70) and (10.71) are well satisfied but equation 10.72 is satisfied for N = 2 only. Thus CMOS output can be connected directly to the input of 74LXX but CMOS can drive only 2 such gates. This is shown in Figure 10.76.

If we see the datas for 74LSXX in Table 10.7 we find that  $I_{\text{IL}} = -360\mu\text{A}$  which just matches the  $I_{\text{OL}}$  of CMOS. Thus CMOS output can be connected directly to only one 74LSXX TTL gate.

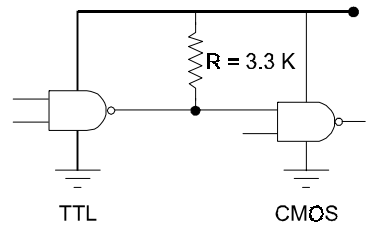


Fig. 10.74 TTL Driving CMOS

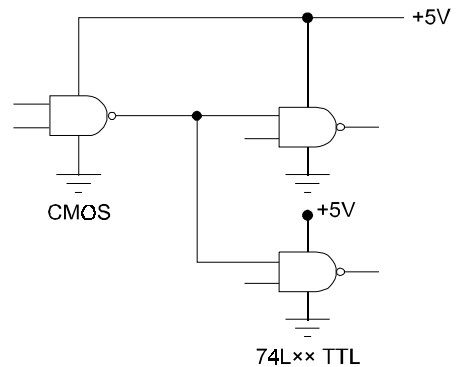


Fig. 10.76 CMOS Driving 74LXX TTL, FAN OUT = 2

If we see datas for 74XX in table 10.7 we find that  $I_{IL} = -1.6$  mA too much current for CMOS to sink. Thus the output of CMOS can not be connected directly to standard TTL. To overcome this problem CMOS buffers are used, as shown in Figure 10.77. One such IC is 74C902 which is a hex buffer. Each of these buffers have

$$I_{OL} = 3.6 \text{ mA and } I_{OH} = 800 \text{ } \mu\text{A}$$

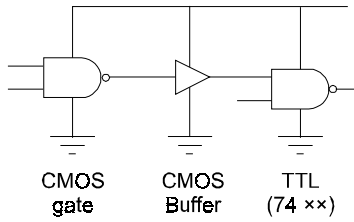


Fig. 10.77 CMOS driving standard TTL

Thus if only one buffer of 74C902 is used then with  $I_{IL} = -1.6$  mA of standard TTL and  $I_{OL} = 3.6$  mA, we can obtain a FANOUT of 2 gates. Use of remaining five buffers can increase the FANOUT further.

### 10.5 COMPARISON OF LOGIC FAMILIES

A comparison of logic families is presented in table 10.12.

Table 10.12 Comparison of logic Gates

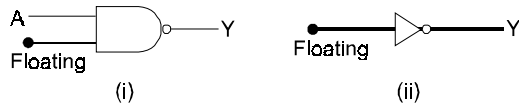
Parameter	RTL	DTL	TTL					ECL	CMOS
			Standard 74XX	High Power 74HXX	Low Power 74LXX	Schottky 74SXX	Low Power schottky 74LSXX		
Noise immunity	Average	good	very good	very good	very good	very good	very good	average	very good
Power Dissipation Per gate (mW)	12	8-12	10	22	1	19	2	40-55	0.01 static 1 mW at 1 MHz
Propagation Delay Per gate (n sec)	14	30	10	6	33	3	10	2	70
Figure of Merit (PJ)	168	300	100	132	33	57	20	95	0.7
FANOUT	5	8	10	10	20	10	20	25	Above 50
Clocking Rate (MHz)	8	72	35	50	3	125	45	Above 60	10

### 10.6 EXERCISES

1. Define the term 'Digital Circuits' and 'Logic Family'.
2. Why semiconductor devices are used in digital circuits instead of relays?
3. What is meant by integrated circuit (IC) and 'chip'?
4. (a) What is meant by intrinsic and extrinsic semiconductors?  
 (b) What type of impurities must be added to a pure semiconductor to obtain a p-type and n-type semiconductor?

5. Calculate the intrinsic carrier concentration in a standard silicon crystal at  $t = 40^\circ\text{C}$ .
6. Define the diffusion current and drift current.
7. Calculate the drift velocity of electrons and holes if mobility of electrons and holes are  $1350 \text{ cm}^2/\text{V}\cdot\text{sec}$  and  $450 \text{ cm}^2/\text{V}\cdot\text{sec}$  respectively. Assume applied electric field is  $1 \text{ V/cm}$ .
8. Explain how depletion region width is decreases when a PN junction diode is forward biased.
9. What is thermal voltage  $V_T$ . Calculate its value at room temperature?
10. Define the term storage time and transition time for PN junction diode.
11. How schottky diode improves the speed of operation?
12. Why npn transistors are preferred over *pn*p?
13. Briefly explain how an emitter resistance can result in active mode operation of a transistor.
14. Explain the switching characteristics of a BJT with the help of diagrams.
15. Why switching speed of a BJT is limited. How a schottky transistor can improve it?
16. Explain the principle of operation of MOS devices.
17. Why MOS devices are called unipolar?
18. Define the threshold voltage of a MOS device.
19. Draw and explain  $I_D$  Vs  $V_{DS}$  curve of MOS devices.
20. Why packing density of NMOS is higher than PMOS?
21. What is CMOS. Why its use in digital circuit is advantageous?
22. Why switch ON to switch OFF time of a BJT switch is larger?
23. Define the terms MSI, VLSI and ULSI.
24. Classify the logic families on the basis of polarity of charge carriers used for current conduction.
25. Define the delays  $t_{PHL}$  and  $t_{PLH}$ .
26. (a) What is meant by figure of merit of a logic gate?  
(b) For a gate value of figure of merit should be high or low. Why?
27. Explain the AC noise margin and DC noise margin. How it helps the designer.
28. Define FAN IN and FAN OUT of a logic gate.
29. What is meant by Passive pullup and Active pullup. Why they are called so?
30. What is meant by pull down?
31. What is meant by open collector output and open emitter output ? Where they are used?
32. What is the output logic performed by a RTL gates, when their output is wired?
33. Calculate the FAN OUT and NOISE Margin of RTL gates.
34. Why propagation delay of a RTL gate increases when number of load gates are increased?

35. What is current hogging in DCTL?
36. Why switching speed of HTL gate is severely affected?
37. Why average power dissipation increases when wired logic is performed?
38. What are the reasons behind limited switching speed of DTL gates ? How it can be improved?
39. What is the purpose of transistor  $Q_1$  in modified DTL gate shown in Figure 10.45?
40. Why the speed of operation of passive pullup TTL is limited?
41. (a) What happens if an input of TTL is left floating?  
(b) What will be the output of TTL gates shown below?



42. Calculate the FAN OUT of DTL NAND gate of figure 10.40 if transistor has  
(i)  $h_{FE} = 20$   
(ii)  $h_{FE} = 40$
43. When outputs of gates are wired, explain  
(a) How the speed of operation is improved?  
(b) Why the FAN OUT is reduced?
44. How FAN OUT of TTL in wired logic can be improved?
45. (a) Explain the operation of pullup transistor and phase splitter in active pullup TTL.  
(b) What is the purpose of using diode in output circuit of active pullup TTL. Comment an its PIV rating?
46. Why wired logic is not recommended for TTL with active pull up?
47. With the help of datas given in Table 10.7, calculate the FAN OUT and figure of merit of 7400, 74S00 and 74LS00.
48. Why schottky TTL is faster than standard TTL?
49. Which is the fastest logic family ? Why?
50. Why propagation delay is lower in ECL?
51. Why FAN OUT of ECL is higher?
52. What are the advantages of using difference amplifier in ECL?
53. Explain the wired logic performed by ECL gates.
54. Why transistors of high  $\beta$  and high cut in voltages are used in ECL?
55. Realize the boalean function  $y = (A + B) + (C + D) + \overline{(A + B)(C + D)}$  by using minimum number of 2 input ECL gates only and 2 input TTL gates only. Comment on the result.
56. In MOS logic why a MOS transistor is used as load instead of a resistor?
57. Why depletion mode load is preferred over enhancement mode load in MOS logic?

- 58.** Draw the circuit diagram of NMOS NOR gate and explain its operation. Also draw the truth table in which voltage levels corresponding to logic states are shown.
- 59.** Why power dissipation in an NMOS NAND gate is lower than that of NMOS NOR gate?
- 60.** With the help of circuit diagram explain CMOS inverter?
- 61.** What is meant by static and dynamic power dissipation?
- 62.** Draw and explain transfer characteristics of CMOS inverters.
- 63.** Why average power dissipation of a CMOS gate increases with frequency of operation?
- 64.** What is three state logic ? Draw the circuit diagram of three state NAND gate and explain its operation?
- 65.** Why three state logics are needed in complex digital system?
- 66.** What are the requirements of interfacing two logic gates?
- 67.** With the help of data sheet obtain interfacing of CMOS to 74LSXX TTL?
- 68.** Compare CMOS and TTL.



# MEMORY FUNDAMENTALS

---

## 11.0 INTRODUCTION

Memories are most essential part of computer and microprocessor systems. They are used to store data and programs for the computer systems. In many general-purpose applications they are also used to store the needed control information for the applications. There are various types of memory devices to accommodate various needs of data storage.

To be used with computer systems mainly we have two types of memories, semiconductor and magnetic memory. The semiconductor devices are fast storage devices and are connected directly to the CPU. For this reason they are many a times called *main memory* or *primary memory*. Since their storage capacity is relatively small they are generally used for temporary storage. Some of them are also used for permanent storage, as we will see later. The magnetic memories are used for very large data storage for long time, and are called *mass storage devices*. Since they are not connected directly to the CPU they are generally called *secondary memory*.

In the computer systems very first memory elements were magnetic and were referred as core memory. These were very bulky and needed large PCBs. In contrast semiconductor memory is compact and fast. With the latest advances in VLSI, they high-density reliable semiconductor memory devices are available at cheaper costs. Such devices internally use BJTs and MOS devices. Both read/write and read only memories are available with semiconductor technology and influenced the computational world greatly.

The advent of programmable read only memories (PROMs) made programmable logic devices (PLDs) to appear. A read only memory (ROM) consist of an array of AND and OR gates which are fixed. A PROM is one in which these arrays can be programmed but only once. The re-programmable PROMs are also available which greatly enhanced the utilization of these memory devices. The flexibility of programming the AND-OR array provided many complex systems to appear. Programmable logic array (PLA) and complex PLDs (CPLDs) are among them. Consequently this made possible the appearance of field programmable graphic array (FPGAs) and application specific integrated circuits (ASICs).

PLDs, CPLDs, PROMs, FPGAs etc have been presented earlier. In this chapter we will study the semiconductor memory devices extensively.

## 11.1 MEMORY BASICS

The ability to remember is the most important characteristics of digital circuits. The circuits or systems designed to facilitate storage of digital information are called *memory*.

In its simplest form a F/F is a basic memory device that can be used to store 1-bit of information. Any type of memory can be supposed to be composed of a no of F/F. The general information that is stored in the memory is called as *Data*. The F/F or circuit arrangement that holds a 1-bit information is conveniently called a *storage cell* or simply memory cell. The total number of bits (or storage cells) that a memory device can store is referred to as its *storage capacity*. e.g. a device that can store 1024 bits is said to have a storage capacity of 1024 bits. In memory data is always stored as a group of bits. In fact a single memory cell is rarely accessed, instead they are accessed in a group of fixed size. The number of bits in a group (of data) is called as the *Word Length* and each group is called as *Word*. e.g. In a memory if there are 10 datas each of which are a group of 4-bits, then memory is said to have ability to store 10 words of word length 4-bits. What about storage capacity of this memory device? It is simply 10 words  $\times$  4-bits (word length) = 40-bits. So,

$$\boxed{\text{Storage Capacity (in bits)} = \text{No. of Words} \times \text{Word Length}} \quad \dots(1)$$

The eq. 1 is the basic relation to determine storage capacity of a memory device. Although word length can be any no of bits (even 1-bit), now a days all have settled down to the basic memory word length as byte.

In a memory device there are several storage cells collected together to form a memory of larger capacity. In order to access them unambiguously each of the storage cell (or group of cell) is associated with a unique name, called *address* of the cell. So each cell may be called an address location or *memory location*, a place where data resides. In an analogy let us consider a residential colony where each house is a memory cell, people living in it are data stored, and house numbers are address of memory cells.

Any memory device may permit two types of operations to be performed on it, the read and the write. A *Read operation*, also called *sensing*, means detecting the status (i.e. information) of memory cell (whether it is 1 or 0) and the *write operation* means setting the status of memory cell either to 0 or to 1 according to the given data.

## 11.2 MEMORY CHARACTERISTICS

All kind of memory devices (i.e. semiconductor, magnetic, and optical memory), used to build the memory system have some functional characteristics in common. One or more of these characteristics may be used to describe the performance of the memory devices. Following are the important functional characteristics that must be considered for any description of memory

- **Storage Capacity.** It is defined as the total number of bits (or words) that a memory device can store. In general storage capacity is specified in Kilobytes (KB), Megabytes (MB), or Gigabytes (GB). Kilobit is represented as Kb. In binary the kilo means  $1024$  (not  $10^3$ ) and Mega means  $1024 \text{ K}$  (not  $10^{11}$ ). If storage capacity of device is 1KB it means it has 1024 storage locations and at each location a word of length 1 byte (i.e. 8-bits) can be stored. So the total capacity in bits would be  $1024 \times 8 = 8192$  bits or 8 Kb.
- **Cost.** The cost of the memory can be defined as the cost per bit. It is expected that the cost/bit should be as low as possible. If  $S$  is the storage capacity,  $C$  is the total cost of memory then the cost per bit  $Cp/b$  can be defined as
 
$$Cp/b = [\text{total cost (C)}/\text{Storage capacity (S)}]$$

- **Memory Access Time.** The performance of a memory device is primarily determined by the rate at which the information can be read or write into the memory.

The average time required to read a fixed amount of information from the selected memory location is termed as *Read Access Time* or *Access Time* and is denoted as ' $t_A$ '. In precise words, it is average time delay after memory address is issued till the data appears at output. The access time should be kept as low as possible. Similarly the average time required to store a fixed amount of information at the selected memory location is termed as *Write Access Time*.

$$\text{Read Access Time} > \text{Write Access Time}$$

The average rate at which memory can be accessed is defined as the *Access Rate* ' $b_A$ ' of a memory and is measured in words/sec.

$$\text{Access Rate } (b_A) = 1/t_A$$

- **Memory Accessing Modes.** It is defined as the way or sequence in which the information is accessed in the memory. The two types of access modes are
  - *Random Access Mode*
  - *Serial Access Mode*

If the memory cells can be accessed in any order, then the access time is independent of the position of cell. This type of access mode is referred as random access mode, and the memories having such access mode are called *Random Access Memory*. In such memories any storage cell (or location) can be reached in a fixed amount of time and each storage cell can be accessed independently of the positions of other cells. Obviously this type of memories are faster. Semiconductor memories are of this type.

If the memory cells can be accessed one after another or only in certain predetermined sequences, the access time is dependent of position of cell. Thus the access time will be different for different cell locations. This is called serial access mode and the memories having such access mode are called *Serial Access Memory*. Magnetic memories are of this type.

- **Memory Cycle Time and Bandwidth.** The minimum time that must elapse between the initiation of two consecutive memory accesses is defined as the *Memory Cycle Time*  $t_C$ . The cycle time  $t_C$  can be greater than the access time  $t_A$ . This is because after completion of one memory access some memory devices need a small amount of time,  $t_{\text{DELAY}}$  to make them ready for next access. In general

$$t_C = t_A + t_{\text{DELAY}} \quad (t_{\text{DELAY}} \text{ is additional delay due to the physical characteristics of memory})$$

Note that  $t_{\text{DELAY}}$  may not be present in all type of memories.

Maximum amount of information that can be transferred to or from memory every second is referred to as the maximum *Data Transfer Rate* in words/sec or the *Bandwidth* ' $b_C$ ', defined as

$$\text{Bandwidth } (b_C) = 1/t_C$$

- **Non Destructive Read Out (NDRO).** It is required that the read process should create a copy (i.e. sense) of the data stored and must not disturb the data. Such a reading is called *Non Destructive Read Out* (NDRO) and is expected from all kind of memory.

In some memories the reading may destroy the data stored at a location. Such a reading is called *Destructive Read Out* (DRO). In such cases the read operation must be followed by a write operation to restore the original state of memory. Obviously such memories would be slower ones.

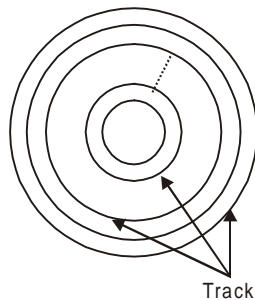
- **Permanence of Storage.** It is concerned with how long a stored information can be retained when
  - (a) *Power is Switched off* If the stored information does not vanish with power off, the memory is referred as *NON VOLATILE* memory. But if the stored information vanishes completely at power off the memory is called as *VOLATILE* memory.
  - (b) *Power is NOT Switched off.* If the data can be retained indefinitely the memory is called as *STATIC* memory. If it can be retained for some definite small amount of time, the memory is called as *DYNAMIC* memory. In the later case the data must be rewritten periodically, this is called refreshing.
- **Alterability.** Alterability is concerned to the capability of memory devices with which the stored data can be changed or updated when they are in use (i.e. online). Memories, which can be altered online, are referred as *Read-Write Memories*. Memories, which can not be altered online (if at all they can be altered offline) are called erasable read only memory devices. However some memory devices can not be altered even when they are off line and are called nonerasable read only memory devices.

### 11.3 MASS STORAGE DEVICES

The devices used to store high volumes of data are referred as mass storage devices. The magnetic memories and Optical memories (CD-ROMs) are such memory devices. The magnetic memories are used for very large data storage for long time, but due to their lower speed they are not connected directly to the CPU. Due to this they are generally called *secondary memory* or *backup storage device*. Optical memories (CD-ROMs) are another type of mass storage device. The optical disks are popular due to large storage capacity, transportability and reliability.

#### 11.3.1 Magnetic Memory

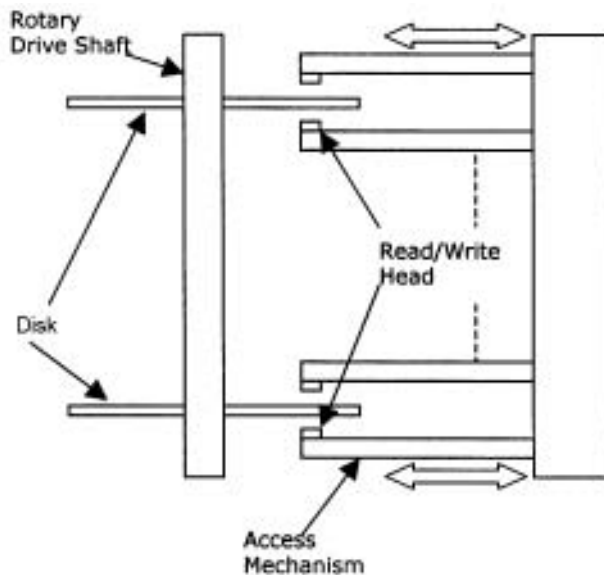
In magnetic memory devices some sort of magnetic coating (usually ferric oxide) runs throughout the base. On this coating direction of magnetic flux or direction of magnetization can be used to store 1 or 0. These devices are called *Serial Access Memory* (SAM) devices, because we can not reach the data directly and to reach a data stored we must go one after another. For example consider an audio tape and if you want to reach song number 3 you must traverse through song numbers 0, 1 and 2.



**Fig. 11.1** Organization of tracks

Hard disks and floppy disk are common type of magnetic memories used in computers. The storage media of magnetic disks are made up of aluminum or plastic base with a thin coating magnetic on its surface where the data are actually stored.

In a **Floppy Disk**, as shown in Fig. 11.1, the whole media is divided into a number of concentric circles, called track. Each track is further divided into a number of sectors. In these sectors bytes of data block are stored. To reach to a stored data first a particular track is selected, and then particular sector is reached on this track, ofcourse starting from the very first sector on this track. To reach to required sector an external rotating mechanism is employed, thus time delay to reach particular sector is called *rotational delay* or *latency time*. The time delay to select a particular track is called *Seek time*. Now a day the 3.5-inch floppy disk of storage capacity 1.44-MB is popular. It contains a total of 40 tracks where innermost track is numbered as Track '0' and outermost track is numbered Track '39'. For each of the data access search starts from 1st sector of Track '0'. The main advantage of using floppy disk is portability, but its use is restricted by its small storage capacity.



**Fig. 11.2** Structure of HARD DISKS

A **Hard Disk** can be supposed to have collection of such circular disk surfaces kept in a cylinder on a common shaft, called spindle for rotation. A simple organization of hard disk storage unit is shown in Fig. 11.2. The spindle rotates the disk the disk surfaces at higher speeds, making data access faster than the floppy disks. On each disk surface there are several hundreds of tracks arranged in concentric circle as shown in Fig. 11.1. Each of the tracks is further divided into a number of sectors where data actually resides. Each of the disk surfaces is provided with the separate Read/Write head that may be connected to form a Read/write arm.

During the access operations the disks are rotated continuously by the spindle, at a constant speed. The Read/Write arm the moves in a fixed linear path to select a particular set of tracks and the sectors on these tracks. The disk monitoring system determines which disk surface has to be accessed to find the target data. In comparison with floppy disks the construction of hard disks are much more robust and reliable but they are not portable.

In both the magnetic disks the digital information can be *stored* on disk surface by applying a current pulse to magnetization coil present in Read/Write head. This causes a change in magnetization in the area under the read/write head. The direction of magnetization will be parallel to the direction of applied field. To *read* the magnetically stored information on the disk the Read/Write head has to sense the state of magnetization resulting from the area under the head. Movement of magnetic disk causes change in magnetic field, which induces a voltage in the coil of head. The polarity of induced voltage is monitored to obtain the Magnetization State, which in turns gives the stored binary bit.

The principal disadvantage of magnetic disk memory is relatively slower speed that is attributed to slow rotational mechanism, seek and latency time and the fact that data transfer through such devices are serial. Also presence of mechanically moving parts makes magnetic memory unreliable.

But their advantage lies in two great facts that it offers very large-scale data storage for long time at very low cost per bit. There are other kind of magnetic memory devices also, namely ferrite core memory, magnetic tape memory, magnetic bubble memories etc.

### 11.3.2 Optical Memory

Optical memories usually takes the form of optical disks which resembles magnetic disks in that they store binary information in concentric tracks on a mechanically rotated disks. The read & write operations are done with the help of Lasers, capable of producing a light spot of the order of 1 micron.

Optical discs are available in many formats: CD-ROM (compact disk-read only memory), CD-R (compact disk-recordable), and CD-RW (compact disk-rewritable). Newer high capacity version of optical disks is DVDs, the Digital Versatile Disk. Similar to the CDs, the DVDs also have different formats as DVD-ROM, DVD-R, and DVD-RW. Most of the CDs can store upto 650 MB where as the simple single layered DVD can store 4.7 GB of data.

The CD-ROM is manufactured using carefully prepared glass master. The master is pressed into injection molded polycarbonate plastic forming the CD-ROM. The resulting CD-ROM contains several small pits and lands (no pits) on the various tracks that can be processed optically. When a computer system want to read the disk, a Laser beam is aimed on the disk from the bottom and the reflection from the pits & lands are interpreted by the photo detector as logic 0s and 1s.

For the purpose of permanent storage (archival storage) the CD-Rs are very popular, and many a times known as WORM (*write once read many*) storage devices. The information can be recorded on CD-R (commonly known as CD burning) by using CD-writers. During the burning process a laser heats gold reflective layer, and a dye layer in the CD-R causing it to have a dull appearance. When read by the CD-R drive the dark burnt areas (like pits) reflect less light. The CD-R reader interprets the shiny areas (lands) and the dull areas (burned) as logic 0s & 1s.

The CD-ROM and DVD-ROM both are manufactured by using the same technology and read data from a spiral track of pits and lands. However the DVD-ROM has a greater storage capacity. It is because the pits and lands on the DVD-ROM are much smaller and more closely packed, thus allowing more information per track. The tracks also are closely spaced allowing more tracks per disk. The DVD-ROMs can store 4.7 GB (single sided layer), 9.4 GB (double sided layer), or 17 GB (double sided double layered), where as the CD-ROMs can store only upto 650 MB.



The CDs are associated with all kind archival storage and have no special standards dictated for the storage of specific type of data on the media. However the DVDs are most commonly associated with the video productions. DVD-video standards are used when disk holds only audio/video (such as movies). The DVD-ROM standards are used when the DVDs are used for data storage as with a computer system.

The *data transfer rate* of a CD-ROM drive is indicated as 1x, 2x, 4x, 16x, 32x etc. A CD-ROM drive with a designation of 1x would have a max data transfer rate of 150 Kbytes/sec. Therefore a 4x drive would have a data transfer speed of  $150 \text{ Kbytes/sec} \times 4 = 1100 \text{ Kbytes/sec}$ . These data transfer rates are theoretical and the actual data transfer rates are usually lesser. The DVD-ROMs can provide data transfer rates of 1.38 Mbytes/sec, which is approximately same as the speed of 9x CD-ROM.

The advantages of Optical disks are huge amount of data storage, low cost, easily transferable, a few mechanical parts results in reliability, and high speed. The main disadvantage could be the complexity of Read/Write head.

Although lot more pages can be shed to explain the architecture and characteristics of magnetic & optical memories, but a detailed discussion is beyond the scope of this text. However interested readers can refer to "*Hayes, J.P.: Computer Architecture*". A glance over magnetic & optical memory was presented to maintain the completeness of the topic. The semiconductor memory is the main consideration of this text so until & unless specified memory must be read as semiconductor memory throughout this text.

## 11.4 SEMICONDUCTOR MEMORY

Numerous developments in semiconductor technology has emerged into large numbers of LSI and MSI memory devices, called memory *chips*. Besides being faster and compatible with CPU they are economical also. A semiconductor memory is organized as a rectangular array (preferably square array) of storage cells that are integrated on a silicon wafer and are available in DIP packages. Due to this organization any memory cell can be accessed randomly, thus all the semiconductor memories are called *Random Access Memory* (RAM). The basic memory cell may be a flip-flop, or a charge storing element like capacitor that is said to hold logic 1 when charged and logic 0 when discharged. The type of transistors e.g. bipolar, or MOS, or CMOS, used to form a memory cell dictates the storage capacity and speed of a memory chip.

### 11.4.1 Basic Memory Unit

A simple block diagram of a memory unit can be drawn as shown in Fig. 11.3. It contains data bus, address bus and control signals. A *bus* is a group of parallel conductors whose job is to carry binary information. Each of the conductors carries 1-bit information. The number of bits that a memory data bus carries simultaneously is called *memory bus width*. Usually, but not necessarily, the memory word length and memory bus widths are same.

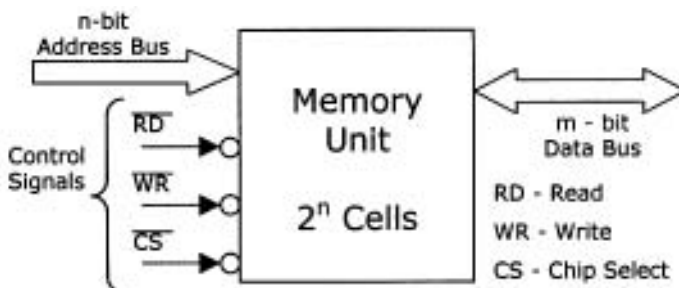


Fig. 11.3 Block Diagram of Memory Unit

The m-bit data bus is used to transfer data to and from the memory. The n-bit address bus carries the address of memory locations. An n-bit address bus can access upto  $2^n$  storage cells i.e. storage capacity is  $2^n$  bits as each cell can store 1-bit. The Read (RD) & Write (WR) signals specify the operation to be performed. As shown the two signals are active low i.e. they are activated when logic '0'. At any time either of the two signals can be activated. In some memory chips read & write controls are available at one signal line. Chip select signal (CS) (sometimes labeled as enable EN also) is used to enable/disable the chip in multiple memory system. It is also a logic low signal. There may be multiple enable signals in a single chip that must be activated simultaneously to enable the chip.

### 11.4.2 Basic Memory Organization

Basic organization of semiconductor memory is shown in Fig. 11.4 (a). It consists of a storage cell array, Read/Write control circuit along with control signals, an address decoder fed by a register called memory address register, and a buffer register through which data transfer is carried out. In a semiconductor memory storage cells are organized as a rectangular array. The address bus is connected to internal address decoder through the address register. The RD and WR control line specifies the type of access: If  $\overline{RD} = 0$  and  $\overline{WR} = 1$  then read operation.

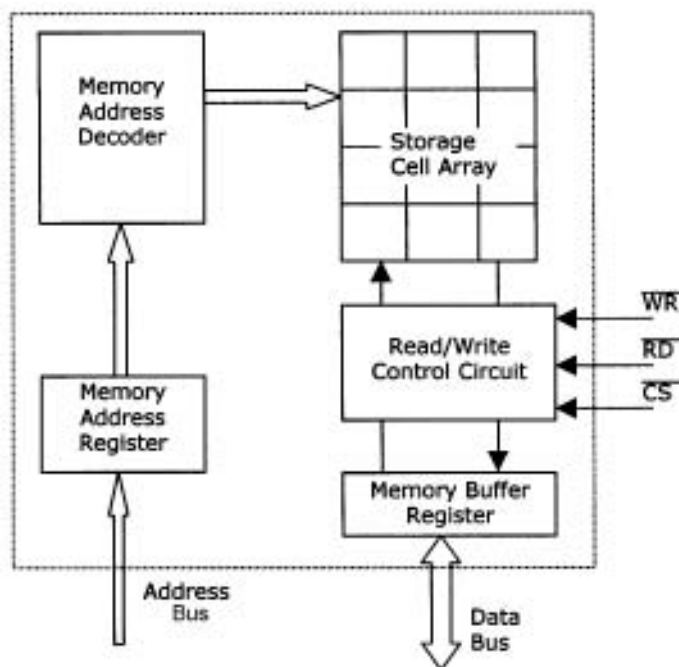


Fig. 11.4 (a) Basic Memory Organization

If  $\overline{RD} = 1$  and  $\overline{WR} = 0$  then write operation. To access memory, address of the required cell is issued to the memory address register through the address bus. The address decoder selects the particular location in memory array. If read operation is requested then the content of selected location is copied to the memory buffer register which transfer it to the data bus. If write operation is requested then content of data bus is brought to buffer register and then transferred to the selected memory location. Memory buffer register contains bilateral three state buffers, one for each bit of data bus. By combining two three state buffers a



1-bit bi-directional input output line is formed. This is shown in Fig. 11.4 (b). The enable or disable signals for the bilateral buffers can be generated through the realization shown in Fig. 11.4 (c). These signals are given to all the bilateral buffers of buffer register. Note that if both RD & WR are activated, or CS is inactivated, arrangement in Fig. 11.4 (c) will generate disable signals for both the buffers i.e. both TSL1 & TSL2 are disabled and data bus enters into high impedance state.

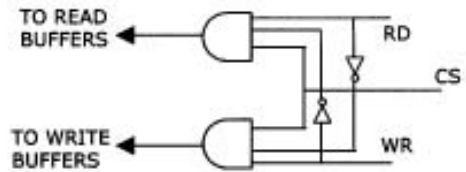
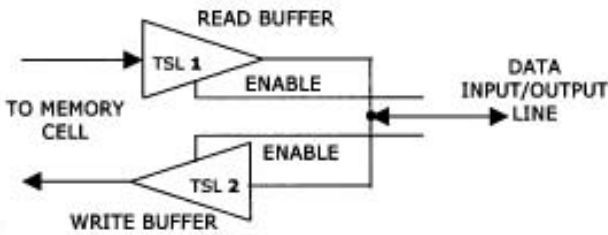


Fig. 11.4 (b) Bilateral Memory Bus Buffer

Fig. 11.4 (c) Generation of Enable/Disable Signals for Buffers

### 11.4.3 Cell Organization (Memory Addressing)

Memory addressing is concerned with the selection of one particular memory cell from storage cell array. To facilitate selection, the memory cells are organized as rectangular array of m-rows and n-columns as shown in Fig. 11.5(a).

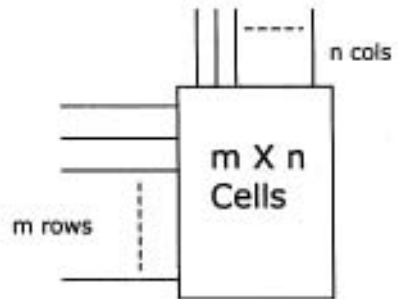
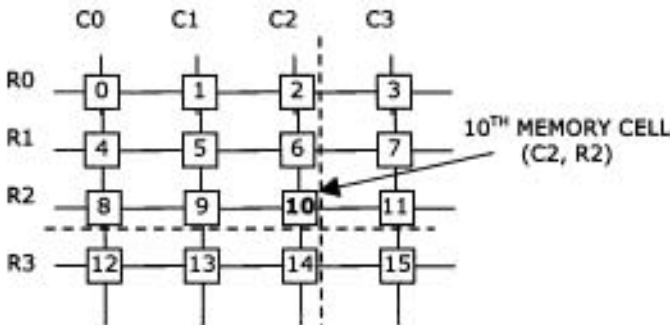


Fig. 11.5 (a) An Array of 4x4 Cells

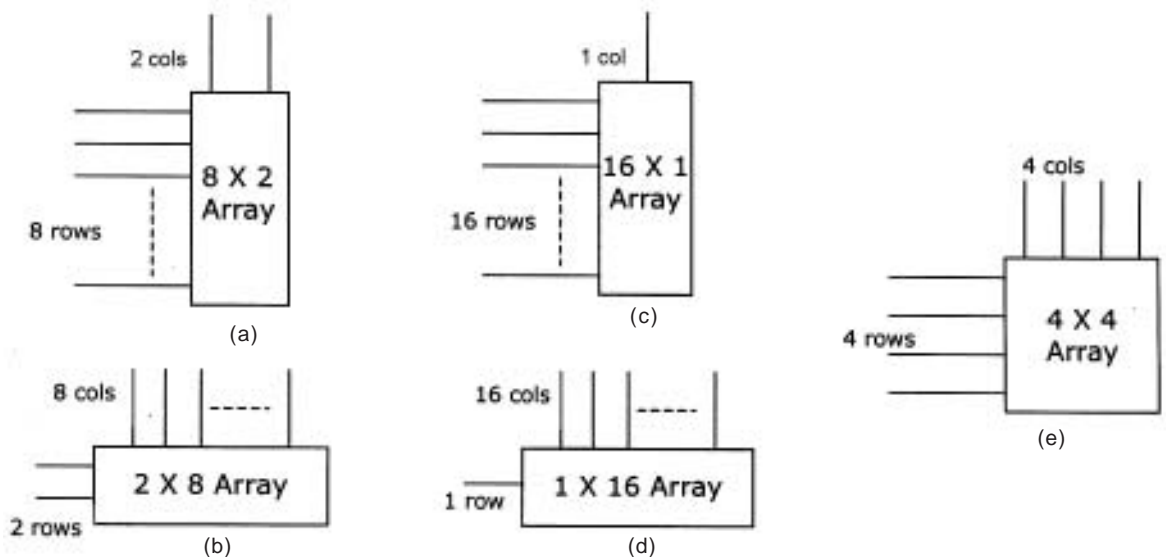
Fig. 11.5 (b) An Array of m x n Cells

Fig. 11.5(a) shows simplified diagram of memory cells array that have 4-rows and 4-columns, thus containing a total of 16 memory cells. The control circuitry associated with memory allows only one row and one column to be activated at a time. In order to select a particular cell we need to specify the appropriate row and column number. Let us consider the selection of 10th cell. It requires column C2 and row R2 to be activated and the cell at the intersection of C2 and R2 is selected which is 10th cell. Dashed lines in Fig. 11.5 (a) show this. So the 10th cell can be designated as the cell C2, R2 (or 2, 2 or 22). This designation is defined as the *address* of the 10th cell. Collectively the lines corresponding to rows and columns are called *address lines*. In a memory array each cell will have a similar type of unique address. Hence any cell can be accessed, just by specifying the row and column number (or address) associated with it. Since any cell can be accessed *randomly*, it is named *Random Access Memory*. Fig. 11.5 (b) shows a simple block representation of a rectangular

array that we will use in following subsections to understand some important facts. If in Fig. 11.5 (b)  $m = 4$  and  $n = 4$  then it represents the array shown in Fig. 11.5 (a).

#### 11.4.3.1 Matrix Addressing

We know that to facilitate selection, the memory cells are organized as rectangular array of  $m$ -rows and  $n$ -columns. When rows and columns are made equal i.e.  $m = n$ , then the array becomes a *Square Array* of capacity  $n \times n = n^2$ . This type of arrangement is called *MATRIX Addressing*. This addressing have the advantage of requiring fewer number of address lines than the number of address lines required by any other rectangular arrangement. This fact is explained below by using an array of 16-cells (the one shown in Fig. 11.5 (a)). Observe carefully the different array arrangements shown in Fig. 11.6.



**Fig. 11.6** Different Array Arrangements for 16 memory cells.

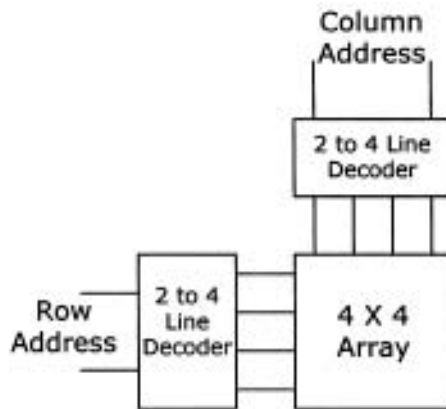
Fig. 11.6 (a) and 11.6 (b) are equivalent array arrangement as one refers to 8-rows and 2-columns and other refers to 2-rows and 8-columns. Both require 10 address lines (8-rows+2-columns) to select any of the memory cells. Similarly Fig. 11.6 (c) and 11.6 (d) are equivalent array arrangement and require 17 address lines. Infact 111 lines will suffice, as there is only one column/row. Fig. 11.6 (e) is square array arrangement with equal number of rows and columns and require 8 address lines. So in order to reduce the number of address lines square array arrangement is best.

The arrangement in Fig. 11.6 (e) is referred as *Matrix addressing* as defined earlier. In contrast the  $16 \times 1$  array arrangement as in Fig. 11.6 (c) is called *Linear Addressing*. It is because there is only one column and to select any of the cells, one needs to specify the row only.

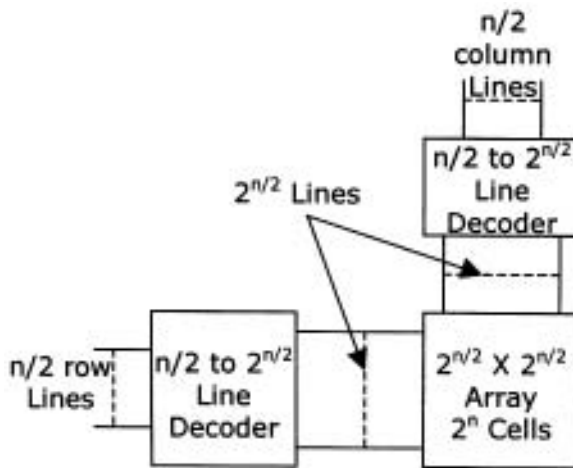
#### 11.4.3.2 The Address Decoding

The address decoding is concerned with the activation of particular row and column to select a memory cell. Here our concern is to see this process exclusively for matrix addressing. Let us reconsider the Fig. 11.6 (e) that requires 4-rows and 4-columns. If we use two 2 to 4 line decoders then the address line requirements will be reduced further, as shown in Fig. 11.7 (a). The decoders will activate only one of the row & column lines out of 4 lines. As

evident from figure now the total number of required address lines are 4 only! The activation of a particular row and column line depends upon the row address and column address. Any column from C0 to C3 can be activated by specifying it as a 2-bit binary number (00 to 11) at column address lines. Similarly specifying a 2-bit binary number at row address lines can activate any of the four rows. Thus a total of 4-bit address can access any memory cell unambiguously. Since 4-bit address representation allows  $2^4 = 16$  unique addresses, we can say that an  $n$ -bit address can be used to define a square array of  $2^n$  memory cells.



**Fig. 11.7 (a)** Square Array of 16 cells with Row and column Address Decoders



**Fig. 11.7 (b)** Square Array of  $2^n$  cells with Row and Column Address Decoders

Fig. 11.7 (b) shows a generalized arrangement of square array with address decoders. The array is organized as  $n/2$  column address lines and  $n/2$  row address lines thus making a total of  $n$ -bit address lines. It should be noticed that in such arrangements the value of  $n$  should be an even integer (2, 4, 6, 8 ...). Since at the output of each of the decoder we have  $2^{n/2}$  lines, the number of cells (or capacity) in this array would be  $2^{n/2} \times 2^{n/2} = 2^n$ . This is exactly the reason why commercially available memories have capacity equal to some integer power of 2 i.e.  $2^{10}=1024$ ,  $2^{12} = 4096$  etc. The above discussion concludes that if  $n$  is the number of address lines then

Number of storage cells =  $2^n$

In general

$2^n \geq$  Number of Storage Locations  
 Or  
 $2^n \geq$  Number of Words  
 Where  $n =$  No. of address lines (An Integer) ... (2)

**Example.** In the Fig. 11.8(a) explain how memory cell C2, R2 can be accessed? Given that the row & column numbers are in true order and activated output line have status logic high.

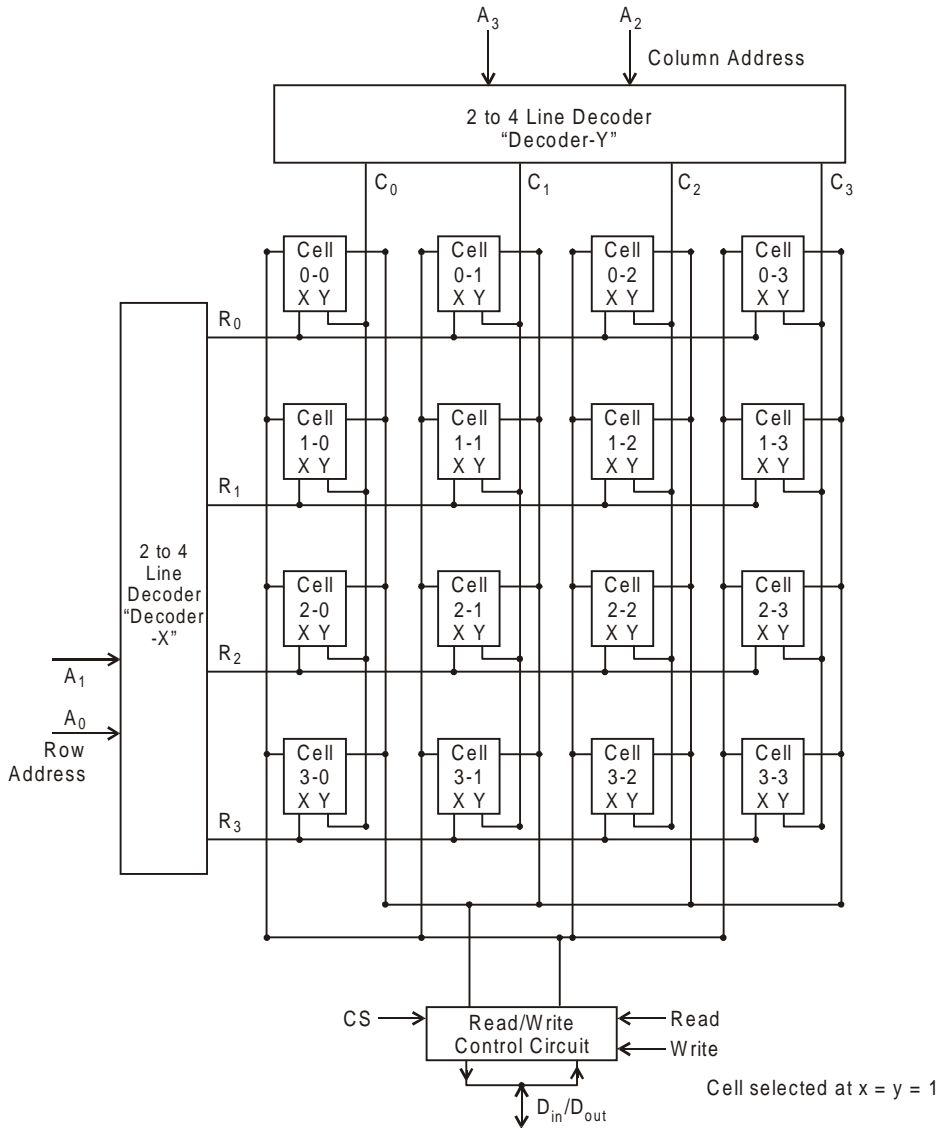


Fig. 11.8 (a) 4x4 Array Organized as 16 x 1 memory

**Solution.** Figure shows an array of  $4 \times 4$  having storage capacity of 16-bits. Issuing its address can access the required cell. Since row & column numbers are in true order, the binary equivalent of these numbers will give required memory address.

Memory cell to be accessed C2, R2

Column C2 specified in binary = 10 =  $A_3 A_2$

Row R2 specified in binary = 10 =  $A_1 A_0$

So the complete address  $A_3 A_2 A_1 A_0 = 1010$

Upon receiving the address the address decoders 'X' and 'Y' will activate row R2 and column C2, respectively. All other rows and columns will be deactivated at this time i.e.  $R_3 = R_1 = R_0 = 0$  and  $C_3 = C_1 = C_0 = 0$ . Now referring to Fig. 11.8(a) we can see that only the cell 2-2 has got both  $x$  and  $y$  input high which is present at the intersection of R2 and C2. Thus the cell C2, R2 is selected for access.

#### 11.4.4 Organizing Word Lengths (Different Memory Organization)

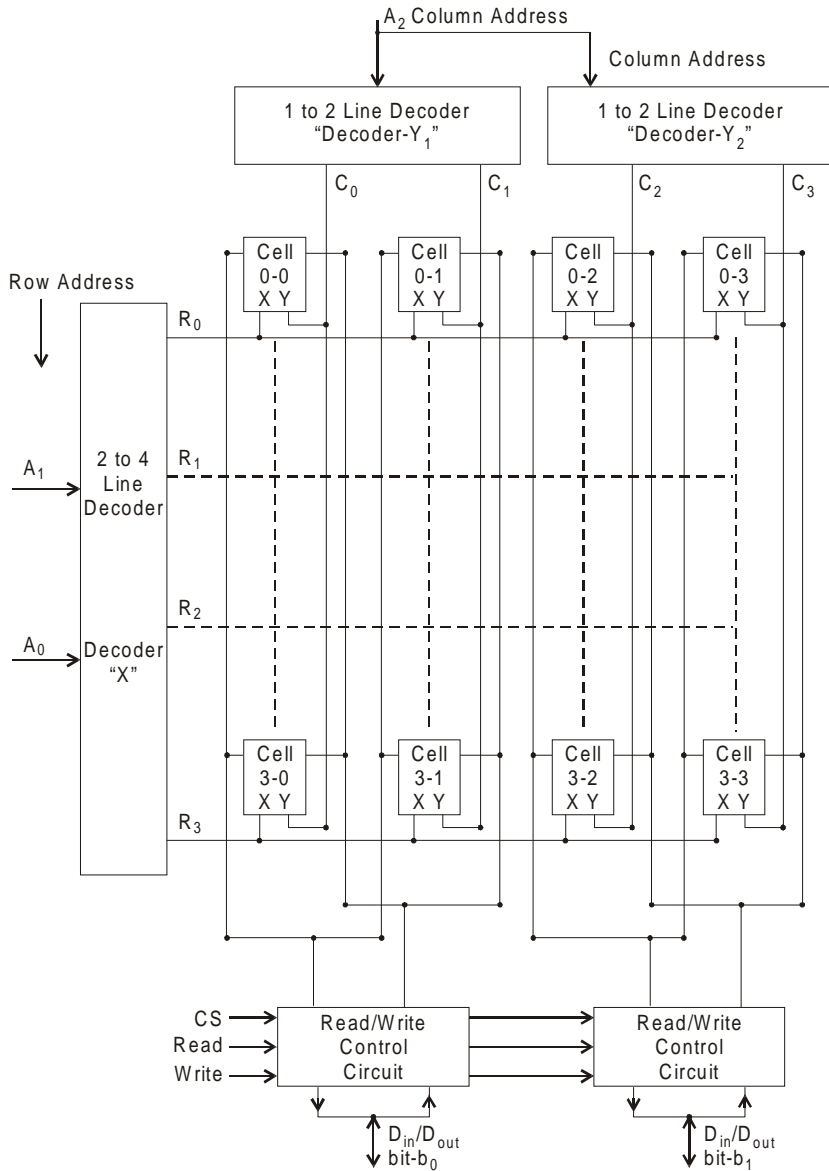
So far we have discussed the memories that can access a single storage cell or 1-bit at a time. In computer systems requirement is to access a group of bits at a time. In this section our prime concern is to discuss the organization that can access a group of bits. More over this section also deals with different types of cell selection mechanism that can be used conveniently.

To begin with let us consider a  $4 \times 4$ -memory array organized as  $16 \times 1$  memory i.e. 16 words each of length 1-bit, as shown in Fig. 11.8(a).

Here two 2 to 4 line decoders, namely decoder-'X' & decoder-'Y', are used to decode row and column addresses respectively. A cell in this array can be selected only when both  $x$  and  $y$  inputs are activated i.e., at  $x = y = 1$ .

The same array can be organized as  $8 \times 2$  memory i.e., 8 words each of length 2-bits. Still the storage capacity remains 16-bits. This is shown in Fig. 11.8 (b). In this case decoder-Y is replaced by two 1 to 2 line decoders, Y1 and Y2. Both Y1 and Y2 are feeded by the same address line ( $A_2$ ) and work simultaneously. Thus at any time two columns are activated as per the truth table given in table 1. Whenever any row is activated,  $x$  inputs of all the 4-cells in that row will be activated. Since two columns are activated at a time, two of the memory cells in the activated row will get their  $y$  input activated. Thus two memory cells have both  $x$  and  $y$  inputs activated i.e. two cells are selected. Alternately with this organization we can access a group of 2-bits at a time.

The array of Fig. 11.8 (a) can be organized as  $4 \times 4$  memory i.e. 4 words each of length 4-bits. Still the storage capacity remains 16-bits. This is shown in Fig. 11.8 (c). To do so, decoder-Y is removed and cells are connected only to decoder-X. Thus to select memory cells only  $x$  inputs have to be activated. Whenever a row is activated  $x$  inputs of all the 4-cells are activated i.e. all the 4-cells of that row are selected. In other words this organization can access a group of 4-bits at a time. Infact this addressing is an example of linear addressing because only rows have to be activated and columns are always selected.



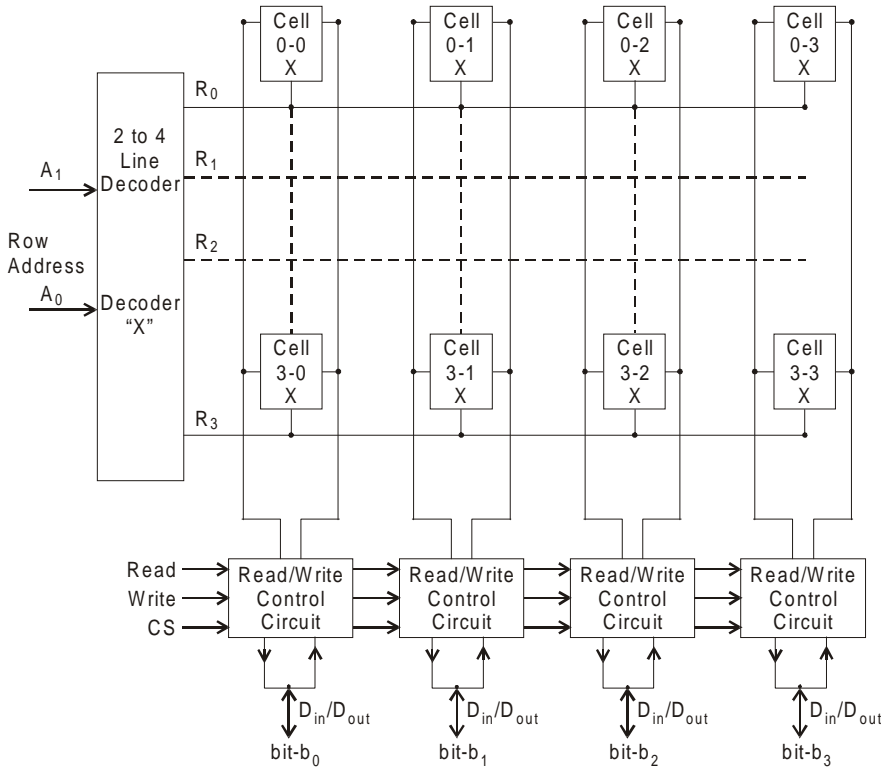
**Fig. 11.8 (b)** 4x4 Array organized as 8x2 memory

**Table 1 : Truth Table for Fig. 11.8(b)**

Column Address	Selected	
" $A_2$ "	Columns	
0	$C_0$	$C_2$
1	$C_1$	$C_3$

Notice the reduction in number of address lines in each case. In Fig. 11.8 (a) 4-address lines were used where as in Fig. 11.8 (b) 3-address lines and in Fig. 11.8 (c) only 2 address lines were used. It is because by increasing the word length the total numbers of words are

reduced. When more than one cell (*i.e.*, group of cells) is accessed at a time than one storage location indicates a group of cells. Recall that a group of bits (or cells) is nothing but a word. It is a simple consequence of equation (1) and (2).



**Fig. 11.8** (c) 4x4 Array Organized as 4x4 (4 words of length 4-bits) memory

by equation (1) we get

$$\boxed{\text{No. of words} = \frac{\text{Storage Capacity (in bits)}}{\text{Word length (in bits)}}} \quad \dots(3)$$

but by equation (2) we get  $2^n \geq \text{No. of Words}$

where  $n$  = no. of address lines

Putting it into equation (3) results

$$\boxed{2^n \geq \frac{\text{Storage Capacity (in bits)}}{\text{Word length (in bits)}}} \quad \dots(4)$$

The equation (4) clearly shows that number of address line 'n' is inversely proportional to word length. Thus increasing word length will cause a reduction in no of address lines.

**Example.** A memory has cell array of  $32 \times 32$ . What would be the required number of address lines if the word length is

- (i) 1-bit      (ii) 2-bits      (iii) 4-bits      (iv) 8-bits      (v) 16-bits

Also find out the number of words in each case. What is the storage capacity of this memory in bits and in bytes?

**Solution.** We know that since a single cell stores 1-bit, total number of cells constitutes the storage capacity of this memory.

Given array is  $32 \times 32$  so,

$$\text{No. of storage cells} = 32 \times 32 = 1024$$

$$\text{Storage capacity} = 1024 \text{ bits or } 1 \text{ Kbits} = 128 \text{ bytes}$$

Number of address lines can be calculated by equation 4

when word length is 1-bit

$$2^n \geq 1024/1 \Rightarrow 2^n = 1024/1 \Rightarrow 2^n = 1024 \Rightarrow n = 10 \text{ lines}$$

when word length is 2-bits

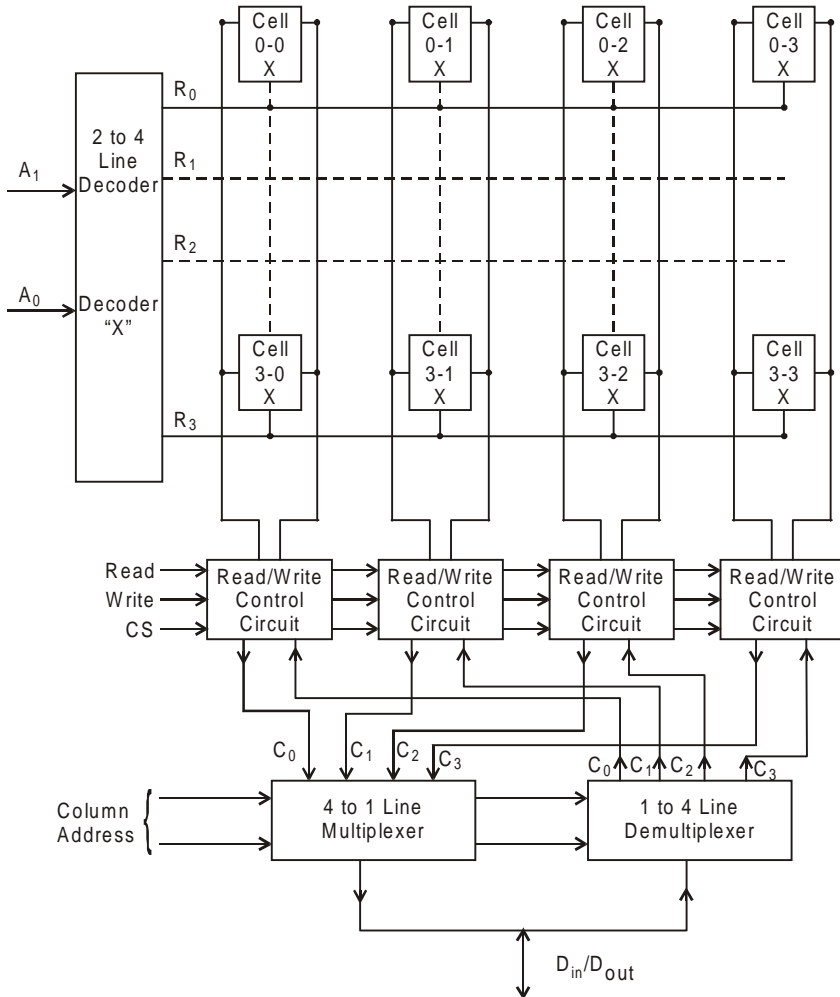
$$2^n \geq 1024/2 \Rightarrow 2^n = 1024/2 \Rightarrow 2^n = 512 \Rightarrow n = 9 \text{ lines}$$

Similarly for other word lengths address lines required can be calculated.

The number of words can be calculated by equation 2 given as

$$2^n = \text{No. of Words} \Rightarrow 2^n = \text{No. of Words}$$

The value of  $2^n$  for each case is calculated already. So 1024 words can be accessed when word length is 1-bit, 512 words can be accessed when word length is 2-bit, and so on.



**Fig. 11.9** As  $16 \times 1$  Memory Organization



A slightly different organization of  $4 \times 4$  memory cells as  $16 \times 1$  memory is shown in Fig. 11.9. Compare the Fig. 11.9 with Fig. 11.8 (a). Here instead of using a column decoder, one 4 to 1 line MUX and a 1 to 4 line DeMUX is used. Whenever any row is activated all the 4-cells are selected. If read operation is requested, the multiplexer connects one of the selected cells to  $D_{in}/D_{out}$  output line. If write operation is requested, then demultiplexer connects its input line (i.e.  $D_{in}/D_{out}$  line) to one of its output line which is connected to one of the cells which were selected by activating the row. The select input of both MUX & DeMUX are fedded with the column address lines.

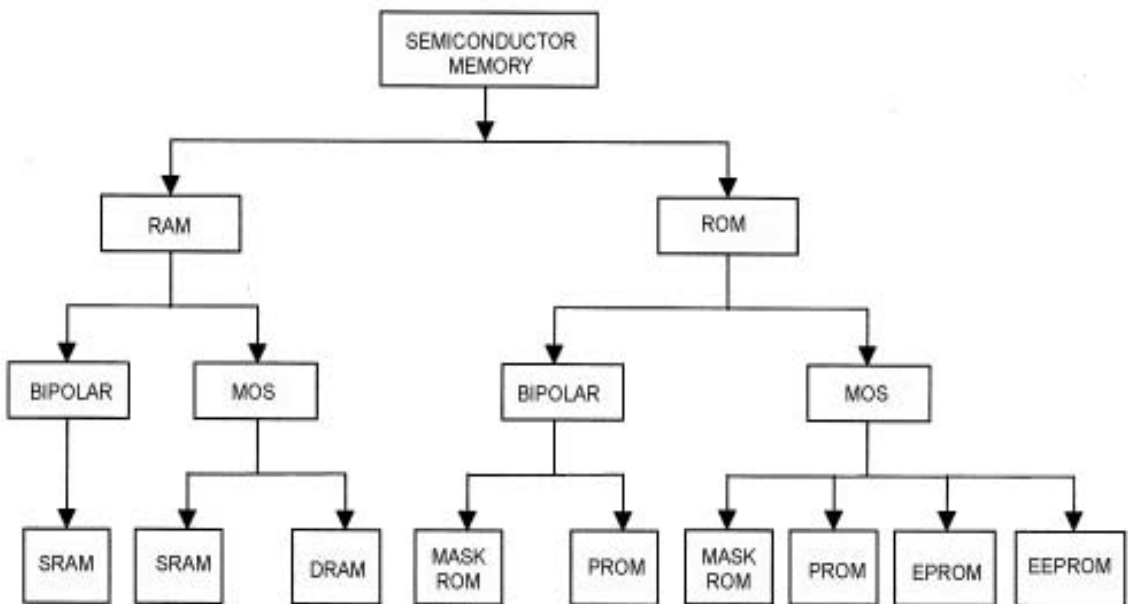
### 11.4.5 Classification of Semiconductor Memory

The semiconductor memory devices can be categorized in several ways according to their functional and architectural characteristics. Fig. 11.10 shows the most general classification. As shown they fall in two general categories.

- Read Write Memory or Random Access Memory (RAM)
- Read Only Memory (ROM)

Read Write Memories or RAMs are those memories, which allows both read & write operation online. They are used in applications where data has to change continuously. They are also used for temporary storage of data. Since each cell is a flip-flop a power off means loss of data. Thus the RAMs are *Volatle Storage Devices*.

ROMs are those memory devices, which allows only read operation online and there is no write mode. ROMs are needed in applications where data does not change e.g. monitor programs, mathematical constants etc. Since the data is permanently stored in ROM, power failure does not result in loss of data. Hence ROMs are *Non Volatile Storage Devices*. In fact ROM is also a random access memory, but in practice saying RAM refers to read write memory.



**Fig. 11.10** Classification of Semiconductor Memories

Both the RAM & ROM are further divided into other subcategories, as shown by figure. We will define each of them in this section whereas detailed discussions proceed in the subsequent sections.

Primarily both RAMs & ROMs are classified as *Bipolar* and *MOS* memory depending upon the type of transistors used to construct the individual cell. If the cells are formed by using bipolar (or MOS) transistors the chip is called *bipolar (or MOS) memory chip*. High-speed operation is possible with bipolar chips but their storage capacity is lesser. However economical MOS/CMOS chips have greater storage capacity, reduced size, and lesser power requirements.

*Static RAM* (SRAM) is a read write memory that uses two cross-coupled transistor (either bipolar or MOS) working as Flip-Flop to form the basic memory cell. SRAM holds the stored data indefinitely if power is on, hence the name static. SRAMs are very high speed but costly memories. A *Non Volatile RAM* (NVRAM) is formed by using battery backed up SRAM.

*Dynamic RAM* (DRAM) is a read write memory that uses a capacitor in conjunction with a MOS transistor to store 1-bit. Since the capacitors are leaky, they cannot hold the charges permanently. So they must be recharged (refreshed) periodically to hold the stored data, hence the name dynamic. They are slower than SRAMs but their storage capacity is greater.

The ROMs are categorized according to the data storing process. The mechanism that stores the data into ROM is called as programming mechanism.

A simple ROM is programmed during the manufacturing process according to the data specified by the user. Such a ROM is referred as *Mask ROM* or *Mask Programmable ROM*.

A PROM (*Programmable ROM*) is a type of ROM that can be programmed by the users in the field. They are also referred as *Field Programmable ROM*. But once programmed they can not be altered.

An EPROM (*Erasable Programmable ROM*) is another type of field programmable ROM that can be reprogrammed by the users after erasing the stored data. The erase operation is performed optically which erases all the data stored in EPROM. Selective erase i.e. erasing just a segment of data is not possible with EPROMs.

An EEPROM (*Electrically Erasable Programmable ROM*) is another type of field programmable ROM that can be programmed by the users as often as required but after erasing the stored data. The erase operation is performed electrically which allows selective erase.

#### 11.4.6 Semiconductor Memory Timing

In this section we are concerned with timing sequences which must be considered for any type of memory operation. Although these timing requirements may not be precisely applicable to all types of memories, but it is useful to grasp the concept. The time duration and sequence in which various signals have to be activated is of great importance. Manufacturer's data sheets of the memory ICs specify these timing requirements. If any of these parameters are not followed then the manufacturer does not guarantee the intended operation. It is expected that design engineer will come out with the necessary access circuitry that meet these timing requirements for proper operation.

Based upon the discussions we had in earlier articles of this chapter we can list out some of the operational requirements.

- A valid memory address must be applied to the address input.

- Before initiating any operation the chip must be enabled through  $\overline{\text{CS}}$  signal line.
- The appropriate control signal then is applied for a certain time.

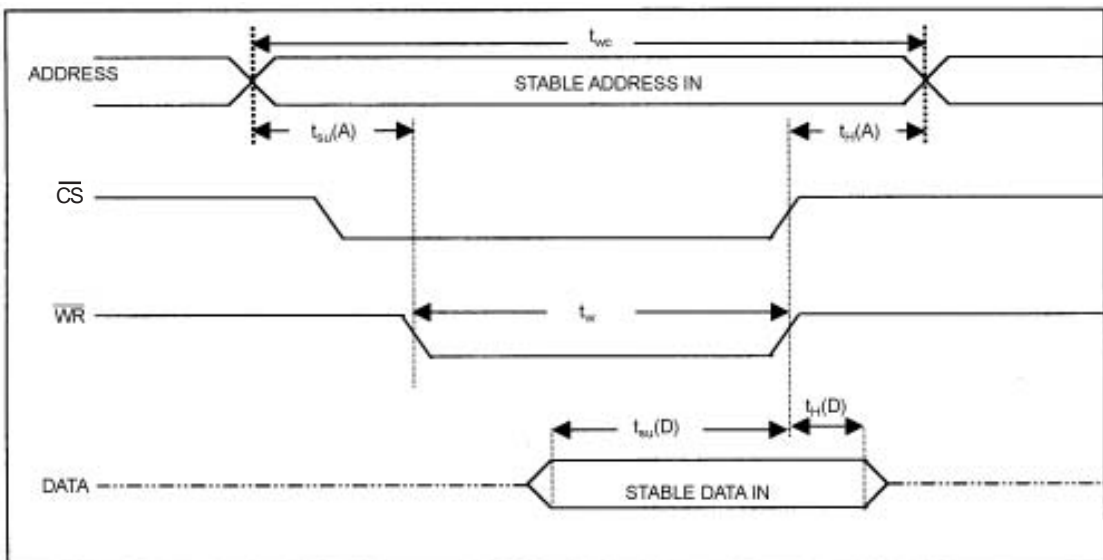
More or less all type of memory operations is followed in above-mentioned sequence. For the sake of easiness we first consider the memory write operation and then read operation. The readers are advised to devote some time to understand these waveforms.

#### 11.4.6.1 Memory Write Operation

*Memory Write operation* means setting the status of the selected memory cell either to 1 or to 0 according to the data supplied through the data bus. The timing waveforms for a typical memory write operation is shown in Fig. 11.11A. A close inspection reveals that to write a data to a memory location one must—

- Apply a valid address to the address bus to select a location.
- Initiate chip select signal by making  $\overline{\text{CS}} = 0$ .
- Apply the write control signal by making  $\overline{\text{WR}} = 0$  for a certain time.
- Apply the data to be written at the selected memory location to the data bus for a certain time.

In response the information present at data bus replaces the word at addressed memory location that might have been stored in it. The waveform also shows the different times that are typical and are explained as below:



**Fig. 11.11** (a) Simplified Memory Write Cycle Waveforms

*Write Cycle Time ( $t_{wc}$ ):* This is defined as the minimum amount of time for which a valid address must be present at the address bus for writing a memory word. As earlier defined this is the minimum time that must lapse between two successive write cycles.

*Address Setup Time  $t_{su}(A)$ :* It is defined as the minimum time for which the address must be stable at the address bus before  $\overline{\text{WR}}$  goes low.

**Address Hold Time  $t_H(A)$ :** This is defined as the minimum time for which the valid address must be stable at the address bus after  $\overline{WR}$  rises.

**Write Pulse Width ( $t_W$ ):** This is defined as the minimum amount of time for which  $\overline{WR}$  must be low in order to store the data in the memory. It must be noted that  $t_W$  is measured from the later of write pulse or chip select going low to the write pulse going high. A write occurs during the overlap of low chip select and low write pulse i.e. when both are low. Also the write pulse should be high during all address transitions.

**Data Setup Time  $t_{SU}(D)$ :** This is defined as the minimum time for which the data to be written at addressed location must be held stable at the data bus before  $\overline{WR}$  rises.

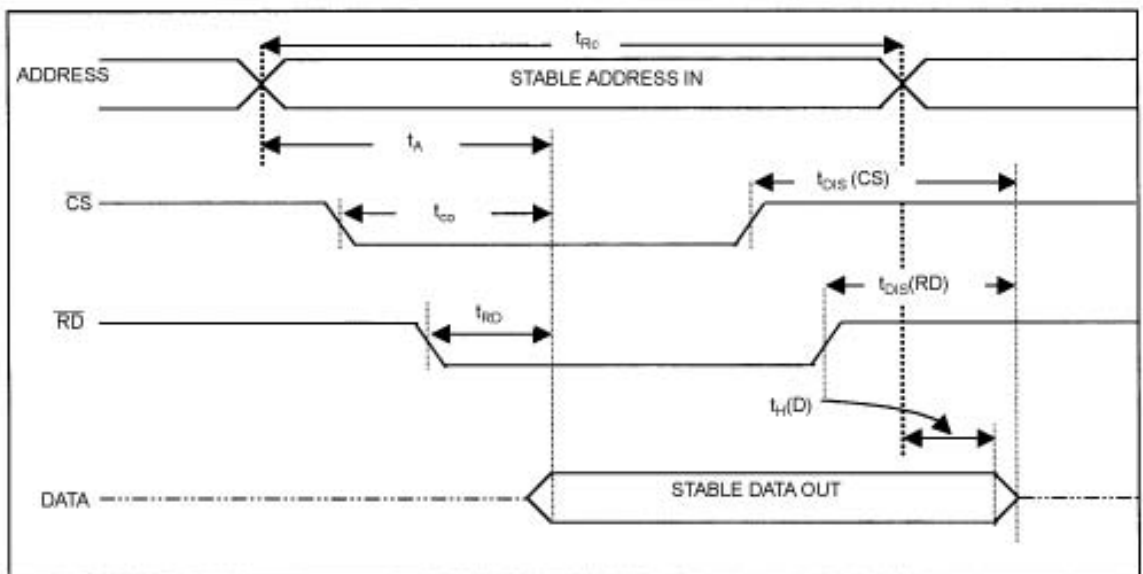
**Data Hold Time  $t_H(D)$ :** This is defined as the minimum time for which the data to be written at addressed location must be held stable at the data bus after  $\overline{WR}$  rises.

#### 11.4.6.2 Memory Read Operation

A *Read operation*, also called *sensing*, means detecting the status (i.e. information) of selected memory cell that whether it is 1 or 0. The timing waveforms for a typical memory read operation is shown in Fig. 11.11B. A close inspection reveals that to read a data from a memory location one must

- Apply a valid address to the address bus to select a memory location.
- Initiate chip select signal by making  $\overline{CS} = 0$ .
- Apply the read control signal by making  $\overline{RD} = 0$  for a certain time.

In response the information from addressed memory location is transferred to the data bus. The waveform also shows the different times that are typical and are explained as below.



**Fig. 11.11 (b)** Simplified Memory Read Cycle Waveforms

**Read Cycle Time ( $t_{RC}$ ):** This is defined as the minimum amount of time for which a valid address must be present at the address bus for reading a memory word. As earlier defined this is the minimum time that must lapse between two successive read cycles.

**Read Access Time ( $t_A$ ):** Read Access Time or *Access time* is defined as the maximum delay between the appearance of stable address and appearance of stable data. This definition is based upon the assumption that chip select goes low after the application of address. Access time is at the most equal to the read cycle time i.e.  $t_A \leq t_{RC}$ .

**Chip Select to Output Valid time ( $t_{CO}$ ):** This is the maximum time delay between the beginning of chip select signal and appearance of stable data at the data bus. It is some times also defined as the access time from chip select  $t_A(\text{CS})$ .

**Read to Output Delay ( $t_{RD}$ ):** This is the maximum time delay between the beginning of read signal and appearance of stable data at the data bus. It is some times also defined as the access time from read  $t_A(\text{RD})$  or read to output valid time.

**Chip Select to Output Disable  $t_{DIS}(\text{CS})$ :** This is the maximum time delay after the end of chip select signal for data bus to go to high impedance state i.e. to get disabled. It is also defined as  $t_{OTD}$ , output 3-state from deselection.

**Read to Output Disable  $t_{DIS}(\text{RD})$ :** This is the maximum time delay after the end of read signal for data bus to go to high impedance state i.e. to get disabled.

$$\text{Generally } t_{DIS}(\text{CS}) = t_{DIS}(\text{RD}) = t_{OTD}$$

**Data Hold Time  $t_H(D)$ :** This is defined as the minimum time for which the stable data is available at the data bus after the end of the valid address. It is also defined as  $t_V(A)$ , output data valid after change in address or output hold from change in address  $t_{OHA}$ .

**Example.** For a memory the read & write cycle times are 100 nsec. Find out the bandwidth for this memory. What will be the access rate if access time is 80 nsec?

**Solution.** By section 11.2 we know that *Data Transfer Rate* or *Bandwidth* of a memory is given as

$$b_C = 1/t_C \text{ words/sec}$$

so bandwidth for given memory is  $b_C = 1/(100 \times 10^{-9})$

$$b_C = 10 \times 10^{11} \text{ words/sec.}$$

Again by section 6.2 we know that *Access Rate* of a memory is given as

$$b_A = 1/t_A \text{ words/sec}$$

so access rate of given memory is  $b_A = 1/(80 \times 10^{-9})$

$$b_A = 12.5 \times 10^6 \text{ words/sec.}$$

or

$$b_A = 125 \times 10^5 \text{ words/sec.}$$

#### 11.4.7 Read Only Memory

A read only memory (ROM) is a semiconductor storage device that allows only read operation to be performed on it. It is used to store the information permanently. It is an essential part computer system that stores *boot up* information required at the time of switch on by the operating systems. It has become an important part of many digital systems because of its high speed, large storage at low cost, and non-volatility. It offers great flexibility in system design and has got many applications including implementation of various complex combinational logic, lookup tables, character displays, embedded systems etc.

ROMs are mainly classified by the method that is used to store the information. The mechanism that stores the data into ROM is called as *programming mechanism*.

Basic organization of a ROM remains same as was discussed from 11.4.1 through 11.4.4 with the only difference that ROMs do not contain write signal. The actual consideration here would be the type of cells that a ROM device must use. Before going for discussion on the typical cells for different kind ROM cells we first consider some simpler ROM organization that can be worked out in labs to demonstrate the idea. The logic diagram of ROM unit is shown in Fig. 11.12.

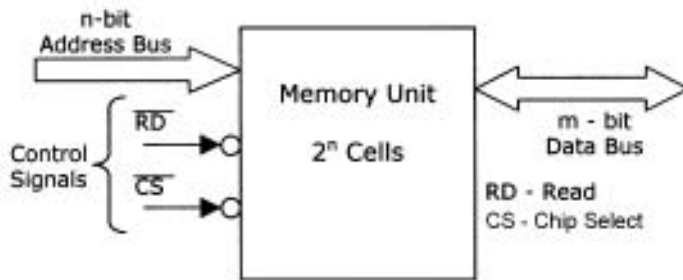


Fig. 11.12 Logic Symbol of A ROM Memory Unit

11.4.7.1 Some Simple ROM Organizations

**Diode Matrix ROM:** They are the simplest types of ROM. They were discussed previously in chapter 5.

**Multiplexer ROM:** It is yet another simple type of ROM. In this case the multiplexer inputs are fixed to logic 1 or logic 0 as desired and the select inputs can be used as address lines to select the particular memory word. Figure 11.13 shows such a 4 × 1 ROM with the truth table of stored information.

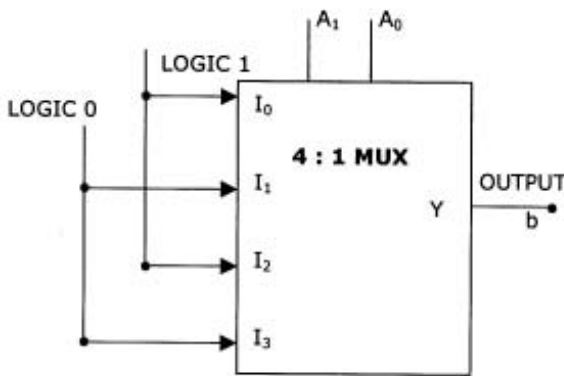


Fig. 11.13 (a) A 4 to 1 line multiplexer connected as 4 × 1 ROM

MEMORY LOCATION or ADDRESS		STORED WORD
A <sub>1</sub>	A <sub>0</sub>	b
0	0	1
0	1	0
1	0	1
1	1	0

Fig. 11.13 (b) Truth Table for ROM in Fig. 11.13(a)

It is interesting to note that forming a multiplexer tree with such fixed data connections can make a ROM of larger word length thus increasing the storage capacity. Such a ROM is shown in Fig. 11.14. The figure shows a 4 X 2 ROM with a storage capacity of 8-bits as opposite to the ROM of Fig. 11.13 which is a 4-bit ROM of word length of 1-bit. Note that the

address lines are shorted together as usual to address the bits of same location from two MUX. Readers are encouraged to find out the words stored at different locations in this ROM.

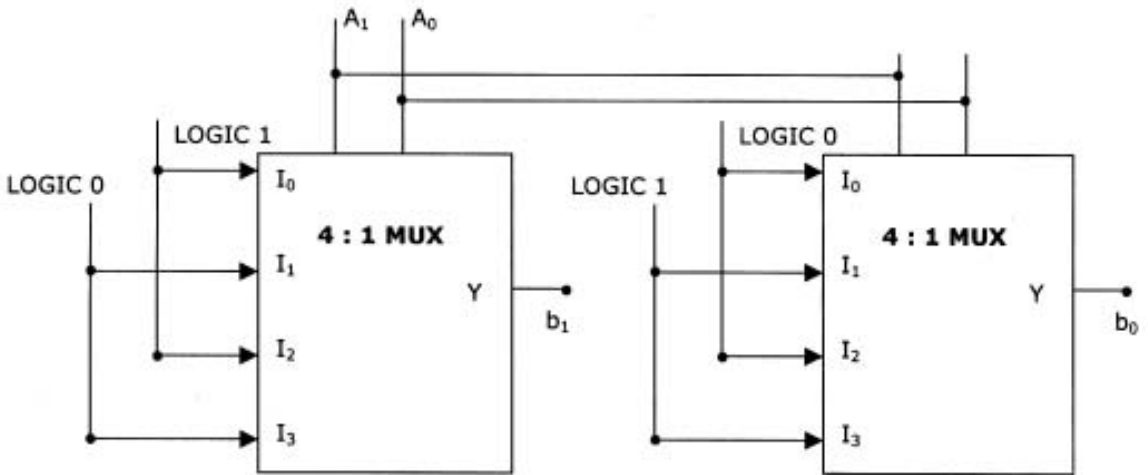


Fig. 11.14 Two 4 to 1 line multiplexer connected as 4 × 2 ROM.

**OR-Gate ROM:** It is yet another simpler ROM but more versatile than the multiplexer ROM. In this case a decoder and n-input OR gates are used. The selection of decoder depends upon the number of words needed to store. The word length determines the number of OR-gates. The select inputs are used as address lines to select the particular memory word. Figure 11.15 shows such a 4 × 4 ROM.

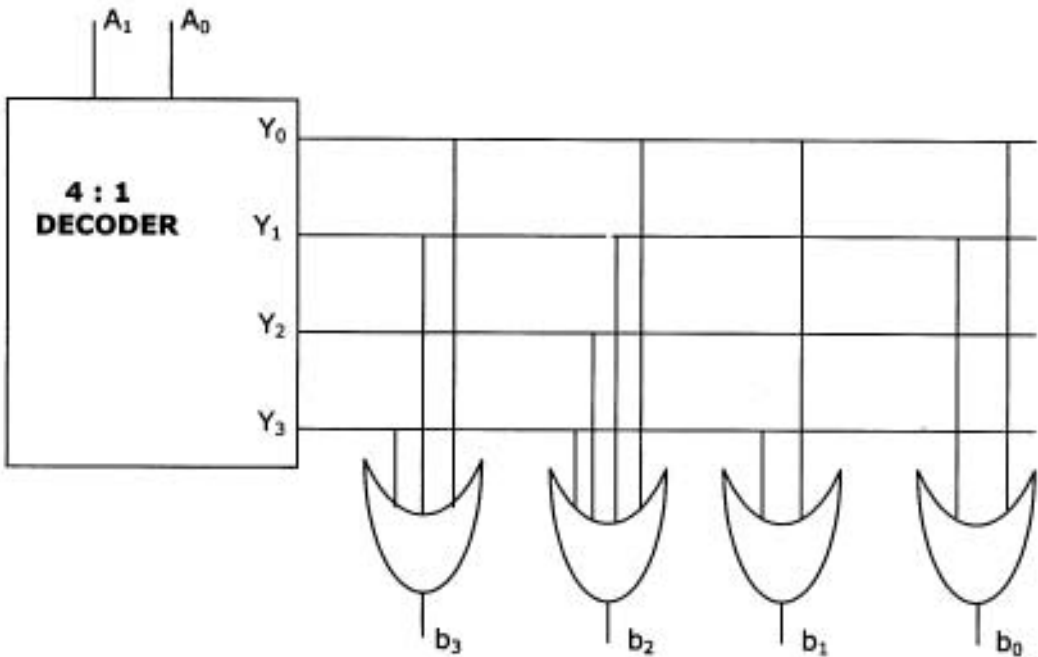


Fig. 11.15 16-bit, 4 × 4 ROM using OR-Gates and 4 to 1 line decoder



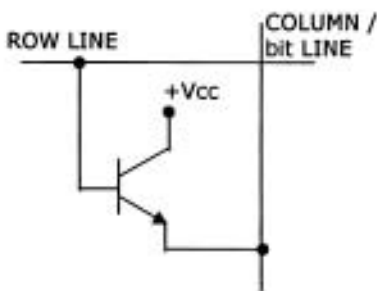
Assuming that upon activation each of the decoder line becomes high we get that location 10 stores the word 0100. The readers are advised to find out the words stored at other locations.

An interesting fact can be noted from all the above organizations that each of them corresponds to AND-OR organization. Diode matrix is an AND-OR configuration and both multiplexers & decoders are internally AND-OR networks. This is nothing but *Sum of Product* (SOP) realization. This implies that the ROMs can also be used for SOP realization!!

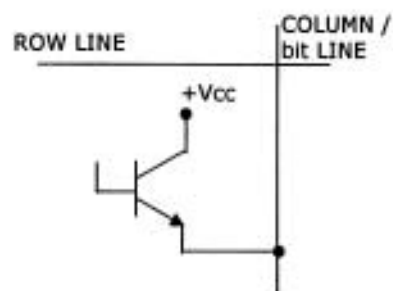
#### 11.4.7.2 Mask Programmed ROMs

In general the word ROM is used to refer to Mask Programmed ROMs. It is the type of memory, which is permanently programmed during the manufacturing process according to the information supplied by the users. Upon receiving the information to be stored into the ROM the manufacturer prepares a photographic template of the circuit, called *Photo Mask*. This photo mask is actually used for production of ROM ICs storing the needed information. Since photo mask is used for programming such a ROM is called Mask Programmed ROM. Due to the various fabrication processes involved in such production like photomasking, etching, diffusion, Aluminium layering etc the overall cost is high. This makes them unsuitable when only a small number of ROMs are needed. For this reason mask programmed ROMs are always used for high volume production.

Both the BJTs and MOS devices can be used as basic memory cell for this kind of ROM. Fig. 11.16 shows the Bi-polar transistors connected to store logic 1 and logic 0. As shown the base connected transistor can be used to store logic 1. If row line is connected to the base of transistor then whenever ROW line goes high transistor turns ON and  $+V_{CC}$  is connected to column line which is interpreted as Logic 1. But if base is unconnected the high on row line does not make it working and status of column line remains low which is interpreted as Logic 0.



**Fig. 11.16** (a) A Bi-POLAR ROM '1'  
Cell storing



**Fig. 11.16** (b) A Bi-POLAR ROM  
Cell storing '0'

In the similar way Gate connected MOS transistor can be used to store Logic 1 and floating Gate MOS transistor can be used to store Logic 0. This is shown in Fig. 11.17.

When organizing the ROM array the output data lines of individual cells in the same column are connected together as shown in Fig. 11.18. This is done by assuming that such connections will form *Wired-OR<sup>ing</sup>* and such assumption is valid for most of the semiconductor technology.



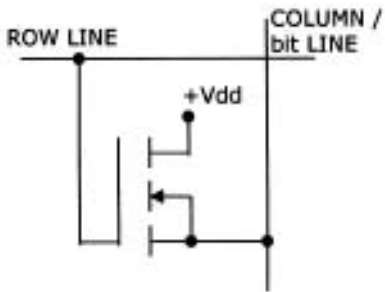


Fig. 11.17 (a) A MOS ROM Cell storing '1'

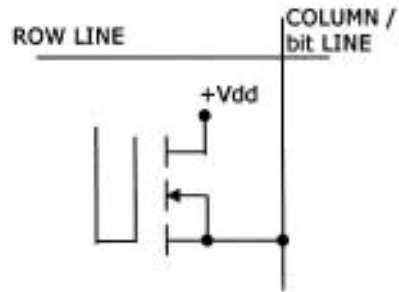


Fig. 11.17 (b) A MOS ROM Cell storing '0'

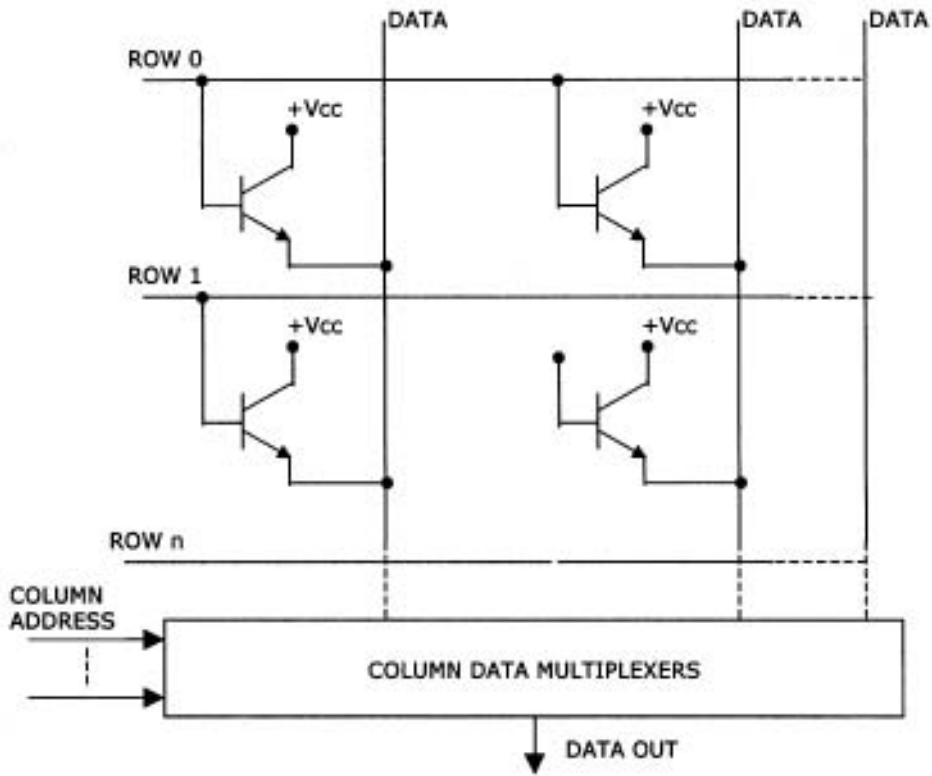


Fig. 11.18 An Array of Bipolar ROM Cells

Finally column multiplexers may be used to select the data from a particular column only. This organization identical to the one, shown in Fig. 11.9.

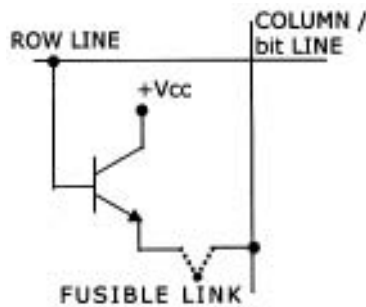
#### 11.4.8 Programmable Read Only Memory (PROM)

The mask programmed ROMs have disadvantage that they are not flexible and with low volume production they become highly expensive. A Programmable ROM is a type of ROM which efforts to minimize these two disadvantages. These ROMs can be programmed by the users in the field, thus they are also called field programmable ROMs. These ROMs provide

some flexibility to user but their disadvantage is that they can be programmed only once. PROMs are available with both the Bipolar and MOS technology. Conceptually the programming technique in both technologies is same, but the real difference appears in the actual programming mechanism. In case of Bipolar technology an instantaneous current pulse is used for programming where as the current pulse is applied for certain period (typically for a few msecs) of time in case of MOS technology.

#### 11.4.8.1 Bi-Polar PROMs

Fig. 11.19 shows a Bipolar PROM cell that allows such kind of programming in PROMs. As shown the emitter is connected with a fusible link that can be blown off or burned by applying a high current to it. When the fuse remains intact it stores logic 1 and when it is burnt it stores logic 0. Initially all the cells have fuse intact i.e. they all stores logic 1 and at the of time of programming users can selectively burn them out to store logic 0.

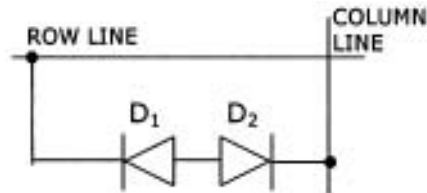


**Fig. 11.19** A Bipolar PROM cell with fusible link

This shows why these memories can not be reprogrammed. Since programming the PROM involves burning the fuse it is also referred as *burning*.

In Bipolar PROM cells the fusible links are always placed between emitter of the transistor and the column/data line, as shown in Fig. 11.19. Such types of cells are called fuse cells. Basically there are types of fuse technology used in Bipolar PROMs as briefed below.

- *Metal Links* are formed by depositing a very thin layer of nichrome as material. This fuse can be burned during programming to store logic 0 by flowing large currents through them.
- *Silicon links* are formed by depositing a thick layer of polycrystalline silicon. This fuse can be burned to store logic 0 by flowing current pulse train of successive wider pulses. This programming process causes a temperature upto 1400°C at the fuse location, and oxidizing the silicon. This oxidization forms insulation around the opened link.
- *Avalanche Induced Migration (AIM) Process* this programming technique uses two PN junction diodes connected back to back as shown in Fig. 11.20. During the programming the diode  $D_1$  is in reversed bias. There flows a large current in reverse direction due to avalanche of electrons. The heavy reverse current along with heat generated at the junction causes some aluminium ions to migrate. Due to this the base to emitter junction is shorted. Although this process requires higher currents and voltages than the earlier two, but this is faster.



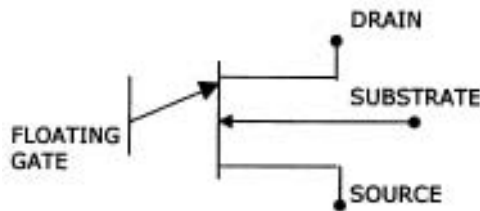
**Fig. 11.20** A Shorted Junction Cell

#### 11.4.8.2 MOS PROMs

The fuse mechanism shown for Bipolar PROMs does not work for MOS technology where high current and resistance levels required for programming is incompatible with MOS impedance levels. Commonly used MOS fabrication techniques are

- Floating Gate Avalanche Injection MOS (FAMOS)
- MAOS

In FAMOS PROMs silicon gate MOSFET with no electrical connection to the gate is used as storage device. In this case gate is floating in an insulating layer of silicon dioxide. The operation of such device is based on charge transport on application of high voltage to the floating gate. This charge gets trapped in the gate when voltage is removed, as there is no electrical connection for gate. Logic symbol of this device is shown in Fig. 11.21.



**Fig. 11.21** Logic Symbol of FAMOS Device

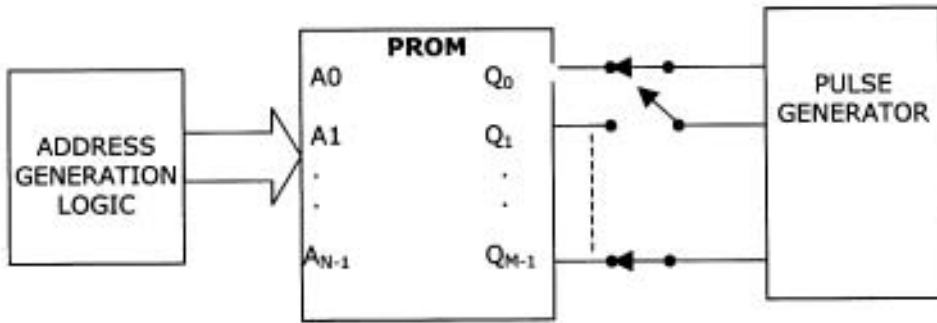
It is interesting to note that the charged trapped in floating gate of FAMOS device can be removed if the cell is exposed to high energy UV radiation for some times.

A MOS memory cell that uses *Alumina* ( $\text{Al}_2\text{O}_3$ ) and silicon nitride as gate dielectric for charge storage is called MAOS memory element. Applying negative and positive polarity voltages can program the MAOS elements. The MAOS elements can be erased by applying opposite polarity voltage at gate and thus provides reprogramming.

Note that the MOS techniques provide reprogramming feature as opposite to Bipolar technology.

#### 11.4.8.3 PROM Programming

As explained above the programming of PROM requires varying amount of currents to be supplied to store the information. This is done through a special device called PROM Programmer that mainly contains programmable pulse generator to meet requirements of different devices. A simplified programming setup is shown in Fig. 11.22.



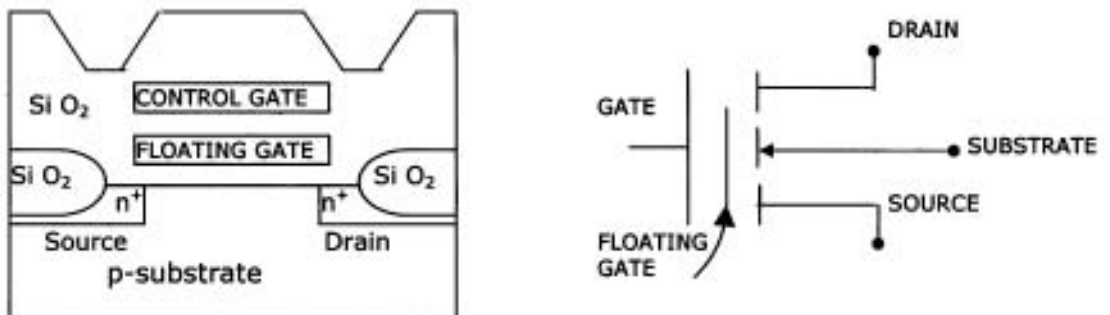
**Fig. 11.22** Simplified PROM Programming Setup

As earlier discussed initially PROMs will contain all 1's or all 0's. Let initially all 1's are stored then to store information first of all find out the bit positions where 0s have to be stored. Connect the corresponding output bit line i.e.  $Q_i$  to the pulse generator. The address generation unit selects the location of the word to be programmed. After the selection of word pulse generator supply the appropriate amplitude pulse to burn out the fuse, thus storing 0 at connected bit positions. The process is repeated for all memory words and the PROM can be programmed completely. Although not shown in figure all the power supply and other connections are assumed to be present for PROM before programming.

#### 11.4.9 Erasable Programmable ROM (EPROM)

One disadvantage of PROM is that once programmed it can not be reprogrammed. Any mistake during the programming can not be corrected and the whole chip may become useless. An EPROM overcomes this problem by providing an erase operation. EPROM is a PROM device that can be reprogrammed as many times as needed.

Note that both ROMs & PROMs can be either Bipolar or MOS but EPROMs can only be MOS. An indication of this fact was given in previous section when discussing the FAMOS & MAOS devices. In its simplest form an EPROM can be viewed as a FAMOS PROM with an additional gate, called control gate. Fig. 11.23 shows the basic structure of such a device and its logic symbol. The programming mechanism is same as previous and to erase UV radiation is used.



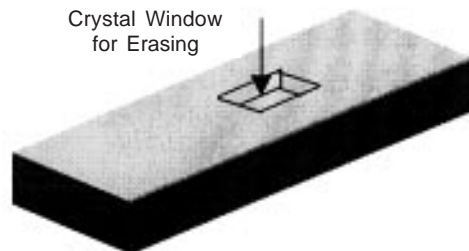
**Fig. 11.23** A Structure of EPROM CELL

Shown in the figure is an n-channel enhancement mode MOSFET with insulated gate architecture. The additional gate, called control gate, is formed within the  $\text{SiO}_2$  layer. The control gate is connected to the row address line. As usual presence or absence of stored charge represents the data.

#### 11.4.9.1 EPROM Programming

To program the memory cell the floating gate must be charged i.e. trap the charge. To accomplish this a high positive voltage is applied between source & drain is applied. A slightly higher +ve voltage is applied at the control gate. Thus high electric field strength is developed which highly energizes the electrons. These energetic electrons thus reach to floating gate and charge is accumulated. As the more electrons (i.e. negative charges) are accumulated the field strength reduces and further electron accumulation is inhibited. Since an insulating layer ( $\text{SiO}_2$ ) surrounds this gate, no discharge path is available for accumulated electrons. Thus charge carriers remain trapped at gate. Thus the programming requires charging the floating gates of individual memory cells. Programming of the EPROMs requires higher current levels for different times. For this purpose a special device, called EPROM Programmer, is used. All the EPROM chips must be physically removed from the memory boards and be connected to the EPROM programmer for reprogramming or programming. This is considered as the disadvantages of the EPROM.

To reprogram all cells of EPROMs must be erased. Illuminating EPROM to strong UV radiations carries out the erase operation. For this reason they are some times called as *UV erasable PROMs* or simply *UV EPROMs*. It typically requires a radiation of wavelength of about 254 nm to be present for about 20 minute. The photons of the incident radiation provide sufficient energy to the trapped electrons to escape to substrate through the  $\text{SiO}_2$  layer.



**Fig. 11.24** CYRSTAL WINDOW to ERASE DATA from EPROM

The EPROMs are provided with a transparent crystal window on the top of the IC, as shown in Fig. 11.24 to allow UV rays to enter to the chip to erase the data. Upon illumination contents of all the locations are erased, which is considered as the biggest disadvantage of EPROMs. At the time of working the EPROM window is covered with the opaque sticker to prevent it from unwanted exposure from sunlight or from other sources.

The EPROMs can be erased by direct sunlight in about one week or by room level fluorescent lighting in about one to three years.

#### 11.4.9.2 The 27XXX EPROM Series

A popular *EPROM family* is 27XXX series and commercially available from many manufacturers. In the IC number XXX represents the storage capacity in Kbits. For example in IC 2732A XXX = 32 means it can stores a total of 32 Kbits. However all ICs of this series

are organized to have a word length of a byte i.e. 8-bits. Hence 2732A is a  $XXX/8 = 32 \text{ Kbits}/8 = 4\text{KB}$  memory device. Since length of each memory word is byte this IC contain 4 K ( $4 \times 1024 = 4096$ ) memory location. Thus by equation 4 get that IC 2732A must have 12 address lines (as  $2^{12} = 4096$ ) to select any of the 4096 words. In the similar way we can work out for all the members of this family to find out the storage capacity and the width of address bus. Data bus width will always be same as 8-bit. Pin diagram of IC 2732A is shown in Fig. 11.24. The signal is  $\overline{OE}/V_{pp}$  a dual purpose signal. Under the normal operation a logic 0 on this pin allows the data from selected location to appear at data output lines. The 2732A is in programming mode when it is connected to 21V. When 2732 is erased all the memory cells are returned to logic 1 status. The programming is done by applying the required input data on the  $O_0 - O_7$  pins.

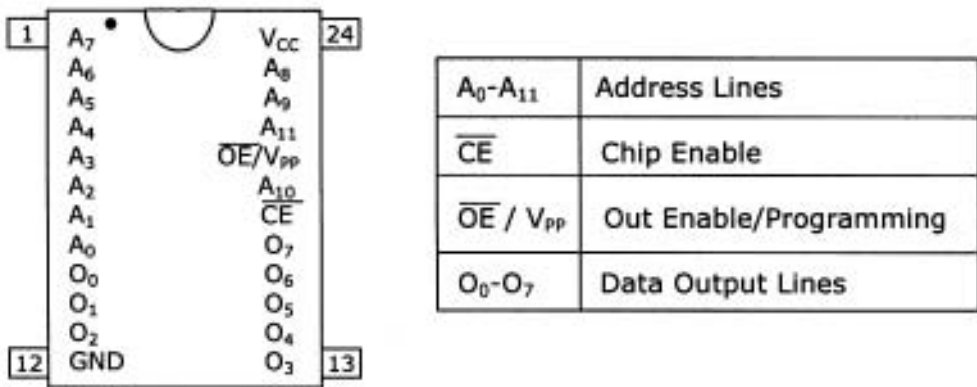


Fig. 11.24 INTEL 2732A EPROM IC, 4K × 8 IC

Fig. 11.25 list out a short summary of some models in 27XXX series. Also note that these chips are fully TTL compatible and has an access time less than 450 nsec that permits an access rate (most of the time Data Transfer Rate) of approximately  $2 \times 10^{11}$  memory words/sec or  $6 \times 10^6$  bits/sec.

EEPROM 27XXX	MEMORY ORGANIZATION	STORAGE	CAPACITY
		KBYTES	BITS
2708	1024 × 8	1 KB	8192
2716	2048 × 8	2 KB	32768
2732	4096 × 8	4 KB	327118
2764	8192 × 8	8 KB	1155311
27128	16384 × 8	16 KB	131072
27256	32768 × 8	32 KB	262144
27512	65536 × 8	64 KB	524288

Fig. 11.25 27XXX SERIES EPROMs

#### 11.4.10 Electrically Erasable Programmable ROM (EEPROM)

One disadvantage of EPROM is that erase operation erase all the memory cells completely and reprogramming involves entering the complete data. There is no option of selective erase. Another disadvantage is that for reprogramming the chip must be removed physically from the

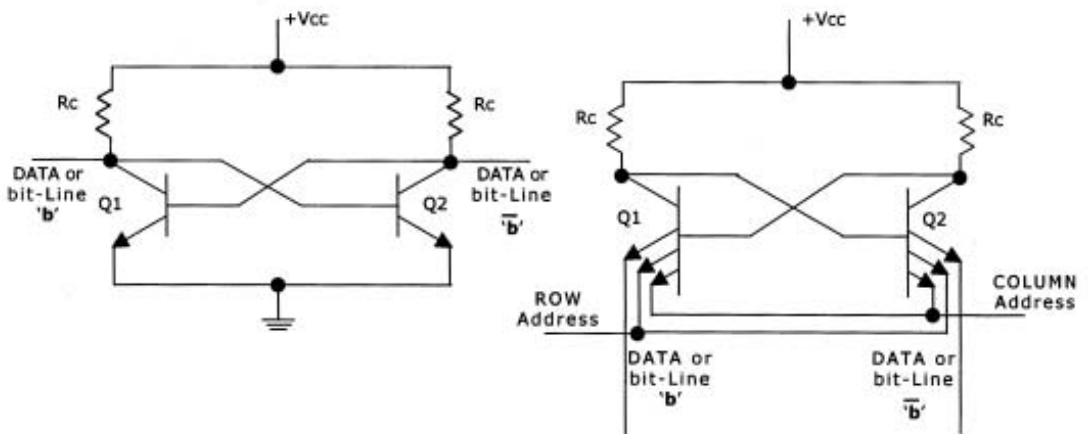




Since SRAM contains array of flip-flops, a large number of flip-flops are needed to provide higher capacity memory. For this reason simpler flip-flop circuits using BJTs and MOS transistors are used for SRAM. This helps to save chip area and provides memory IC at relatively reduced cost, increased speed and reduces the powers dissipation as well. The static RAMs have very small access times typically less than 10 nsec. SRAM with battery backup is commonly used provide non-volatile RAM (NVRAM).

#### 11.4.12.1 The Bi-Polar SRAM Cell

For the reasons stated above we use simpler flip-flop circuits to implement flip-flops in memory. The circuit shown in Fig. 11.27 (a) forms basic bipolar latch that can be used to store 1-bit of binary information. Note that it contains two cross-connected collector coupled transistors  $Q_1$  and  $Q_2$ . At any time one of the transistor remains on and other remains off i.e. status of their collector are always complement to each other thus satisfying the basic definition of flip-flop. That's why the data lines are labeled as  $b$  and  $\bar{b}$ .



**Fig. 11.27** (a) Basic bipolar Latch (or Storage Cell) **Fig. 11.27** (b) Basic bipolar SRAM Cell

Fig. 11.27(b) shows the same latch but along with the necessary control and the data signals. Use of multi-emitter transistors accommodates the necessary signals of BJT SRAM cell. To read or write to this cell both the row and col lines must be high.

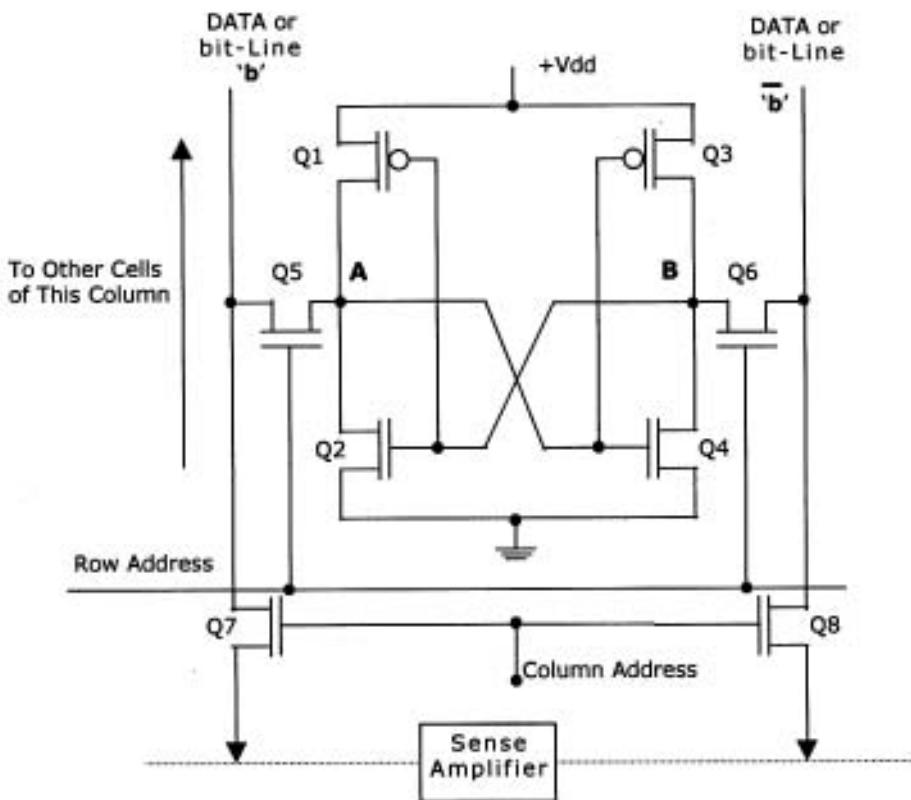
To **write** the two bit lines are placed with required status and row and col lines are placed high. This causes the two lower emitters of both the transistors to stop current conduction. If a logic 1 was to be store then bit line  $b$  placed as  $b = 1$ , this makes upper emitter of  $Q_1$  to cease to conduct but the upper emitter of transistor  $Q_2$  will remain conducting as its bit line is placed as  $\bar{b} = 0$ . This makes  $Q_1$  to go to off state and  $Q_2$  to go to on state, thus giving status of  $Q_1$  as logic 1 and  $Q_2$  as logic 0. The similar process can be used to store logic 0 to this cell. Every a time the state of transistor  $Q_1$  can be read to sense the status of this cell.

To **read** the data from this cell row and col lines are placed high and current flowing through the upper emitter can be sensed by read amplifier to declare it as 0 or 1.



11.4.12.2 *The MOS SRAM Cell*

A CMOS realization of SRAM cell is shown in Fig. 11.28. The cell comprises two cross-connected inverters,  $Q_1$  and  $Q_2$  as inverter 1 and  $Q_3$  and  $Q_4$  as inverter 2, thus forming a latch capable of storing 1-bit of information. When the cell stores logic 1 the voltage at point A is maintained high and that at B is maintained low by having  $Q_1$  and  $Q_4$  ON and  $Q_2$  and  $Q_3$  OFF. Thus if  $Q_5$  and  $Q_6$  are turned ON by activating the row address line, the bit lines will have their status respective to voltage at A and B. Finally upon activation of column address lines  $Q_7$  and  $Q_8$  turn ON and pass this information to read/sense amplifier. Note that the bit line is connected to all the cells of a column by forming wire-OR connection. The write operation for this cell is same as BJT cell in that bit lines are first fixed to required status and then the cell is forced to enter in that status.

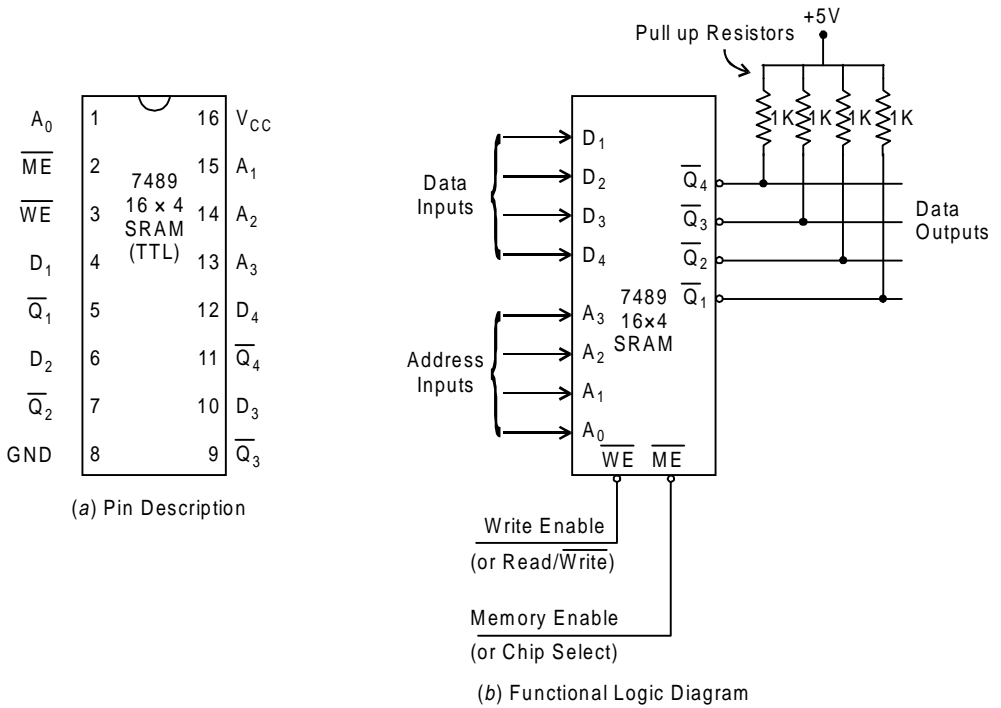


**Fig. 11.28** CMOS SRAM Storage Cell

The power supply  $V_{dd}$  is 5 volts in standard CMOS SRAMs and 3.3 volts in low-voltage versions. The biggest advantage of using CMOS cell is very low power consumption because the current can flow in the cell only when they are accessed. Otherwise one of the transistor from each inverter remains OFF ensuring that there is no active path between the supply voltage  $V_{dd}$  and ground. Although CMOS is slower than the BJT but latest advances have reduced the difference in the Access time of the two types of SRAM. Also since CMOS devices have very large packing density, almost all the SRAM devices with capacity more than 1024 bits are MOS devices.

11.4.12.3 SRAM ICs

- IC 7489-114 bit TTL SRAM:** Fig. 11.29 shows the pin diagram and logic diagram along with truth table & signal description. This is a 114-bit SRAM organized as 111 X 4 memory. The data sheet for this memory device specify a maximum read data delay or access time of 50 nsec. Note that the device contains separate data busses for input and output. The output of this IC always the complement of bits stored in the memory. Moreover this IC package provides open collector output thus it requires external pull up resistors to be used as shown in functional diagram. Note the third entry of the truth table with the mode of operation as inhibit storage. In this mode the device simply works as inverter in that the data present at the input complemented and presented at the output. In this mode neither the data can be stored nor the data read weather address is applied or not.



Signal	Description
$\overline{ME}$	Selects or Enables the chip when $ME = 0$
$\overline{WE}$	Selects write operation when $WE = 0$ Selects read operation when $WE = 1$
$A_0$ to $A_3$	Address lines to select any of the 16-words
$D_1$ to $D_4$	4-bit Data Input Bus
$\overline{Q_1}$ to $\overline{Q_4}$	4-bit Data Output Bus-Data output is complement of bits stored.

(c) Signal Description

Operating Mode	Inputs		Output Condition
	$\overline{ME}$	$\overline{WE}$	
Write	0	0	Complement of Data Inputs
Read	0	1	Complement of Selected word
Inhibit Storage	1	0	Complement of Data Input. No read or write is done
Do Nothing	1	1	All Outputs High

(d) Truth Table

**Fig. 11.29** 64 bit (16x4) TTL SRAMIC

- TMS 4016-2KB MOS SRAM:** Fig. 11.30 shows the pin diagram of this MOS SRAM device. This device is representative of all other  $2K \times 8$  SRAM MOS ICs such as Intel 2016, and most popular in this range the 6116 IC. The figure shows the signal description also, with the help of which truth table for this IC can be worked out.

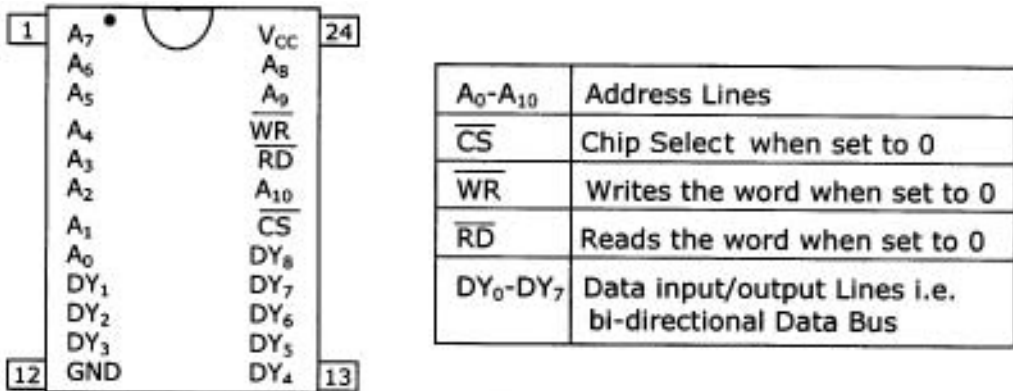


Fig. 11.30 TMS 4016 MOS SRAM IC, A  $2K \times 8$  IC

An interesting point to note that many of the EPROM ICs and SRAM ICs (such as 27XXX & 62XXX family) are pin compatible. For example compare the Fig. 11.24 to the figure 11.30. One can find that few signal names are changed and WR signal is replaced by  $V_{pp}$ .

- Popular SRAM ICs are from 62XXX MOS family like IC-6264 a  $8K \times 8$  memory, IC-62256 a  $32K \times 8$  memory etc. Readers are encouraged to search & list out several other SRAM ICs from various references.

### 11.4.13 Dynamic Random Access Memory (DRAM)

SRAMs are faster but they come at high cost, as their cell requires several transistors. Less expensive RAMs can be obtained if simpler cells are used. A MOS storage cell based on dynamic charge storage is much simpler and can be used to replace the SRAM cells. Such a storage cell can not retain the charge (i.e. information) indefinitely and must be recharged again, hence these cells are called as dynamic storage cells. RAMs using these cells are called Dynamic RAMs or simply DRAMs.

#### 11.4.13.1 Basic DRAM Cell

A basic DRAM cell is shown in Fig. 11.31, in which information is stored in the form of charge stored on a capacitor. The cell consists of a MOS transistor and a storage capacitor C. Information can be stored as presence or absence of charge on capacitor C. But the charge stored on the capacitor tend to decay with time. It is because of two facts- first even though the transistor is OFF there flows a small leakage current, and secondly the capacitor itself has a leakage resistance. The result is charge (i.e. information) can only be retained for a few milliseconds. Thus to retain the information for a longer period of time the capacitor must be recharged periodically before charge goes below a threshold value. The process of restoring charge is called as refreshing. Even though DRAM requires additional circuitry to deal periodic refresh and destructive read the advantage of greater storage capacity on a single chip make it worth, as it still offers a low cost per bit solution.

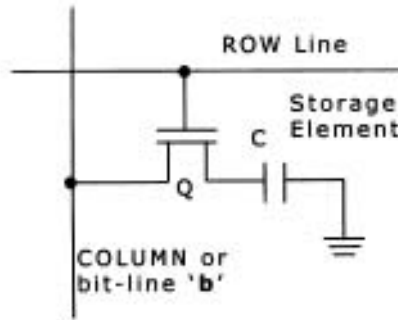


Fig. 11.31 Basic DRAM Storage Cell

- Read and Write Operation** to store the information into the cell the row address line is activated to turn ON the transistor, and a voltage (either high or low corresponding to 1 or 0) is placed on the bit line. This causes a known amount of charge transfer to the storage capacitor if status of bit line is 1 and no charge transfer if bit line is 0. During read, the transistor is turned ON and charge on the capacitor is transferred to the bit line. A sense amplifier connected to bit line detects whether the stored charge is below or above a threshold value or not and accordingly interpret it 1 or 0. Since the read process discharges the capacitor, information is lost, it called as *Destructive Read Out (DRO)* process. For this reason the information that was read out is amplified and written back to the storage cell. A DRAM cell therefore is refreshed every time it is read. Alternately, the activation of row line refreshes the cell every time. This forms the basis of periodic refreshing of DRAM, as we will see later, to keep it from losing the information. Also note that transistor itself acts as transmission gate so no additional transistors are needed as was required in MOS SRAM cell, see Fig. 11.28.

11.4.13.2 One MOS Transistor DRAM Cell

The circuit used for DRAM MOS cells may use one or more transistors. The simplest is one transistor cell as shown in Fig. 11.32(a). It consists of an NMOS transistor and a MOS capacitor,  $C_{MOS}$ , fabricated on a chip. The equivalent circuit is shown in Fig. 11.32(b), in which  $C_j$  represents the junction capacitance of transistor. If MOS capacitor plate lead & substrate leads are connected to ground, it then forms a parallel connection of  $C_j$  and  $C_{MOS}$ . The storage capacitor is  $C = C_j + C_{MOS}$  which can be used to store charge in dynamic cells.

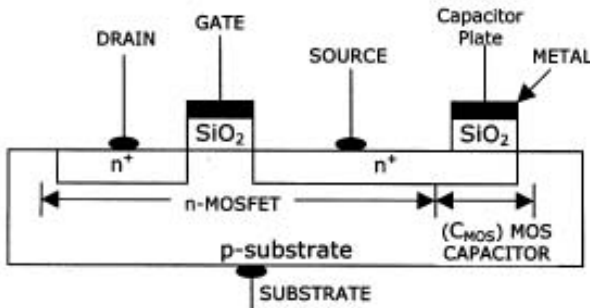


Fig. 11.32(a) Fabrication of One Transistor Dynamic Memory Cell

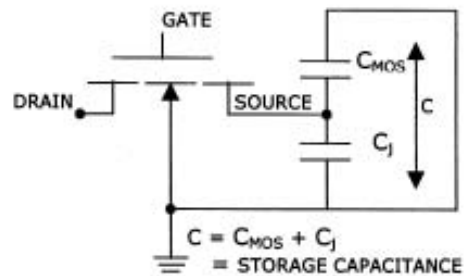


Fig. 11.32(b) Equivalent Circuit Diagram

11.4.13.3 DRAM Organization

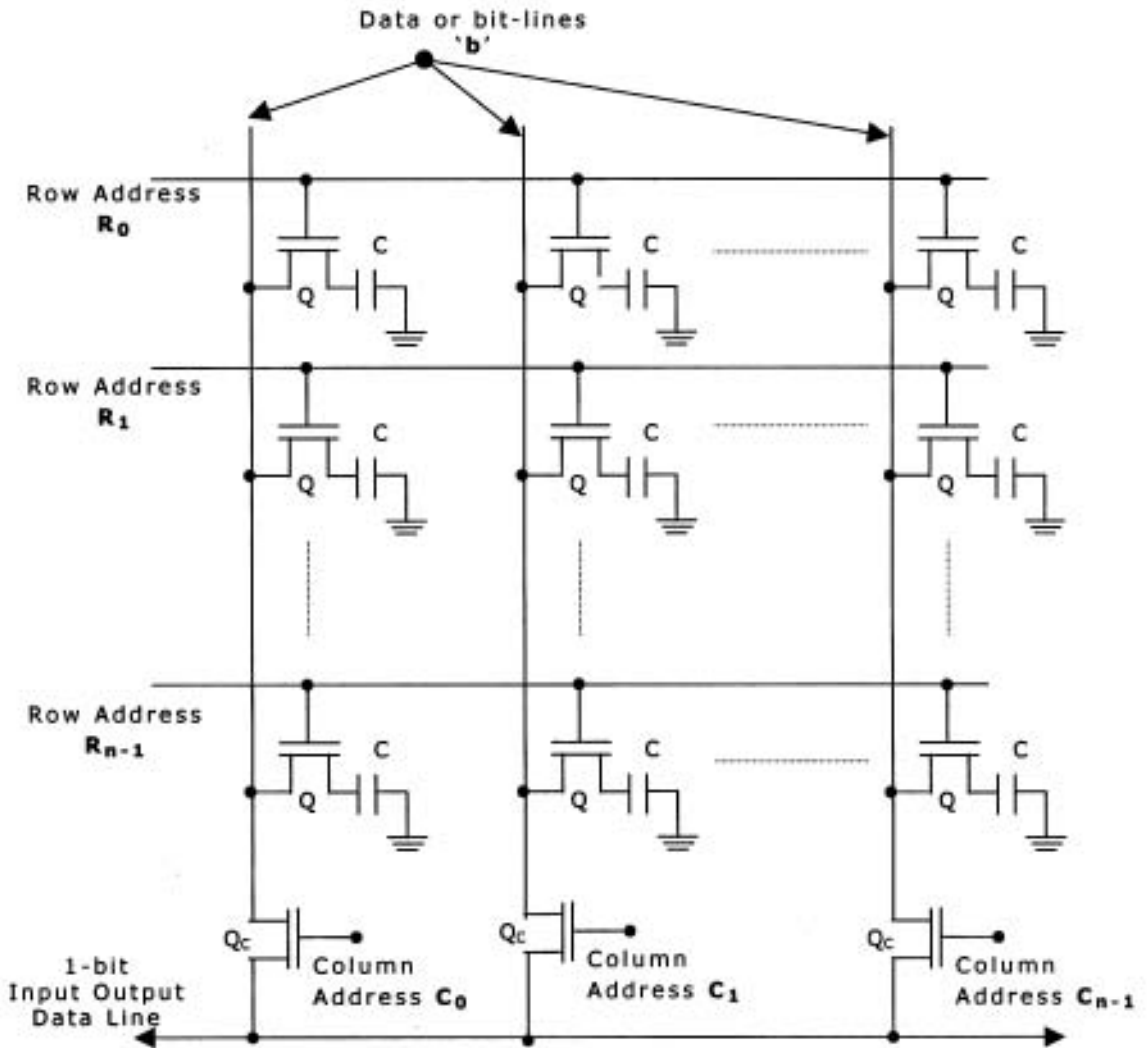


Fig. 11.33 Organization of  $n \times 1$  MOS DRAM Memo

The organization of a DRAM allows many cells to be accessed together by a minimum amount of circuitry. A DRAM memory contains an array of basic dynamic storage cells as shown in Fig. 11.33. Readers are advised to study the organization carefully. The organization is such that upon activation of a row all the cells of that row are selected. Then cells transfer the stored information to their corresponding bit line. Thus all the cells of the activated row must be subsequently written back i.e. refreshed, to restore the information. Application of column address will turn ON any one of the  $Q_c$  transistors connected to the bit line. Thus the data bit from desired cell is obtained which is then fed to the sense amplifier before transferring to the output. Similarly for the write operation only one of the  $Q_c$  transistors will be turned ON and the data present at the input is transferred to the desired cell. From the above discussion it is clear that for any operation with the DRAM the row address must

be applied first and then column address is applied. Moreover due to the fact that charge stored in DRAM cell is very small the actual read and write processes are too intricate. Thus selecting more than one cell at a time to read and write may create their own unique problems. For this reason DRAMs are organized to have a word length of 1-bit. An external circuitry (device) called DRAM controller is used along with DRAMs to attain periodic refresh of DRAM. Since activation of a row refreshes all the cells in that row, the DRAM controller simply activates all the rows to refresh entire DRAM. The DRAM controller also takes care about all the timing requirements of DRAM chip like after what time span the refresh operation must be initiated.

11.4.13.4 The DRAM Structure

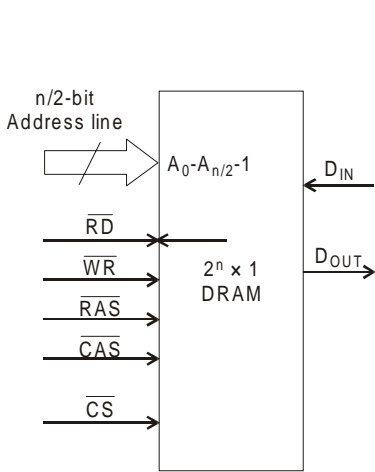


Fig. 11.34(a) General Block Diagram of a  $2^n \times 1$  DRAM

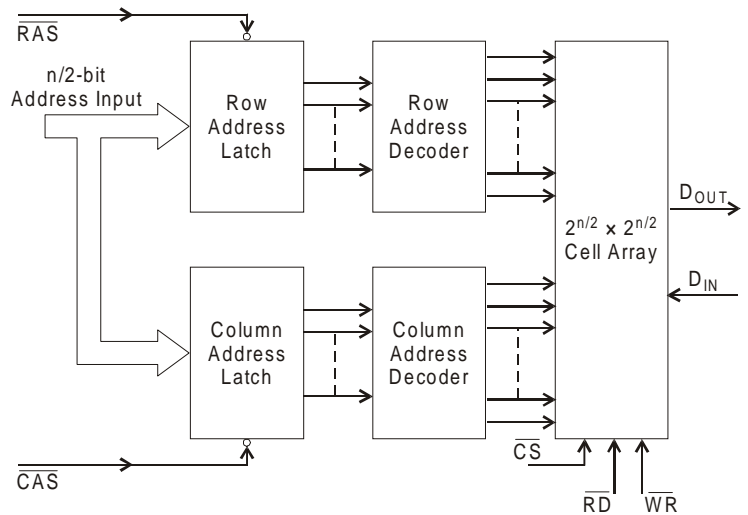
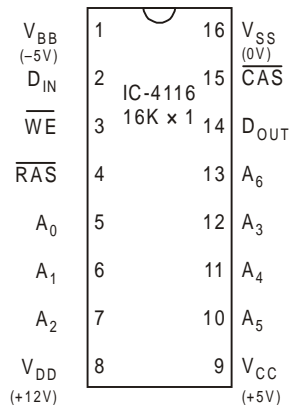


Fig. 11.34(b) General structure of A  $2^n \times 1$  DRAM

The generalized structure of DRAM memory is shown by Fig. 11.34. Let us first turn our attention to the generalized block diagram of a DRAM shown in Fig. 11.34(a). Since DRAMs have higher storage capacity the requirements of address lines are also large. That means too many pins have to be devoted to the address bus. To deal with, manufacturers have multiplexed the address bits in two groups; the row address and the column address. That's why the diagram shows  $n/2$ -bit address bus. The two signals  $\overline{RAS}$  and  $\overline{CAS}$  are used to tell when row address and when the column address respectively, is applied. RAS stands for *row address strobe* and CAS stands for *column address strobe*. A low on any of these two lines strobes the address present on address bus into an internal latch. The latch can be row address latch if RAS is activated or column address latch if CAS is activated, as shown in Fig. 11.34(b). Output of these two latches are used by the two decoders to address the required cell. Note that the  $\overline{CAS}$  also performs the chip select function in most of the DRAMs. As previously stated for any operation the first the row address followed by  $\overline{RAS}$  is applied. After that the column address of desired cell followed by  $\overline{CAS}$  is applied. The DRAM controller handles this sequence for the users and fulfills the required timing between the signals. In fact DRAM controllers are invisible for users from operational point of view, even though they actually communicate with them only. It is DRAM controller that actually communicates with DRAM IC by means of various signals. A practically useful DRAM is IC-4116 whose pin

diagram is shown in Fig. 11.35. This IC employs architecture identical to the one shown in Fig. 11.34(b) but utilizes a 128 X 128 cell array. Note that the IC does not have any chip select signal thus it is performed by the  $\overline{\text{CAS}}$  signal.



**Fig. 11.35** Pin Diagram of IC 41111 A 111Kx1 DRAM

**Refreshing DRAMs.** To refresh, the DRAM controller utilizes the fact that activation of a row refreshes all the cells connected to that row. The DRAM controller simply outputs a row address and activates the  $\overline{\text{RAS}}$  signal to refresh a row. Such a memory cycle is called as RAS cycle. The total number of such cycles needed to refresh all the rows constitute one refresh cycle that refreshes the entire DRAM. The DRAM controller initiates a refresh cycles after every predetermined time interval as specified by data sheet of DRAM. Yet another method is to interleave the refresh operation in which only one row is refreshed at a time and each successive row is refreshed after a fixed amount of time. Consider for example a DRAM having 5 rows and after 5 seconds data are lost. Then it is require to refresh each successive row within 1 seconds so that all the rows are refreshed once in every 5 seconds. Readers are encouraged to figure out situations where interleaved refreshing is useful and compare this scheme with refreshing all the rows at once. In either the scheme when refresh operation is taking place the DRAM can not be accessed.

**Simultaneous Module.** As shown in figure the DRAMs have a word length of 1-bit where as most modern computers requires to read or write an m-bit word at time. Therefore m identical DRAM ICs are connected in parallel to give m-bit word length. This is done by mounting the ICs on a single board called, *Single Inline Memory Module* or simply SIMM module. For example IC 41256 is 256K x 1 DRAM where as 41256A8 is a SIMM module using 8 41256 ICs that offers a 256K x 8 DRAM memory.

**Page Addressing Mode.** A special addressing mode that is available with most DRAMs to support the transfer of large data block is page addressing mode. In this case the data bits are stored in successive cells of the same row. Thus for the data transfer the row address has to be specified only once and by specifying the successive column address the data bits can be read. The data transfer rate becomes double in this case as compared to the normal case in which each time a cell is accessed both the row & column address must be issued.



**Pseudostatic DRAMs.** Newer DRAMs are available in which the refresh logic is incorporated in the memory chip itself. In this case the dynamic nature of these memories are almost invisible. Such a dynamic memory is referred as pseudostatic, or simply PDRAMs. Note that when internal refreshing is going on the device can not be accessed.

#### 11.4.14 SRAMs and DRAMs

Having been studied the SRAMs & DRAMs it is worth to devote some time in comparative study of the two memories. Here are the few, which must be appreciated for the two types.

- The SRAMs cells can either be Bi-polar or MOS but DRAM cells can only be MOS.
- The DRAM cells are much simpler than the SRAM cells.
- SRAMs can retain information as long as power is ON where as DRAMs loose the retained information after a small time and thus requires refreshing.
- Data storage in DRAMs involves charge storage where as in SRAMs it involves ON/OFF the transistors.
- DRAM cells can be constructed around a single transistor where SRAMs may require 11 transistors per cell. Thus DRAMs offer higher storage density. Infact the DRAMs are among the densest VLSI circuits in terms of transistor per chip. Due to this almost all the RAMs greater than 16KB are DRAMs.
- DRAMs are cost effective memory solution than the SRAMs.
- The actual read/write mechanism of DRAMs are much more tedious than that of SRAMs.
- DRAMs are always organized with the word length of 1-bit so that they are used with SIMM module where as SRAMs are organized for many different word lengths.
- DRAMs require many hardware pins so that address lines have be multiplexed where as it is seldom the case with SRAMs.
- Timing requirements of DRAMs are very complex as compared to the timing requirements of SRAMs.
- The DRAMs suffer from destructive read out so that each read must be followed by a write whereas there is no such problem with SRAMs.
- The DRAM requires extra hardware circuitry, called DRAM Controllers, for its proper operation where as SRAM does not need such supports.
- DRAMs are slower than the SRAMs due to destructive read out, address multiplexing and mainly due to refresh requirements.
- Future expansion of DRAM is easier as they are used with SIMM module in which case expansion involves replacement of board on the slot of main board. Needless to mention that using SIMM further increases the storage capacity than achieved by using a single DRAM chip.
- The DRAM cells offer lesser power consumption compared to either BJT or MOS SRAM cell.

At this point we strongly recommend the readers to find out similar comparison between other memory devices like BJT & MOS SRAMs, BJT & MOS ROMs & PROMs etc.



### 11.4.15 Memory System Design

An important aspect of study of memories to design a memory system that suits for a particular application. A design engineer is suppose to design a memory system that meets number of required storage words and the word length with the available resources. The memory system design main involves the two things

- The Address Space Allocation or Address decoding.
- Obtaining the required storage capacity with a given word length.

The first point requires the understanding of address range available with memory IC and that with the available with processing system. The second problem mainly requires different types of connection that can be made with the memory ICs to achieve the required capacity and word length. We access these steps one by one in following subsections to elaborate the idea. It has been our experience that students are often feared about how to determine number of address lines, how to find out address ranges and how it is written. Thus we first address this issue in order to make the newcomers feel easy.

#### 11.4.15.1 Determining Address Lines & Address Range

To determine the number of address lines in a memory IC we first see its organization (e.g.  $1K \times 8$ ) to find out the number of storage locations or words in that memory. For example in  $1K \times 8$  memory  $1K=1024$  storage locations or words are present. Now by using equation 2 we determine the number of address lines in a memory. We have equation 2 as

$$2^n \geq \text{Number of Storage Locations}$$

Where

$n$  = number of address lines and is always integer.

<b>Address bits</b> →	$A_9$	$A_8$	$A_7$	$A_6$	$A_5$	$A_4$	$A_3$	$A_2$	$A_1$	$A_0$	<b>Address In Hex</b>
Starting Address	0	0	0	0	0	0	0	0	0	0	000 <sub>H</sub>
Ending Address	1	1	1	1	1	1	1	1	1	1	3FF <sub>H</sub>

**Fig. 11.36** Address Bit Map or Address Range for  $1K \times 8$  Memory

So for  $1K \times 8$  memory  $2^n \geq 1K = 1024$  results in  $n = 10$ . i.e. it have 10 address lines. The address bits are written as  $A_0, A_1 \dots A_{n-1}$  i.e., for  $n = 10$  they are labeled as  $A_0, A_1 \dots A_9$ . To determine the address range we use the fact that addresses are specified as binary bit and each binary bit can have max change from 0 to 1. Thus to determine range we consider this max change for address bits at a time as shown in Fig. 11.36 for  $1K \times 8$  memory. While writing the memory addresses the usual way is to write in hexadecimal instead of binary for the sake of easiness.

Thus the address ranges from  $000_H$  to  $3FF_H$  for a  $1K \times 8$  memory. The bit wise representation of memory address as shown in Fig. 11.36 is called as *Address Bit Map*. We will use such type of bit maps in designing problems. From the above we can conclude that adding  $3FF_H$  to the initial address will give the last address of  $1K$  memory locations. The same can be said alternately, that by changing the 10 address bits we traverse through any of the  $1K$  address location.

11.4.15.2 Parallel and Series Connections of Memory

In this subsection we are concerned with how connect more than one memory ICs to get the desired word length and desired number of address location. In general we can connect the memory in three different configurations. They are

- Parallel Connection
- Series Connection
- Series-Parallel Connection

The *parallel connection* of memory ICs are used when we need to increase the word length while keeping the address location same. In this case the address lines and control signals are tied together by forming wired-OR and data lines from all the ICs are taken at the output to provide desired word length. Let we have two ICs organized as  $1K \times 4$  and we need a memory of  $1K \times 8$ . A Parallel connection shown in Fig. 11.37 can solve this issue.

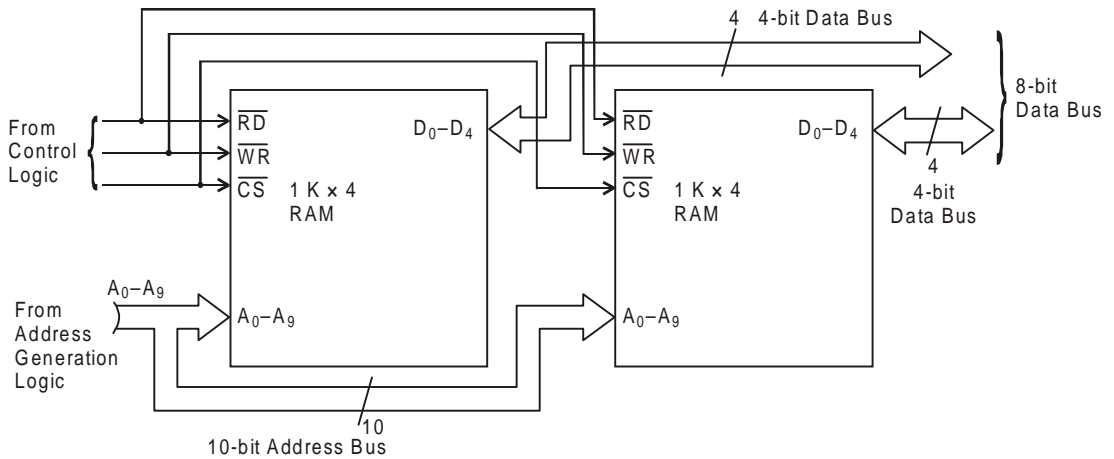


Fig. 11.37 Parallel connection of  $1024 \times 4$  RAMs to obtain  $1024 \times 8$  RAM

Observe the figure carefully and notice that 4-bit data bus from two ICs are taken at the output in parallel giving a total of 8-bit data bus, that why it is called parallel connection. Any group of the four bits can be defined least significant or most significant.

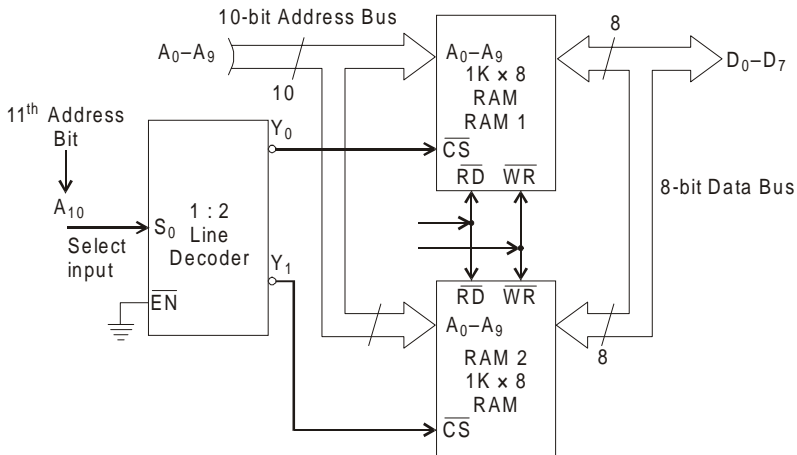


Fig. 11.38 (a) Series connection of two  $1K \times 8$  RAMs to obtain  $2K \times 8$  RAM

The *series connection* of memory ICs are used when we need to increase the address location while keeping the word length same. In this case the address lines, data lines and control signals are all tied together by forming wired-OR. A decoder can then be used to select one of the ICs. This provides the desired numbers of address locations. Let we have two ICs organized as  $1K \times 8$  and we need a memory of  $2K \times 8$ . A series connection shown in Fig. 11.38(a) can solve this issue.

Since there are two ICs of  $1K$  locations each, when combined together they can give  $2K$  locations. But having  $2K$  locations requires 11-address bits. The select input of 2:1 line decoder provides this 11th address bit as labeled in figure. When this bit 0 the upper RAM is selected and when 1 lower RAM is selected. Thus  $2K$  locations can be accessed. The address bit map for this connection is shown in Fig. 11.38(b). Observe how the address is changing for the two values of  $A_{10}$  which is connected to the select input of decoder. Thus we can say that we have address bank of  $2K$  locations. As clearly seen by the bit map 1st  $1K$  addresses of this bank are assigned to the upper RAM and 2nd  $1K$  are assigned to the lower RAM. Since the two RAMs are connected one after another they are called to be series connected.

Address bits →	$A_{10}$	$A_9$	$A_8$	$A_7$	$A_6$	$A_5$	$A_4$	$A_3$	$A_2$	$A_1$	$A_0$	Address In Hex
Starting Address	0	0	0	0	0	0	0	0	0	0	0	000 <sub>H</sub>
Ending Address	0	1	1	1	1	1	1	1	1	1	1	3FF <sub>H</sub>
Starting Address	1	0	0	0	0	0	0	0	0	0	0	400 <sub>H</sub>
Ending Address	1	1	1	1	1	1	1	1	1	1	1	7FF <sub>H</sub>

**Fig. 11.38** (b) Address Bit Map or Address Range for  $1K \times 8$  Memory

The *series-parallel connection* of memory ICs are used when we need to increase both the word length and the number of address locations. This is the combination of first two connections. If readers are through with the first two configurations this can easily be understood. For this type of connection we can consider the connection shown in figure 11.38(a) in which each of the  $1K \times 8$  ICs are derived from Fig. 11.37. Readers are instructed to draw the connections to obtain a memory of  $2K \times 8$  by using the 4-ICs of  $1K \times 4$  on the basis of Fig. 11.37 and 11.38(a). In this fashion the desired word length and desired number of storage locations can be obtained. In fact it is the connection often required.

#### 11.4.15.3 Address Space Allocation—The Address Decoding

The address-decoding problem is associated with what addresses to be assigned to which memory devices from entire address space available, and what exactly should be the scheme to assign it. In nutshell it is the problem of memory address allocation. In a general microprocessor system the number of address lines are large allowing it to access large amount of memory. Where as the memory ICs available may be of smaller storage capacities i.e. they can process smaller number of address lines. Now the problem could be which of the ICs are to be given which address. Infact the problem can be solved by careful analysis of requirement and applying the appropriate decoding scheme.

There are two types of decoding schemes, *Exhaustive Decoding* and *Partial Decoding*. In the exhaustive or fully decoded system all the address lines of processor must be decoded. In case of partial decoding scheme all the address lines of processor are not decoded. But in this case the addresses may not be unique and more than one address can be assigned to same memory word.

To understand it let us consider a residential colony in which house numbers can be a 4-digit numbers and any of the house can be allotted a number between 0000 to 9999. Now suppose only the numbers from 1000 to 2000 are allotted to houses. In this case saying house numbers 500 or even 3500 or 4500 is same as 1500 as we know that only house numbers are one from 1000 to 2000. Saying just 500 as house number means *Partial Decoding* in case of memory address decoding. Where as saying house number 1500 is *Exhaustive Decoding*. Now consider the case when allotted house numbers are 1000 to 3000 then saying just 500 to mention same house creates trouble that whether it refers to 1500 or 2500 as both are present. In memories it means two memory ICs are assigned same address. So both the ICs will start working at a time causing *BUS Contention* to occur. *BUS Contention* is a situation in which more than one device attempts to drive the system bus simultaneously in which case busses may be damaged. To avoid all such problems we must give complete house number i.e. 1500 to refer to that house. Thus before saying 500 we must ensure that there is no other house number whose last 3 digits are 500. The same is ensured in memory systems too, to avoid bus contention. Thus irrespective of the decoding schemes the goal of design is to assign unambiguous (or unique) addresses to memory devices. The way two schemes are realized are explained in next subsection.

#### 11.4.15.4 Formation of Memory System

In this section we are concerned with how to deal with the requirements of different systems. Here we must analyze the different requirement given to us and check the resources available and then take the appropriate steps to solve the problem. Infact a design engineer is supposed to come with the appropriate access circuitry for the memory modules given to him/her in order to obtain the required system. It has our best observation and reading that it is far better if we consider the problems to illustrate the ideas to students. So we start with the example in this subsection to consider the different situation and its fulfillment. *Note that we illustrate many important facts and make the important conclusions through these examples only.* We first start with the assumption that processing system can be a microprocessor or any other system capable of providing the required number address bits and control signals required for memory. Our job is mainly to find out the decoding circuitry for the circuits.

**Example 1.** *It is required to obtain a memory system of  $2K \times 8$  bits for a certain application. Given that memory ICs available are  $1K \times 8$ . Obtain the desired system.*

**Solution.** By reading the problem it is revealed that we have already solved this problem even before saying any thing!!!!. If surprised see Fig. 11.38-A, it is same as required by problem. Sit comfortably now because we are already through with it. For the reference we reproduce the Fig. 11.38-A and 11.38-B again.

It's now time to list out the different steps in order to solve these types of problems. Carefully read all steps we take to solve this problem again.

1. > *Determine the total number of address lines required for the memory system to be designed.*

Given total requirement is  $2K \times 8$  means we need a total of 11 address lines because  $2^{11} = 2048 = 2K$ .

- > Determine the number of address bits available with the given memory ICs.

Given ICs are  $1K \times 8$  means we have 10 address lines available with the individual ICs because  $2^{10} = 1024 = 1K$ .

- > Determine the number of bits available for the chip decoder.

Since total required address bits are 11 and individual memories can handle 10-bits of address, we are left with 1 address bit. This extra bit can be feeded to the decoder as shown in the Fig. 11.39 (a).

- > Draw the Address Bit Map for Entire Memory Space and indicate change in bits for individual memory ICs. This is needed to assign particular address to individual ICs. The same is shown Fig. 11.39 (b).

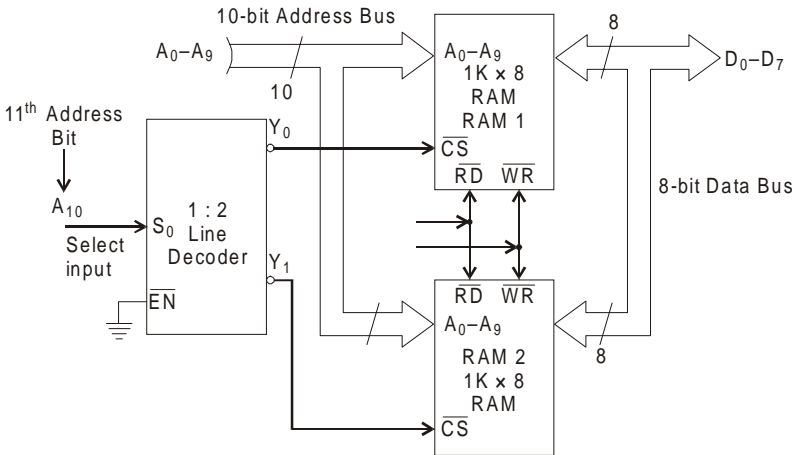


Fig. 11.39 (a) Series connection of two  $1K \times 8$  RAMs to obtain  $2K \times 8$  RAM

	Address bits handled by memory ICs											HEX Addr	IC
	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>		
Start Addr	0	0	0	0	0	0	0	0	0	0	0	000 <sub>H</sub>	RAM 1
End Addr	0	1	1	1	1	1	1	1	1	1	1	3FF <sub>H</sub>	
Start Addr	1	0	0	0	0	0	0	0	0	0	0	400 <sub>H</sub>	RAM 2
End Addr	1	1	1	1	1	1	1	1	1	1	1	7FF <sub>H</sub>	

Fig. 11.39 (b) Address Bit Map for Fig. 11.39(a) of example 1

Observe that bit  $A_{10}$  does not change anywhere within the address range for RAM1 and remains 0 always. The same is true for RAM2 where bit  $A_{10}$  is 1 always. This justifies the step 3 to take the left over bits for decoder. As one can see that the decoder bit selects how much memory address (1K in this case) can be selected by each output of the decoder. Infact this is the main purpose of drawing address bit maps.

- > Draw the Connection diagram for the memory ICs as per Address Bit Map. This is the one we have done already in Fig. 11.39(a). But this done for this example only. We will draw the connection diagram in this step only for the other examples.

- > Draw the Address Map. In step we diagrammatically represents the entire memory address space and show which address is assigned to which IC as shown in Fig. 11.39(c).

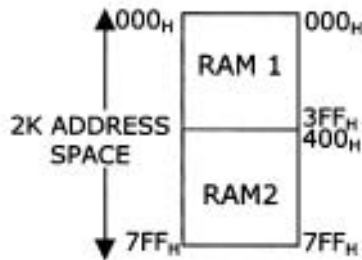


Fig. 11.39 (c) Address Map for figure 11.39-A

From this address map we also conclude that adding 7FF<sub>H</sub> to the initial address will give the last address of 2K memory locations.

If you are waiting for step 7, we are sorry to say that problem is over and you have given the complete solution to the problem posed to you. We advise the readers to go to the beginning of this example and carefully read it again, as these are the findings we would be using in following examples.

**Example 2.** Given two 2K × 8 ROM ICs and two 2K × 8 RAM ICs obtain a memory system of 8K × 8 bits.

**Solution.** We start step by step as defined by the previous problem.

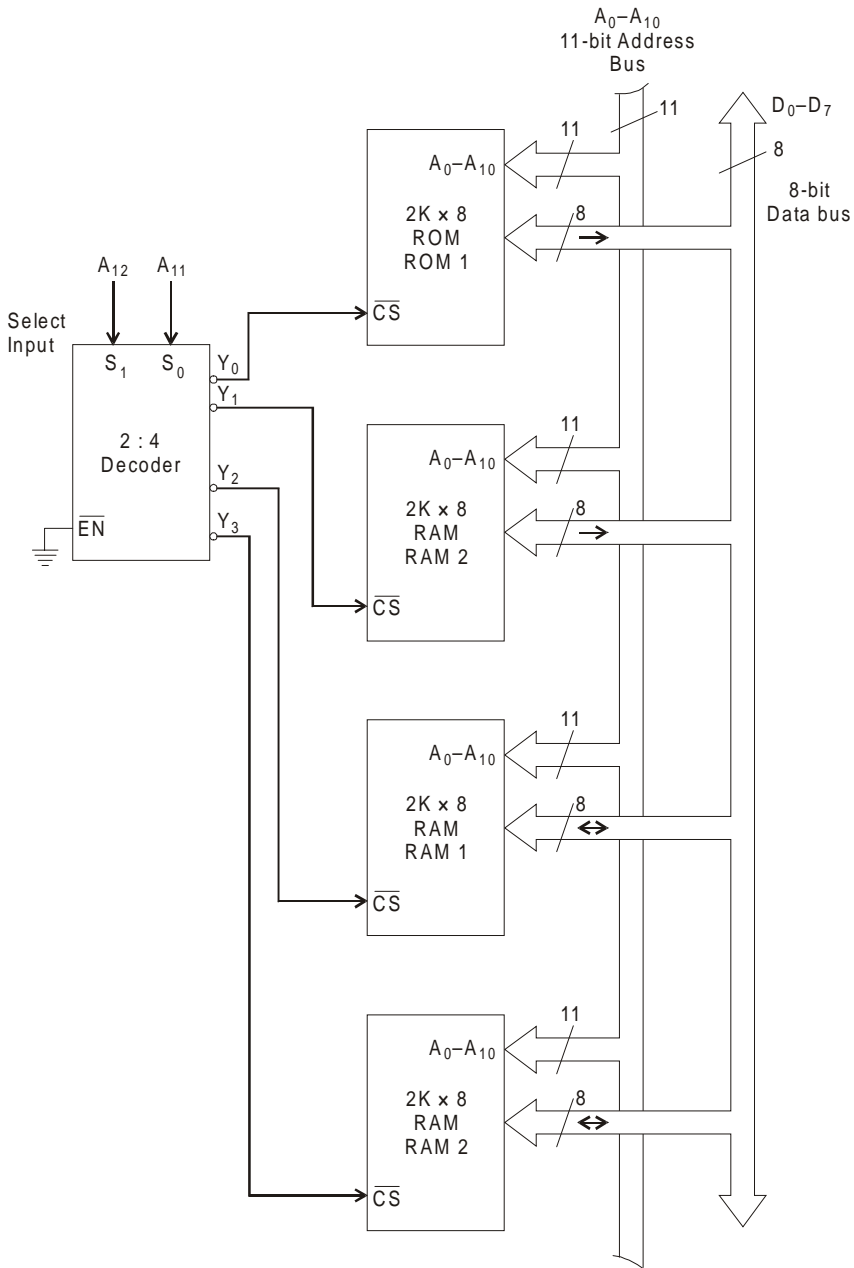
- > Given total requirement is 8K × 8 means we need a total of 13 address lines because 2<sup>13</sup> = 8192 = 8K.
- > Given ICs are 2K X 8 means we have 11 address lines available with the individual ICs because 2<sup>11</sup> = 2048 = 2K.
- > Since total required address bits are 13 and individual memories can handle 11-bits of address, we are left with 2-address bit, which can be feeded to the decoder.
- > Draw the Address Bit Map for Entire Memory Space and indicate change in bits for individual memory ICs. This is needed to assign particular address to individual ICs. The same is shown in Fig. 11.40(a).

	Decoder bit														
	A <sub>12</sub>	A <sub>11</sub>	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	HEX Addr	IC
Start Addr	0	0	0	0	0	0	0	0	0	0	0	0	0	0000 <sub>H</sub>	ROM 1
End Addr	0	0	1	1	1	1	1	1	1	1	1	1	1	07FF <sub>H</sub>	
Start Addr	0	1	0	0	0	0	0	0	0	0	0	0	0	0800 <sub>H</sub>	ROM 2
End Addr	0	1	1	1	1	1	1	1	1	1	1	1	1	0FFF <sub>H</sub>	
Start Addr	1	0	0	0	0	0	0	0	0	0	0	0	0	1000 <sub>H</sub>	RAM 1
End Addr	1	0	1	1	1	1	1	1	1	1	1	1	1	17FF <sub>H</sub>	
Start Addr	1	1	0	0	0	0	0	0	0	0	0	0	0	1800 <sub>H</sub>	RAM 2
End Addr	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF <sub>H</sub>	

Fig. 11.40 (a) Address Bit-Map for 8K × 8 Memory System of example 2

It is clear from the address bit map that each decoder output (selected by A12 and A11 bits) selects one 2K-memory address space.

- 5. > Connection diagram for the memory ICs is shown in Fig. 11.40 (b).
- 6. > The Address Map is shown in Fig. 11.40 (c).



**Fig. 11.40 (b)** Formation of 8KB memory system using RAMS and ROMS

From this address map we also conclude that adding  $1FFF_H$  to the initial address will give the last address of 8K memory locations. Alternately we can say that we must change 13 address bits at a time to traverse through 8K memory locations.



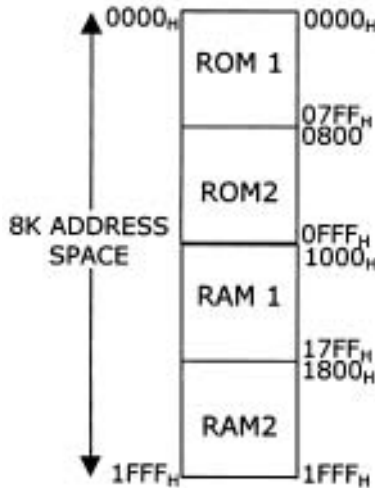


Fig. 11.40 (c) Address Map for Fig. 11.40(b).

**Example 3.** Given two 2K × 8 ROM ICs and two 2K × 8 RAM ICs obtain a memory system of 8K × 8 bits. It is required that 1st 2K address must be assigned to a ROM, 2nd 2K address to a RAM, 3rd 2K address to a ROM, and the 4th 2K address to a RAM.

**Solution.** Read the statement once again it is essentially the same as previous problem, but the only difference is now it specifies the address where the particular memory devices are to be connected.

We know that for 8K address we need 13 address lines and with the arrangement shown in Fig 11.40B it reveals that we can access four 2K memory segments whose addresses can be obtained as shown in bit map of Fig. 11.40(a).

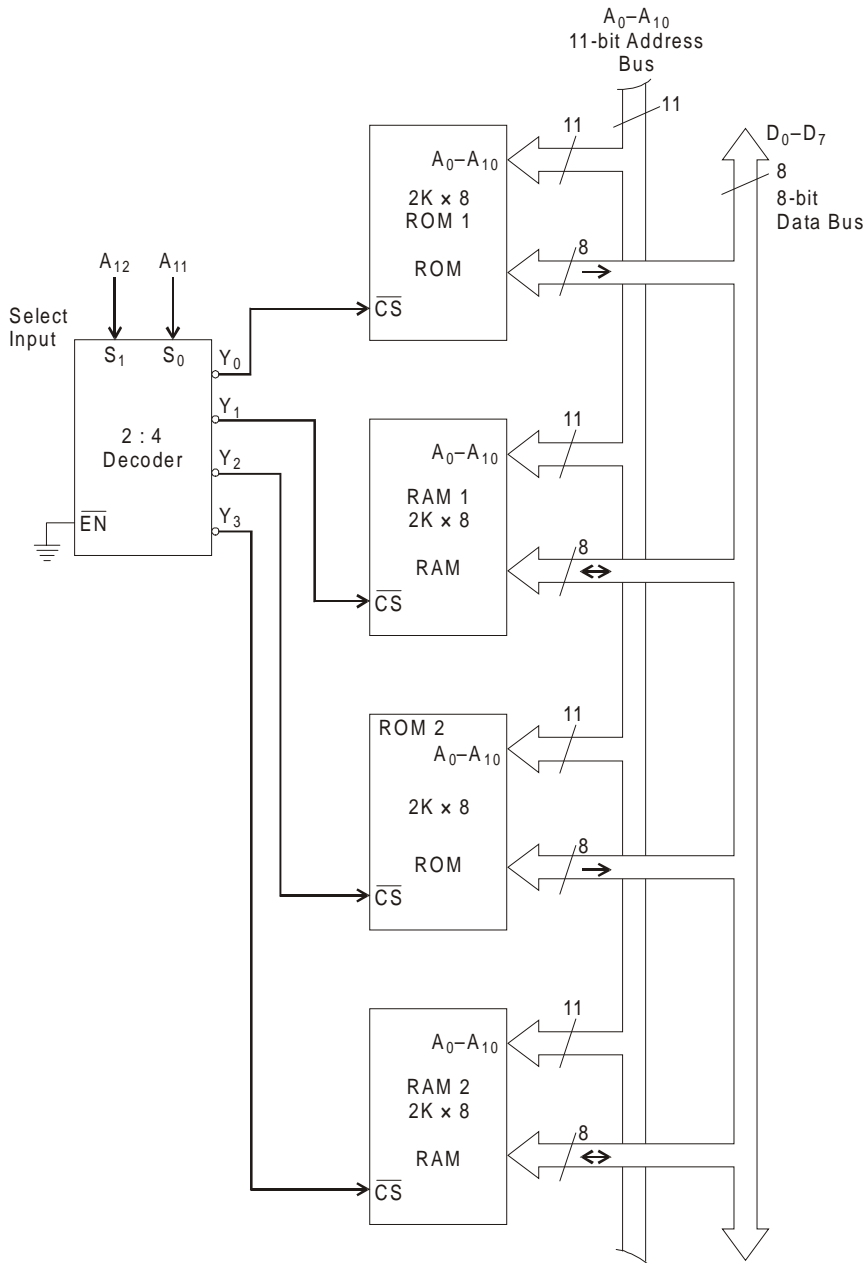
	Decoder bit														
	A <sub>12</sub>	A <sub>11</sub>	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	HEX Addr	IC
Start Addr	0	0	0	0	0	0	0	0	0	0	0	0	0	0000 <sub>H</sub>	ROM 1
End Addr	0	0	1	1	1	1	1	1	1	1	1	1	1	07FF <sub>H</sub>	
Start Addr	0	1	0	0	0	0	0	0	0	0	0	0	0	0800 <sub>H</sub>	RAM 1
End Addr	0	1	1	1	1	1	1	1	1	1	1	1	1	0FFF <sub>H</sub>	
Start Addr	1	0	0	0	0	0	0	0	0	0	0	0	0	1000 <sub>H</sub>	ROM 2
End Addr	1	0	1	1	1	1	1	1	1	1	1	1	1	17FF <sub>H</sub>	
Start Addr	1	1	0	0	0	0	0	0	0	0	0	0	0	1800 <sub>H</sub>	RAM 2
End Addr	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF <sub>H</sub>	

Fig. 11.41 (a) Address Bit-Map for 8K × 8 Memory System of example 3

Now the only thing we need to do is to connect the device at correct outputs of the decoder, which can be obtained from bit map shown in Fig. 11.41(a). Go through the details of this map carefully.

The similar connection diagram and memory address map would also appear but with the ICs changed to positions required by problem. Fig. 11.41(b) and 11.41(c) shows this.

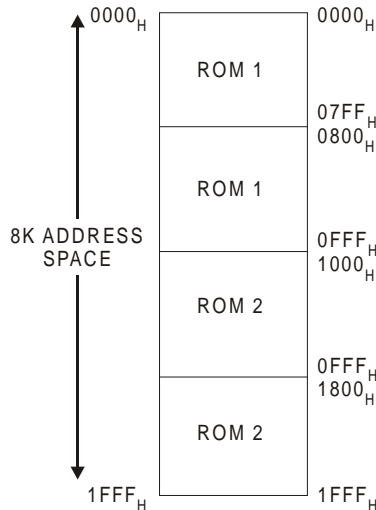




**Fig. 11.41 (b)** Formation of 8KB memorysystem using RAMS and ROMS

**Example 4.** Given two  $2K \times 8$  ROM ICs and two  $2K \times 8$  RAM ICs obtain a memory system of  $8K \times 8$ . It is required that two ROMs must be connected to addresses beginning from  $0000H$  and  $1000H$ . Also the RAMs should be connected to begin from  $0800_H$  and  $1800_H$

**Solution.** Example 4 is just another way or giving statement for problem example 3. But we will consider it separately to understand that what should be the way to solve these types of issues.



**Fig. 11.41** (c) Address Map for Fig. 11.41(b)

Except the step 4 i.e., generating address bit map, all other steps are exactly same so that we consider only address bit map that too in much simplified way. Recall that in example 1 we concluded that adding 7FF<sub>H</sub> to the initial address would give the last address of 2K memory locations. We use this to generate the bit map as given ICs are 2K × 8.

*For ROMs:* given that a ROM should start from 0000<sub>H</sub> and 1000<sub>H</sub>

First when connected at 0000<sub>H</sub>:

$$\text{Last address} = 0000_{\text{H}} + 07\text{FF}_{\text{H}} = 07\text{FF}_{\text{H}}$$

So the address range is 0000<sub>H</sub> to 07FF<sub>H</sub>

Second when connected at 1000<sub>H</sub>:

$$\text{Last address} = 1000_{\text{H}} + 07\text{FF}_{\text{H}} = 17\text{FF}_{\text{H}}$$

So the address range is 1000<sub>H</sub> to 17FF<sub>H</sub>

*For RAMs:* in the similar way we get for RAMs

First when connected at 0800<sub>H</sub>:

Address range is 0800<sub>H</sub> to 0FFF<sub>H</sub>

Second when connected at 1800<sub>H</sub>:

Address range is 1800<sub>H</sub> to 1FFF<sub>H</sub>

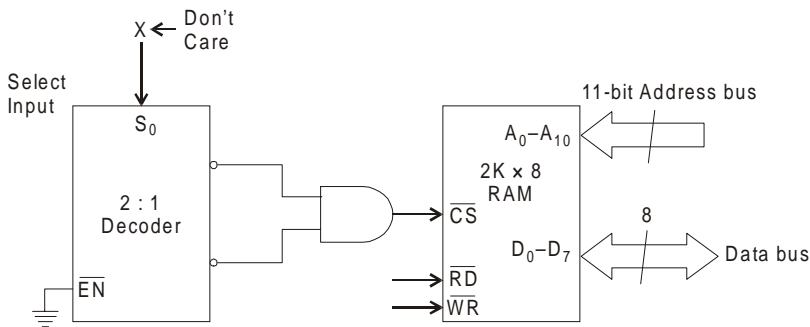
Compare these addresses with those shown in the bit map of Fig. 11.41 (a), you reached the same map. Thus the entire Fig. 11.41 can redrawn for example 4.

It is interesting to see that instead of adding 07FF if we would have changed the least 11 significant address bits (as  $2^{11} = 2\text{K}$ ) of initial address to 1 even in that case also we could have reached the same point. Infact this is what we have done till example 3. Either the way we proceed it is same because we found the number 07FFH from the fact that the highest hexadecimal number that can be written by 11 address bits is 07FFH. Many a times you will find that adding a number to initial address is easier because it is less time consuming than case of drawing complete bit map. But this short hand is useful only when you are through with concept and saying a hexadecimal number displays its bit pattern in your mind.

**Shadowing or Address Overlapping** address overlapping means assigning more than one address space to the memory device. It is the situation similar with a student who has his roll number and university enrolment number unique to him. So that when called by enrolment number or by roll number the same student will respond and none of the other students respond. An example will illustrate this best.

**Example 5.** A system needs a  $4K \times 8$  memory system. But the requirement is such that a single  $2K \times 8$  RAM IC be used and the two consecutive  $2K$  memory addresses should be overlapped to this device only.

**Solution.** We know that for  $4K$  system we will have 12 address lines. The  $2K$  devices can process only 11 address bits thus the left over is 1-bit that can be used to access two  $2K$  segments. By address overlapping we mean that both the segments would be referenced for same device. This is shown in Fig. 11.42.



**Fig. 11.42** Illustration of memory address overlapping with  $2K \times 8$  IC

Notice the use of AND gate used to generate chip select for memory device. As shown, upon activation decoder outputs goes low so weather activated output is  $Y_0$  or  $Y_1$  AND gate out output is always low. This means memory is selected irrespective of the status of select input of decoder.

If we draw (readers are instructed to draw) the bit map for  $4K$  as two  $2K$  address block we see that first  $2K$  block is  $000$  to  $7FF_H$  and second  $2K$  block is  $800_H$  to  $FFF_H$ . With the configuration shown in Fig. 11.42 both the address blocks refers to the same IC thus address blocks are overlapped to each other and hence the name address overlapping. In the similar way we can overlap any two equal sized address blocks present in entire address map (or space).

We now turn our attention to exhaustive and partially decoded memory system. For illustration purpose we again consider the problem posed in example 2. Because write now we are very much through with it and once we understand the idea we can apply it to design. Read the problem statement carefully as it is going to create the difference.

**Example 6.** It is required to obtain an  $8K \times 8$  memory system for  $8085$  microprocessor system that has an addressing capability of  $64K$  locations. Given memories are  $2K \times 8$  ROM ICs and  $2K \times 8$  RAM ICs. Obtain the exhaustive or fully decoded as well as partially decoded system.

**Solution.** We first determine the number of address lines this processor has.  $64K$  means  $2^{16}$  i.e., the processor has 16 address bits where as the requirement for memory is just

13-bits, see step 1 in example 2. This means 3 address bits of processor will remain unused (or unchanged). This is shown by the address bit maps in Fig. 11.43, which now contains 16 bits instead of 13.

*For exhaustive decoding:* As stated for this scheme we must decode all the address bits of microprocessor. The address bit map for this scheme is shown in Fig. 11.43(a). It shows that bit  $A_{13}$  to bit  $A_{15}$  are unchanged for entire 8K-address space. Since in this scheme we must decode all address bits, we can use these 3-bits to generate  $\overline{EN}$  signal for Decoder which can also be used enable or disable the memory system by enabling/disabling decoder.

	← Unchanged bit →			Decoder bit		← Address bits handled by memory ICs →												
	$A_{15}$	$A_{14}$	$A_{13}$	$A_{12}$	$A_{11}$	$A_{10}$	$A_9$	$A_8$	$A_7$	$A_6$	$A_5$	$A_4$	$A_3$	$A_2$	$A_1$	$A_0$	HEX Addr	IC
Start Addr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000 <sub>H</sub>	ROM 1
End Addr	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	07FF <sub>H</sub>	
Start Addr	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0800 <sub>H</sub>	RAM 1
End Addr	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0FFF <sub>H</sub>	
Start Addr	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1000 <sub>H</sub>	ROM 2
End Addr	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	17FF <sub>H</sub>	
Start Addr	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1800 <sub>H</sub>	RAM 2
End Addr	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF <sub>H</sub>	

Fig. 11.43(a) Address Bit-Map for exhaustive decoded 8K × 8 Memory System of example 6

The diagram for decoder along with enable generation is shown in Fig. 11.43 (b). The figure indicates which decoder output to be connected to which IC. Infact connections will be same as Fig. 11.40 (b) and the decoder of Fig. 11.43 (b) will replace decoder of Fig. 11.40 (b).

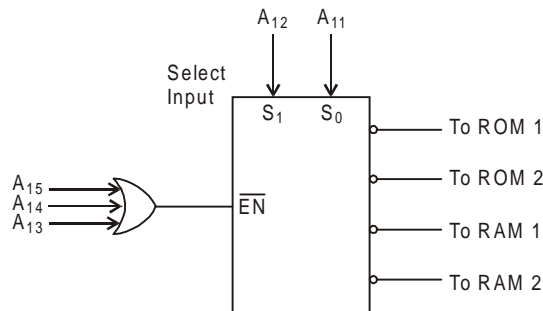


Fig. 11.43 (b) Exhaustive decoded system for example 6.

Notice the use of OR gate. Output of OR gate is low only when all its inputs are low otherwise high. Thus saying an address higher than 1FFF<sub>H</sub> will not be responded by this system. If we say 8000H it means bit  $A_{15}$  goes high and output of OR gate goes high thus disabling the decoder and consequently the entire memory system.

*For partial decoding:* Here the idea is not to use all the address bits of processor for address decoding. As the 3 most significant address bits  $A_{15}$ ,  $A_{14}$ ,  $A_{13}$  are unchanged we can say that they are not used i.e. don't care. So it hardly matters if we use them for decoding or not. The address bit map shown in Fig. 11.44 (a) show this.

	Unused bits			Decoder bit	Address bits handled by memory ICs												HEX Addr	IC
	A <sub>15</sub>	A <sub>14</sub>	A <sub>13</sub>	A <sub>12</sub>	A <sub>11</sub>	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>		
Start Addr	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0000 <sub>H</sub>	ROM
End Addr	X	X	X	0	0	1	1	1	1	1	1	1	1	1	1	1	07FF <sub>H</sub>	1
Start Addr	X	X	X	0	1	0	0	0	0	0	0	0	0	0	0	0	0800 <sub>H</sub>	RAM
End Addr	X	X	X	0	1	1	1	1	1	1	1	1	1	1	1	1	0FFF <sub>H</sub>	1
Start Addr	X	X	X	1	0	0	0	0	0	0	0	0	0	0	0	0	1000 <sub>H</sub>	ROM
End Addr	X	X	X	1	0	1	1	1	1	1	1	1	1	1	1	1	17FF <sub>H</sub>	2
Start Addr	X	X	X	1	1	0	0	0	0	0	0	0	0	0	0	0	1800 <sub>H</sub>	RAM
End Addr	X	X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF <sub>H</sub>	2

Fig. 11.44 (a) Address Bit-Map for partially decoded 8K × 8 Memory System of example 6

This means irrespective of the status of A<sub>15</sub>, A<sub>14</sub>, and A<sub>13</sub> the memory system can be used. Thus Fig. 11.40 (b) is direct implementation of partially decoded system, as it does not include these bits.

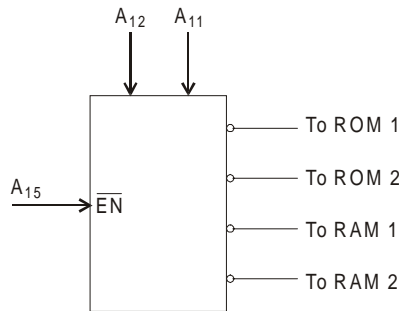


Fig. 11.44 (b) Partially decoded system for example 6.

A variation of implementation is shown in Fig. 11.44-B, which uses A<sub>15</sub> to enable the address decoder. Since A<sub>14</sub> and A<sub>13</sub> are not used it is still a partially decoded system.

**Example 7.** It is required to obtain an 8K × 8 memory system for 8085 microprocessor system that has an addressing capability of 64K locations. Given memories are 2K × 8 ROM ICs and 2K × 8 RAM ICs. Obtain the exhaustive decoded system, which maps the 8K-memory system to begin from 8000<sub>H</sub>.

**Solution.** It essentially the same problem as was done in example 6. The change now is to assign the particular address. For this already we have seen example 4. So we do not go to complete design rather we just draw the bit-map and decoder circuit required by stating the net outcomes of detailed design.

The only change in bit map of Fig. 11.45 (a) as compared to that of Fig. 11.43 (a) is just the status of A<sub>15</sub>, which is set to 1 now. It is because 8K system has to begin from 8000<sub>H</sub>. No other change can be observed in this bit map. So we connect A<sub>15</sub> through not gate to the input of OR gate to generate the enable signal for this address range. This is shown in Fig. 11.44 (b).

The readers are directed draw an address map for this design as was drawn in Fig. 11.40(c) of example 2.

	← Unchanged bit →			Decoder bit	← Address bits handled by memory ICs →												HEX Addr	IC
	A <sub>15</sub>	A <sub>14</sub>	A <sub>13</sub>	A <sub>12</sub>	A <sub>11</sub>	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>		
Start Addr	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000 <sub>H</sub>	ROM
End Addr	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	87FF <sub>H</sub>	1
Start Addr	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	8800 <sub>H</sub>	RAM
End Addr	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	8FFF <sub>H</sub>	1
Start Addr	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	9000 <sub>H</sub>	ROM
End Addr	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	97FF <sub>H</sub>	2
Start Addr	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	9800 <sub>H</sub>	RAM
End Addr	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	9FFF <sub>H</sub>	2

Fig. 11.45(a) Address Bit-Map for exhaustive decoded 8K × 8 Memory System of example 7

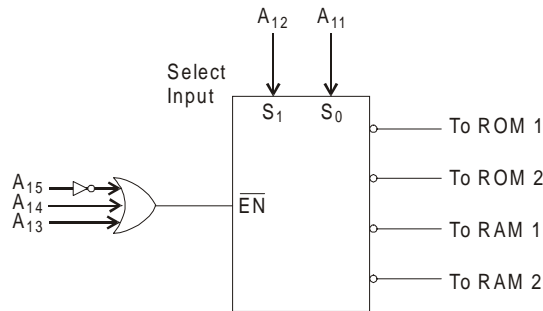


Fig. 11.45 (b) Exhaustive Decoded system of Example 7

Although all the above examples does not contain all types design variation we hope that it will serve as the best building block to understand the problem of memory bank formation. Notice that same system was taken to illustrate the different ideas because once we are familiar with a system then it is easy to view it in different situations. In general the problems define IC numbers not the organization (e.g., 2K × 8) of memories for which user must know the address bus width and data bus width. For this reason readers are advised to prepare a list of ICs and memorize the bus width and control signals.

11.5 EXERCISES

1. What is meant by primary memory, secondary memory and mass storage devices?
2. Define the terms storage capacity, memory word, storage location, and word length.
3. Determine the storage capacity of the memories shown in Fig. 11.46.

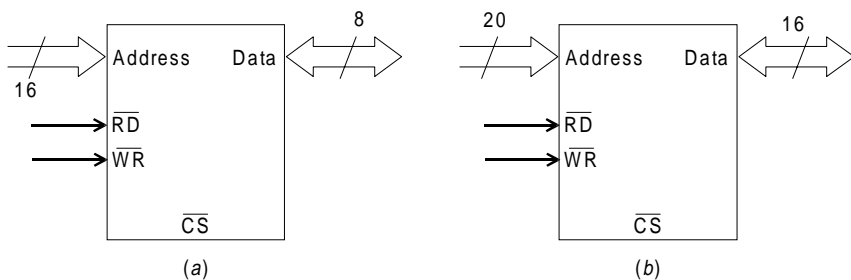


Fig. 11.46 Memories for problem 3

4. Define the terms (i) Access Time (ii) Cycle time (iii) Bandwidth (iv) Volatile (v) Dynamic (vi) Static
5. Explain organization of tracks in magnetic memories with the help of figure ?
6. What are optical memories ?
7. Explain basic semiconductor memory organization with the help of figure.
8. With the help of figure explain the memory cell addressing.
9. In the Fig. 8-b how memory cell "Cell 3-2" is accessed ?
10. Explain the memory cell organization shown in Fig. 9. How a particular cell can be selected in this case ?
11. A memory has cell array of  $114 \times 114$ , what will be the required number of lines of word length is (a) 2-bits (b) 4-bits (c) 8-bits.
12. Explain the sequence of operation to be performed to write a data into memory ?
13. A memory has access time of 50 n sec. It has an additional delay called precharge time of 20 n sec. Find out the access rate and bandwidth of this unit ?
14. Explain how a multiplexer can be used as ROM ?
15. What are Mask Programmed ROM ? How logic 1s logic 0 are stored in such ROMs.
16. Explain the MOS PROMS ? What are their advantages over Bipolar PROMs ?
17. Explain the basic structure of a EPROM cell ? Why they are so popular ?
18. How an EPROM can be reprogrammed ?
19. Explaining the basic logic structure of SRAM ?
20. With the help of figure explain the operation of a bipolar SRAM cell and MOS SRAM cell ?
21. What is Dynamic RAM ? Why they are most widely used ?
22. With the help of figure explain the basic DRAM storage cell ?
23. Draw the organization of a DRAM ? How a particular cell can be read or write in this organization ?
24. What are DRAM controllers ? Why they are needed for operation of DRAM ?
25. What is partial decoding and exhaustive decoding ?
26. For a 16 Address-bit processor systems it is required to form 32 KB memory system using one 2764, Four 4016 and two 6264 memory ICs. Obtain the decoding circuitry and draw the address map. Use exhaustive decoding.
27. A system needs to create 16KB of memory using 6224 ICs. It is required that the 2nd and 3rd 8K address should overlap. Obtain the decoding circuitry and draw the address maps.
28. For a 16 address bit processor it is required to obtain 16KB memory system using IC 6232. Use partial decoding schemes and obtain the decoding circuitry and address maps ?

## REFERENCES

1. A.K. Singh, Manish Tiwari, *Digital Principles Function of Circuit Design and Application*, New Age International Publishers, Delhi, 2006.
2. H. Taub, D. Schilling, *Digital Integrated Electronics*, McGraw-Hill, Koga Kusha, 1997.
3. A.S. Sedra, K.C. Smith, *Microelectronics Circuits*, 4th ed, Oxford University Press, New York, 1998.
4. J. Millman, H. Taub, *Pulse Digital and Switching Waveforms*, Mc-Graw Hill, Singapore.
5. M.M. Mano, *Digital Design*, 2nd ed, Prentice-Hall of India, 1996.
6. R.L. Tokheim, *Digital Electronics: Principles and Applications*, 6th ed, Tata McGraw-Hill, New Delhi 2004.
7. J. Millman, C.C Halkias, *Integrated Electronics: Analog and Digital Circuits and Systems*, Tata McGraw-Hill, New Delhi, 1994.
8. A.P. Malvino, D.P. Leach, *Digital Principles and Applications*, 4th ed, Tata McGraw Hill, New Delhi, 1991.
9. R.P. Jain, *Modern Digital Electronics*, Tata McGraw-Hill, New Delhi, 1992.
10. Virendra Kumar, *Digital Technology; Principles and Practice*, New Age, International Publishers, Delhi.
11. J.P. Hayes, *Computer Architecture and Organization*, 2nd ed, McGraw-Hill, Singapore, 1988.
12. V.C. Hamacher, Z.C. Vranesic, S.G. Zaky, *Computer Organization*, 4th ed, McGraw-Hill, 1996.
13. Gopalan, *Introduction to Digital Microelectronics Circuits*, Tata McGraw Hill, 1998.
14. P.K. Lala, *Digital System Design using Programmable Logic Devices*, BS Publication, Hyderabad, 2003.
15. J.M. Rabey, *Digital Intergrated Circuits: A Design Perspective*.
16. Charles H. Roth, Jr., *Fundamentals of Logic Design*, 4th ed, Jaico Publishing House, 2003.
17. ZVI Kohavi, *Switching and Finite Automata Theroy*, 12th ed, Tata McGraw Hill, 1978.



# APPENDICES

---

## APPENDIX A

### INTEGRATED CIRCUITS FABRICATION FUNDAMENTALS

In 1948, Schokley, Baardeen and Brattain, invented the transistor device. This device was not only cheap, more reliable, less power consuming, but side by side very small in size as compared to the vacuum tubes prevailed in those days. As the size of the active component (i.e., those components, which produces gain, viz., vacuum-tube amplifier etc) decreases, so the passive components (viz., resistors, capacitors, coils etc.) also got very much reduced in size, making the complete electronic circuitry very small. The development of “Printed Circuit Boards” (PCBs), further reduced size of electronic equipments as a whole, by eliminating bulky wiring and tie-points.

In early 1960s, a new field called as **Micro Electronics** was developed, to meet the requirements of Military, which wanted to reduce the size of its electronics equipments, to about 1/10th of its existing one. This led to the development of micro-electronic circuits, later called **integrated Circuits (ICs)**, which were so small that the fabrication was done under microscopes.

An IC can be described as a complete microelectronic-circuit in which both the active and passive components are fabricated on a extremely thin chip of silicon (called as substrate), in general.

ICs are produced by the same processes, as are used for making individual diodes, transistors etc. In such circuits, different components are isolated from one another by isolation diffusion, within the crystal-chip, and are inter-connected by an aluminum layer, that serves as wires. J.K. Kilby was the first to develop (in 1958) such an IC—a single monolithic silicon chip, with both the active and passive components. He was soon followed by Robert Noyce, whose IC included the interconnection, on the chip itself. It had about 10 individual components, the chip was 3 mm (0.12 inch) square. The development of Large-Scale Integration (LSI) during the early 1970s made it possible to pack thousands of transistors and other components on a chip of roughly the same size. This technology gave rise to the **Micro-Processor** on IC, that contained all the arithmetic logic and control circuitry needed to carryout the functions of a digital computer’s Central Processing Unit (CPU). Very Large Scale Integration (VLSI), developed during the 1980 has vastly increased the circuit-density of microprocessors, as well as of “memory” and support chips. This technology has yield, 30, 00,000 transistors on a chip of less than 4 cm chip.

ICs are of several types: Mono-lithic ICs, also called semi-conductor or silicon integrated circuits. These are formed by the superposition of layers of materials, in various patterns,

to form a single, three-dimensional block of circuitry. The circuit elements are formed by the particular patterns and topology of conducting, semi-conducting, and insulating layers, which have been used in building up the structure. The silicon “chip” is also called as **Wafer** (or Substrate). Thus all the components are automatically part of the same chip. Active components like diodes, transistors, as well as passive ones like resistors, capacitors etc., are all fabricated at appropriate spots in the wafer, using epitaxial diffusion-technology. These types of ICs are in wide use because for mass-production, this (monolithic) process has proved to be more economical.

The other type of ICs is called as “Thick & Thin Film IC.” These IC’s are not formed within a silicon wafer, but on the surface of insulating substrate, such as glass ceramic material. Only passive components (resistors, capacitors) are formed through thick or thin film techniques, on the insulating surface. The active components (transistors, diodes) are added externally as discrete elements, to complete a functional circuit. These discrete active components, in their turn, are generally produced by using the monolithic process. To produce resistors and conductors using materials of different resistivity, the width and the thickness of the film is varied on the insulating surface. Capacitors are produced by switching an insulating oxide film between two conducting films. This deposition is done through a mask. Small Inductors can be made by depositing a spiral formation of the film. Another method is called “Cathode Sputtering”, in which atoms from a cathode made of the desired film material, are deposited on the substrate, which is located between the cathode and anode. However, both these (i.e., vacuum evaporation and cathode sputtering), methods come under thin film technology. In thick-film ICs, silk-screen printing technique is used. These screens are made of fine stainless steel wire meshes and the “Inks” are pastes (of pulverized glass and aluminium) which have conductive, resistive or dielectric properties. After printing, the circuits, are high-temperature fired in a furnace, to fuse the films to the insulating substrate (glass or ceramic). Here also, active components are added externally, as discrete components.

The third type of ICs is called Hybrid or Multi-Chip ICs, Here a number of chips are interconnected by a combination of film and monolithic IC techniques. In such ICs, active components are first formed within a silicon wafer (using monolithic technique), which is later covered with an insulating layer (such as  $\text{SiO}_2$ ). Then they use the film technology to form the passive components on the  $\text{SiO}_2$  surface. Connections are then made for the film to the monolithic structure through **Windows** cut to the  $\text{SiO}_2$  layer.

Of these three types of ICs, monolithic types are most common with the advantage of lowest cost and highest reliability. But in these ICs (*i*) isolation in between the components is poorer, (*ii*) inductors cannot be fabricated alone, (*iii*) their range and a flexibility in design is very limited, because film ICs are made on an insulating surface, inter-components isolation is better. Further, because discrete active components are connected externally, it gives more flexibility in circuit design. The multi-chip ICs are the costliest and least reliable of the three types.

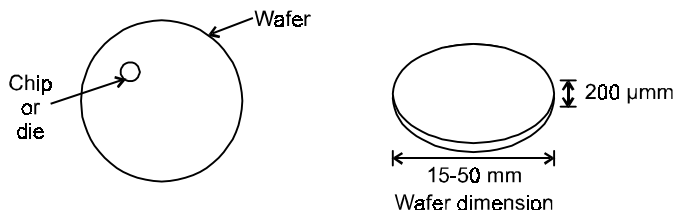
ICs may also be classified on the basis of their function. The two most important types are (*i*) Linear and (*ii*) Digital. Linear ICs are also called **Analog ICs** because their inputs and outputs can take on a continuous range of values, and the outputs are generally proportional to inputs (linearly). Linear IC’s are quickly replacing their discrete circuit components in many applications, making the new unit more reliable, because so many external connections (major source of circuit failure) are eliminated. The LICs (Linear Integrated Circuits) are now being used as (1) Operational Amplifiers, (2) Small-signal Amplifiers, (3) Power Amplifiers, (4) R/F, I/F amplifiers, (5) Microwave Amplifiers, (6) Multipliers, (7) Voltage comparators, (8) Voltage Regulators etc.

However, about 80% of the IC market has been captured by Digital type of ICs (mostly the Computer Industry). These are mostly monolithic in construction. Such ICs use input and output voltages, limited to only two possible levels-low or high, as digital signals are usually binary. Sometimes, these are also called as Switching Circuits. Such ICs include (1) Logic Gates, (2) Flip Flops, (3) Counter, (4) Clock-chips, (5) Calculator chips, (6) Memory chips, (7) Micro processors etc. In addition to the various ICs described above, we have monolithic ICs based on MOSFET structure. These are also processed in the same way as bipolar ICs, however, their fabrication is much easier as in a single diffusion step, both the drain and source regions can be formed (compared to 2-4 steps needed in case of BJTs-ICs). Likewise, there is no need for isolation, technique for the Enhancement MOSFET devices. Since in their case, the drain and source regions are isolated from each other by a P-N Junction formed with the N-type substrate. Besides, packing-density of MOS ICs is at least 10 times maximum than that for Bipolar ICs. The MOS resistor occupies less than 1% of the area, as compared to the one by a conventionally diffused resistor. This high packing density makes MOS ICs especially suited for LSI and VLSI mass productions. Their main disadvantage is slower operating speed, as compared to Bipolar ICs.

### IC FABRICATION TECHNIQUE (STEPS)

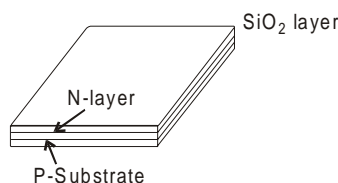
The actual process of IC fabrication consists of a number of steps. Which are described briefly as:

1. **WAFER PREPARATION:** A **Wafer** is a thin slice of a semiconductor material die 15-50mm (mostly silicon) either circular or Wafer dimension rectangular in shape. These wafers are prepared on formed by cutting a P-type silicon bar into very thin slices (see Fig. A.1). These wafers, after being lapped and polished to micro-finish, same as the base (or substrate) for hundreds of ICs.



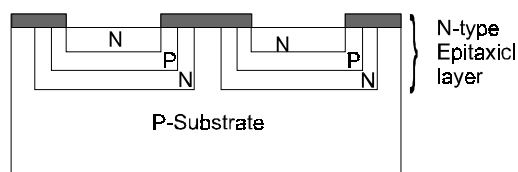
**Figure A.1** Wafer Preparation

2. **EPITAXIAL GROWTH:** An N-type of silicon layer (about 15  $\mu\text{m}$  thick) is grown on the P-type substrate, by placing the wafer in a furnace at 1200°C and introducing a gas containing phosphorus (donor impurity). It is in the epitaxial layer, that all active and passive components of IC are formed. This layer ultimately becomes the collector for transistors, or on element for a diode or a capacitor.
3. **OXIDISATION:** A thin layer of silicon dioxide ( $\text{SiO}_2$ ) is grown on the N-type layer by exposing the wafer to an oxygen atmosphere at about 1000°C (Fig. A.2).



**Figure A.2** Epitaxial growth and oxidation

4. PHOTO-LITHOGRAPHIC PROCESS: Here selective etching of  $\text{SiO}_2$  layer is carried out with the help of a photographic mask, photo resist and etching solution. Thus, selective areas of the N-layer became subjected to diffusion.
5. ISOLATION DIFFUSION PROCESS: which divides the layer into N-type Islands on which active components may be fabricated.
6. BASE AND EMITTER DIFFUSION: After getting the islands formed on the N-type layer, the P-type base of transistor is now diffused into the N-type layer, which itself acts as a collector. The sequence of steps is the same as 4 and 5 i.e., by the use of photo-resists and mask, which creates windows in the  $\text{SiO}_2$  layer. This is carried out in a phosphorus (donor impurity) atmosphere. Next, N-type emitter is diffused into the base, after another photoresist and mask process. No further N-type diffusion is needed for the resistor, which uses the resistivity of the P-type metal itself for this purpose. Thus, one N-P-N transistor and one resistor fabricated simultaneously. (Fig. A.3).

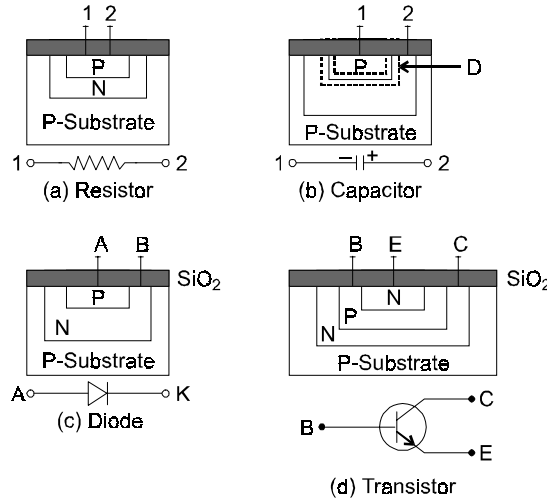


**Figure A.3** Fabrication of transistor and resistor

7. PRE-OHMIC ETCH:  $\text{N}^+$  regions are diffused into the structure to ensure good metal-ohmic contact. This is done once again by the  $\text{SiO}_2$  layer, photo resist and masking process.
8. METALLIZATION: It is done for making inter-connections and provide bonding pads around the circumference of the chip, for later connection of wires. It is carried out by evaporating aluminum over the entire surface and then selecting etching among the aluminum to leave behind the desired Inter-connecting condition pattern, and bonding pads.
9. CIRCUIT PROBING: In this step, each IC or the wafer is checked electrically for *its* proper working. This is done by placing probes on the bonding pads. Faulty chips are marked, isolated and discarded, after the wafer has 'scribed' and broken down into individual chips.
10. SCRIBING and SEPARATING: After metallisation and test probing, the wafer is broken down into individual chips, containing the ICs. For this the wafers are first scribed with a diamond tipped tool and then separated into single chips.
11. MOUNTING and PACKING: As the individual chip is very small, it is cemented or soldered to a gold-plated header, **through** which leads **have already** been connected. After this the ICs are thermetically sealed. These can be (a) Top Hat package, which can accommodate 14 pins at its top. If the cover is of metal, it provides magnetic shielding to the ICs, which is not possible with ceramic or plastic top, (b) Flat packages, which are quite cheap and widely used for industrial and consumer applications, where high temperatures are not met. Ceramic is used

in those cases. Dual-in line (DIL) flat packages are more convenient for circuit break use than top hat ones, because being flatter, they allow high circuit densities.

12. ENCAPSULATION: After placing a cap over the circuit, sealing is done in an inert atmosphere. A few of the fabrications and their symbols are given in Fig. A.4. In (a) we have the resistor, In (b) a capacitor and in (c) a diode and in (d) a transistor.



**Figure A.4**

### ADVANTAGES OF ICs

The ICs are very popular, and being used in various electronic circuits. The astounding progress in Micro-Electronics, has been because ICs have the following plus points:

- (1) Extremely small size
- (2) Too little weight

Weight and size are of great importance in military and space applications:

- (3) Very much low cost
- (4) Extremely high reliability

Because of the absence of soldered connections. IC logic gate has been found to be 10 times more reliable than a vacuum tube logic gate, and too times more reliable than the BJT logic gate. High reliability indirectly means that ICs work for longer periods without any trouble-something very desirable, both for military and consumer application.

- (5) Low power consumption
- (6) Easy replacement

ICs are hardly repaired, because in case of failure, It is more economical to replace them than to repair.

- (7) ICs are ideally suited for small-signal operation, as the various components of a IC are located very close to each other, In or On as silicon wafer, then chance of electrical pick up is preferably nil.

## APPENDIX B

## DIGITAL ICs

<b>74LS00</b>	4x Two input NAND
<b>74LS01</b>	4x Two input NAND, Open collector
<b>74LS02</b>	4x Two input NOR
<b>74LS03</b>	4x Two input NAND, Open collector
<b>74LS04</b>	6x Inverter (NOT)
<b>74LS05</b>	6x Inverter (NOT), Open collector
<b>74LS06</b>	6x Inverter (NOT), High voltage Open collector
<b>74LS07</b>	6x Buffer (NO-OP), High voltage Open collector
<b>74LS08</b>	4x Two input AND
<b>74LS09</b>	4x Two input AND, Open collector
<b>74LS10</b>	3x Three input NAND
<b>74LS11</b>	3x Three input AND
<b>74LS12</b>	3x Three input NAND, Open collector
<b>74LS13</b>	2x Four input, Schmitt Trigger NAND
<b>74LS14</b>	6x Inverter (NOT), Schmitt Trigger
<b>74LS15</b>	3x Three input AND, Open collector
<b>74LS16</b>	6x Inverter (NOT), High voltage Open collector
<b>74LS17N</b>	6x Buffer (NO-OP), High voltage Open collector
<b>74LS19</b>	6x Inverter (NOT), Schmitt Trigger
<b>74LS20</b>	2x Four input NAND
<b>74LS21</b>	2x Four input AND
<b>74LS22</b>	2x Four input NAND, Open collector
<b>7423</b>	2x Four input NOR with Strobe
<b>7425</b>	2x Four input NOR with Strobe
<b>74LS26</b>	4x Two input NAND, High voltage
<b>74LS27</b>	3x Three input NOR
<b>74LS28</b>	4x Two input NOR
<b>74LS30</b>	Eight input NAND
<b>74LS31</b>	6x DELAY (6nS to 48nS)
<b>74LS32</b>	4x Two input OR
<b>74LS33</b>	4x Two input NOR, Open collector
<b>74LS37</b>	4x Two input NAND
<b>74LS38</b>	4x Two input NAND, Open collector
<b>74LS39</b>	4x Two input NAND, Open collector

<b>74LS40</b>	4x Two input NAND, Open collector
<b>74LS42</b>	Four-to-Ten (BCD to Decimal) DECODER
<b>74LS45</b>	Four-to-Ten (BCD to Decimal) DECODER, High current
<b>74LS46</b>	BCD to Seven-Segment DECODER, Open Collector, lamp test and leading zero handling
<b>74LS47</b>	BCD to Seven-Segment DECODER, Open Collector, lamp test and leading zero handling
<b>74LS48</b>	BCD to Seven-Segment DECODER, lamp test and leading zero handling
<b>74LS49</b>	BCD to Seven-Segment DECODER, Open collector
<b>7450</b>	2x (Two input AND) NOR (Two input AND), expandable
<b>74LS51</b>	(a AND b AND c) NOR (c AND e AND f) plus (g AND h) NOR (i AND j)
<b>7453</b>	NOR of Four Two input ANDs, expandable
<b>74LS54</b>	NOR of Four Two input ANDs
<b>74LS55</b>	NOR of Two Four input ANDs
<b>74LS56P</b>	3x Frequency divider, 5:1, 5:1, 10:1
<b>74LS57P</b>	3x Frequency divider, 5:1, 6:1, 10:1
<b>74S64</b>	4-3-2-2 AND-OR-INVERT
<b>74S65</b>	4-3-2-2 AND-OR-INVERT
<b>74LS68</b>	2x Four bit BCD decimal COUNTER
<b>74LS69</b>	2x Four bit binary COUNTER
<b>7470</b>	1x gated JK FLIPFLOP with preset and clear
<b>7472</b>	1x gated JK FLIPFLOP with preset and clear
<b>74LS73</b>	2x JK FLIPFLOP with clear
<b>74LS74</b>	2x D LATCH, edge triggered with clear
<b>74LS75</b>	4x D LATCH, gated
<b>74LS76A</b>	2x JK FLIPFLOP with preset and clear
<b>74LS77</b>	4x D LATCH, gated
<b>74LS78A</b>	2x JK FLIPFLOP with preset and clear
<b>74LS83</b>	Four bit binary ADDER
<b>74LS85</b>	Four bit binary COMPARATOR
<b>74LS86</b>	4x Two input XOR (exclusive or)
<b>74LS90</b>	Four bit BCD decimal COUNTER
<b>74LS91</b>	Eight bit SHIFT register
<b>74LS92</b>	Four bit divide-by-twelve COUNTER
<b>74LS93</b>	Four bit binary COUNTER
<b>74LS94</b>	Four bit SHIFT register
<b>74LS95B</b>	Four bit parallel access SHIFT register



<b>74LS96</b>	Five bit SHIFT register
<b>74LS107A</b>	2x JK FLIPFLOP with clear
<b>74LS109A</b>	2x JK FLIPFLOP, edge triggered, with preset and clear
<b>74LS112A</b>	2x JK FLIPFLOP, edge triggered, with preset and clear
<b>74LS114A</b>	2x JK FLIPFLOP, edge triggered, with preset
<b>74LS116</b>	2x Four bit LATCH with clear
<b>74121</b>	Monostable Multivibrator
<b>74LS122</b>	Retriggerable Monostable Multivibrator
<b>74LS123</b>	Retriggerable Monostable Multivibrator
<b>74S124</b>	2x Clock Generator or Voltage Controlled Oscillator
<b>74LS125</b>	4x Buffer (NO-OP), (low gate) Tri-state
<b>74LS126</b>	4x Buffer (NO-OP), (high gate) Tri-state
<b>74LS130</b>	Retriggerable Monostable Multivibrator
<b>74128</b>	4x Two input NOR, Line driver
<b>74LS132</b>	4x Two input NAND, Schmitt trigger
<b>74S133</b>	Thirteen input NAND
<b>74S134</b>	Twelve input NAND, Tri-state
<b>74S135</b>	4x Two input XOR (exclusive or)
<b>74LS136</b>	4x Two input XOR (exclusive or), Open collector
<b>74LS137</b>	3-8 DECODER (demultiplexer)
<b>74LS138</b>	3-8 DECODER (demultiplexer)
<b>74LS139A</b>	2x 2-4 DECODER (demultiplexer)
<b>74S140</b>	2x Four input NAND, 50 ohm Line Driver
<b>74143</b>	Four bit counter and latch with 7-segment LED driver
<b>74LS145</b>	BCD to Decimal decoder and LED driver
<b>74LS147</b>	10-4 priority ENCODER
<b>74LS148</b>	8-3 gated priority ENCODER
<b>74LS150</b>	16-1 SELECTOR (multiplexer)
<b>74LS151</b>	8-1 SELECTOR (multiplexer)
<b>74LS153</b>	2x 4-1 SELECTOR (multiplexer)
<b>74LS154</b>	4-16 DECODER (demultiplexer)
<b>74LS155A</b>	2x 2-4 DECODER (demultiplexer)
<b>74LS156</b>	2x 2-4 DECODER (demultiplexer)
<b>74LS157</b>	4x 2-1 SELECTOR (multiplexer)
<b>74LS158</b>	4x 2-1 SELECTOR (multiplexer)
<b>74159</b>	4-16 DECODER (demultiplexer), Open collector
<b>74LS160A</b>	Four bit synchronous BCD COUNTER with load and asynchronous clear



<b>74LS161A</b>	Four bit synchronous binary COUNTER with load and asynchronous clear
<b>74LS162A</b>	Four bit synchronous BCD COUNTER with load and synchronous clear
<b>74LS163A</b>	Four bit synchronous binary COUNTER with load and synchronous clear
<b>74LS164</b>	Eight bit parallel out SHIFT register
<b>74LS165</b>	Eight bit parallel in SHIFT register
<b>74LS166A</b>	Eight bit parallel in SHIFT register
<b>74LS169A</b>	Four bit synchronous binary up-down COUNTER
<b>74LS170</b>	4x4 Register file, Open collector
<b>74LS174</b>	6x D LATCH with clear
<b>74LS175</b>	4x D LATCH with clear and dual outputs
<b>74LS170</b>	Four bit parallel in and out SHIFT register
<b>74LS180</b>	Four bit parity checker
<b>74LS181</b>	Four bit ALU
<b>74LS182</b>	Look-ahead carry generator
<b>74LS183</b>	2x One bit full ADDER
<b>74LS190</b>	Four bit Synchronous up and down COUNTER
<b>74LS191</b>	Four bit Synchronous up and down COUNTER
<b>74LS192</b>	Four bit Synchronous up and down COUNTER
<b>74LS193</b>	Four bit Synchronous up and down COUNTER
<b>74LS194</b>	Four bit parallel in and out bidirectional SHIFT register
<b>74LS195</b>	Four bit parallel in and out SHIFT register
<b>74LS198</b>	Eight bit parallel in and out bidirectional SHIFT register
<b>74LS199</b>	Eight bit parallel in and out bidirectional SHIFT register, JK serial input
<b>74LS221</b>	2x Monostable Multivibrator
<b>74LS240</b>	8x Inverter (NOT), Tri-state
<b>74LS241</b>	8x Buffer (NO-OP), Tri-state
<b>74LS244</b>	8x Buffer (NO-OP), Tri-state Line driver
<b>74LS245</b>	8x Bidirectional Tri-state BUFFER
<b>74LS259</b>	Eight bit addressable LATCH
<b>74LS260</b>	2x Five input NOR
<b>74LS273</b>	8x D FLIPFLOP with clear
<b>74LS279</b>	4x SR LATCH
<b>74LS283</b>	Four bit binary full ADDER
<b>74LS373</b>	8x Transparent (gated) LATCH, Tri-state
<b>74LS374</b>	8x Edge-triggered LATCH, Tri-state
<b>74LS629</b>	Voltage controlled OSCILLATOR
<b>74LS688</b>	Eight bit binary COMPARATOR

## REFERENCES

1. H. Toub, D. Schilling, Digital Integrated Electronics, McGraw-Hill, Koga Kusha, 1997.
2. A.S. Sedra, K.C. Smith, Microelectronics circuits, 4th ed., Oxford University Press, New York, 1998.
3. J. Millman, H. Taub, Pulse Digital and switching waveforms, Mc Graw-Hill, Singapore.
4. M.M. Mano, Digital Design, 2nd ed, Prentice-Hall of India, 1996.
5. R.L. Tokheim, Digital Electronics: Principles and Applications, 6th ed., Tata McGraw-Hill, New Delhi, 2004.
6. J. Millman, C.C. Halkias, Integrated Electronics: Analog and Digital circuits and systems, Tata McGraw-Hill, New Delhi, 1994.
7. A.P. Malvino, D.P. Leach, Digital Principles and Applications, 4th ed., Tata McGraw-Hill, New Delhi, 1991.
8. R.P. Jain, Modern Digital Electronics, Tata McGraw-Hill, New Delhi, 1992.
9. Virendra Kumar, Digital Technology; Principles and Practice, New Age International.
10. J.P. Hyes, Computer Architecture and Organization, 2nd ed., McGraw-Hill, Singapore, 1988.
11. V.C. Hamacher, Z.C. Vranesic, S.G. Zaky, Computer Organization, 4th ed., McGraw-Hill, 1996.
12. Gopalan, Introduction to Digital Microelectronics CKts/TMH, 1998.
13. P.K. Lala, Digital System Design using Programmable Logic devices, BS Publication, Hyderabad, 2003.
14. J.M. Rabey, Digital Integrated circuits: A design Perspective.

# INDEX

r's Complement 35  
(r - 1)'s Complement 35, 36  
1's Complement 27, 28, 29, 30, 31, 32, 33, 35, 36, 43, 61, 62  
10's Complement 34, 35, 39, 40, 60, 61, 62  
2's Complement 27, 28, 29, 30, 31, 32, 33, 34, 35, 61, 62  
4-bit Magnitude Comparator 178, 179  
9's and 10's Complement 34, 39, 60, 62  
9's complement 34, 35, 43, 60, 61, 62

## A

Absorption Law 68  
Active pullup output 428, 444  
Active region 395, 399, 424, 430, 431, 432, 435  
Active voltage 395  
Adders 143, 145, 169, 171, 176, 192  
Address decoding 205  
Algorithmic State Machine 362, 363  
Altera 204, 208  
Analog 1, 2, 3, 4, 5, 6, 7  
Analog and Digital Signals 2, 4, 5  
Analog or Continuous Signal 3  
Analog -to-digital conversion 7  
AND Gate 71, 79, 80, 84, 85, 86, 90, 92, 94, 95, 96, 97, 98, 99, 100, 104, 105, 106, 108, 109  
Antifuse 210, 211  
Arithmetic Circuits 143  
ASM Block 362, 365, 368, 370, 371, 372  
ASM chart 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 379, 381

Associative Law 67  
Asynchronous 266, 272, 273, 280, 282, 283, 288, 291, 293, 296, 298, 307, 328, 329, 331, 332, 333, 334, 335, 336, 339, 341, 345, 346, 347, 349, 351, 353, 354, 356, 360, 361  
Asynchronous Counter 272, 273, 280, 288, 291, 293, 307, 329  
Asynchronous Counter Circuits 272  
Asynchronous Sequential Circuits 214, 272  
Axiomatic Systems 65

## B

BCD 15, 16, 37, 38, 39, 41, 42, 43, 272, 287, 307, 329  
BCD Adder 174, 175, 176  
BCD Addition 38, 39  
BCD codes 41, 42, 43  
BCD Subtraction 39  
Biasing 386, 392, 393, 421, 424, 432  
Bi-directional 268, 271  
Binary 12, 13, 14, 15, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 37, 38, 40, 41, 42, 43, 45, 47, 50, 52, 53, 54, 56, 57, 58, 61, 62  
Binary Addition 25  
Binary Arithmetic 24  
Binary Division 26  
Binary Coded Decimal 15, 37, 42  
Binary Logic 8, 9, 63, 67  
Binary Multiplication 26  
Binary Signals 5, 9, 10  
Binary Subtraction 25

- Binary subtractors 146
  - Binary to gray code converter 149, 150
  - Binary to Gray conversion 45
  - Binary to Octal Conversion 19
  - Bipolar Junction Transistor 391, 392
  - Bipolar logic families 403, 438
  - Bistable 214, 215
  - Bistable Multivibrator 215
  - Bit 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 26, 27, 28, 31, 32, 33, 34, 37, 38, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 56, 57, 62
  - BJT Characteristic 395
  - Block codes 47, 48, 52
  - Boolean Functions 69, 71, 72, 75, 97, 105
  - Bound charges 385
  - Breakdown voltage 389
  - Buffer 82, 83, 92
  - Burst Error Detection 46, 47
  - Bus contention 444
  - Bytes 12, 15, 17, 16, 18
- C**
- Characteristic equation 224, 232, 234, 238, 241, 243, 262
  - Checksums 46
  - Chip 382, 402, 440, 448
  - Circuits with latches 335, 340
  - Circuits without Latches 335
  - Clamping (or Protective) Diodes 425
  - Classical Method 185
  - Classification of Signals 4
  - Clock 214, 219, 222, 223, 224, 225, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 241, 243, 246, 248, 258, 259, 261, 262, 264
  - Clock frequency 333, 334
  - clock period 333
  - Clock Skew 331
  - Clock width 333
  - Clocked D Flip-Flop 228, 229, 264
  - Clocked Sequential Circuit 214, 219, 241, 246, 258, 333, 334
  - CMOS 402, 403, 440, 441, 442, 443, 444, 446, 447, 448, 449, 451
  - CMOS Gates 440
  - CMOS Inverter 440, 441, 443, 451
  - CMOS Series 443, 444
  - CMOS to TTL Interface 447
  - Codes 12, 20, 21, 40, 41, 42, 43, 44, 47, 48, 52, 53
  - Collector current 393, 394, 395, 396, 398, 399, 417, 421, 424, 425
  - Combination Logic 10
  - Combinational circuit 141, 142, 143, 144, 146, 147, 156, 161, 164, 167, 169, 174, 177, 179, 180, 184, 185, 186, 189, 194
  - Common base current gain 394
  - Common emitter current gain 393
  - Commutative Law 67
  - Compatibility 403, 443, 444, 446
  - Compatible 351, 352, 353, 357, 358
  - Complementary MOS 402, 440
  - Complements Law 67
  - Complex PLDs (CPLDs) 206
  - Complex Programmable Logic Devices (CPLDs) 196
  - Conditional Output Box 363, 364
  - Consensus Law 68
  - Continuous systems 2
  - Continuous Time and Discrete Time Signals 4
  - Control subsystem 362, 371, 372, 376, 377, 378, 379, 380, 381
  - Counter ICs 282
  - Counters 265, 272, 273, 274, 275, 278, 280, 282, 284, 287, 288, 295, 299, 307, 309, 311, 316, 322, 323, 324, 325, 328, 329
  - CPLDs 196, 206, 207, 210
  - Critical race 345, 346, 347, 349, 353, 354, 359, 361
  - Current hogging 415, 450
  - Current sink logic 421
  - Current source logic 414
  - Cut off region 395, 396
  - Cut-in voltage 388, 391, 395

Cycle 333, 345, 346, 360, 361  
 Cyclic Codes 52, 53

**D**

Data bus 265  
 Data register 265  
 De Morgan's Law 67  
 Decade Counter 291, 292, 293, 298, 299, 307, 309, 310, 312, 321, 329  
 Decimal 12, 13, 14, 15, 16, 19, 20, 21, 22, 23, 24, 25, 26, 34, 35, 37, 38, 39, 40, 41, 42, 43, 44, 52, 61, 62  
 Decision box 363, 364, 365, 368, 370, 371, 376  
 Decoder 159, 160, 161, 164, 165, 167, 195  
 Decoders (Demultiplexers) 159  
 Delay time 398  
 De Morgan's Law 67  
 Demultiplexer 159, 160, 161, 164, 193, 195  
 Demultiplexing 160  
 DEMUX 160, 161, 195  
 Depletion region 385, 386, 387, 393, 400, 449  
 Design Procedure 141, 142, 149, 156, 161  
 Difference Amplifier 431, 432, 433, 434, 450  
 Diffusion current 384, 449  
 Digital Circuit 382, 403, 409, 448, 451  
 Digital Hardware Algorithm 363  
 Digital Logic Family 403  
 Digital Multiplexer 156  
 Digital or Discrete Signal 2, 3  
 Digital systems 4, 7  
 Diode and Gate 407, 409, 415  
 Diode Logic 407  
 Diode or Gate 409  
 Diode Transistor Logic 415  
 Direct Coupled Transistor Logic (DCTL) 415  
 Distributive Law 66, 67, 76  
 Divide-by-6-Circuit 315  
 Divide-by-9 Circuit 315  
 Don't care Map Entries 124  
 Doped semiconductor 383, 384  
 Drift current 384, 449

Duality 67, 68, 69, 78, 111  
 Duty cycle 333  
 Dynamic Hazard 181, 184  
 Dynamic Power Dissipation 442, 443, 451

**E**

ECL OR/NOR Gate 433, 434  
 Edges 363, 365  
 Electrically Programmable ROMs 197  
 Eliminating a static-0 hazard 184  
 Eliminating a static-1 hazard 182  
 Emitter Coupled Logic (ECL) 431  
 Emitter followers 433, 434, 437  
 Encoders 167  
 Enhancement Type 399, 400, 401, 402  
 Equivalent Faults 188  
 Erasable ROMs 197  
 Error Correcting Codes 47  
 Error Detecting Codes 45, 47  
 Essential Hazard 184  
 Excess 3 Code 42  
 Exclusive NOR gate 91  
 EXCLUSIVE OR gate 84, 88  
 Extrinsic Semiconductor 383, 448

**F**

Fall Time 399  
 Fan Out 405, 411, 413, 420, 421, 424, 425, 426, 427, 431, 437, 449, 450  
 FAST ADDER 172  
 Fault detection 185, 187, 188, 189  
 Fault detection problem 185  
 fault-location problem 185  
 Field Programmable Gate Arrays (FPGAs) 196, 207  
 Field Programmable ROMs 197  
 Field-Programmable Logic 199  
 Field-Programmable Logic Array (FPLA) 199  
 Figure of merit 404, 405, 426, 427, 449, 450  
 Finite state machine 362  
 Five Variable K-Map 125

- Flip-Flop 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 236, 237, 238, 239, 241, 243, 244, 247, 248, 249, 251, 257, 259, 261, 262, 264  
 Flow table 339, 340, 342, 343, 346, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 361  
 Forward biasing 386, 393, 432  
 Four Variable K-Map 120, 121  
 Full Adder 144, 145, 146, 163, 164, 169, 170, 171, 172, 174, 192, 193  
 Full subtractor 147, 148, 149  
 Fundamental mode asynchronous sequential circuit 334, 336
- G**
- Gate Definition 78  
 Generic Array Logic (GAL) 204  
 Glitches (spikes) 179, 180, 185  
 Gray code 43, 44, 45  
 Gray to Binary conversion 45
- H**
- Half Subtractor 146, 147, 148, 149  
 Half-adder 143  
 Hamming Code 49, 50, 51, 53  
 Hamming distance 48  
 Hardware Algorithm 362, 363  
 Hazard 179, 180, 181, 182, 183, 184, 185, 194  
 Hexadecimal 12, 15, 16, 20, 21, 22, 35, 57, 58, 62  
 High Threshold Logic (HTL) 422  
 History sensitive circuits 11  
 Huntington Postulates 65, 66, 67
- I**
- I/O blocks 207  
 IC 74193 283, 284, 286, 329  
 IC 7490A 307, 308, 309, 310, 311, 312, 313, 329  
 IC 7492A 313  
 Integrated Circuit 1, 10  
 Integrated Circuit or IC 382
- Interconnect 207, 209, 211  
 Intersection Law 67  
 Intrinsic semiconductor 383, 384  
 Inverter 65, 66, 67, 71, 82, 83, 84, 85, 86, 92, 94, 98, 99, 100, 104, 105  
 Involution Law 67  
 Ionization process 384
- J**
- JK flip-flops 232, 234  
 Johnson Counter 318, 319, 321, 322
- K**
- Karnaugh 113, 123, 125, 137, 138, 139  
 Karnaugh MAP (K-Map) 113  
 Knee voltage 389
- L**
- Latch 215, 216, 217, 218, 219, 220, 221, 222, 223, 228, 234, 261, 262, 264  
 Leakage current 387, 389, 441  
 Level translators 433, 436  
 Loading of DTL Gate 418, 419  
 Loading Inputs 425  
 Logic 63, 64, 65, 67, 68, 71, 72, 77, 78, 79, 80, 82, 83, 84, 85, 86, 88, 90, 91, 92, 94, 95, 96, 97, 98, 100, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111  
 Logic blocks 207, 208  
 Logic gates 8, 10, 77, 92, 94, 95, 96  
 Logical adjacency 68  
 Look-ahead Carry Generator 171, 172, 173  
 Lookup tables (LUT) 208  
 LSI 10, 224  
 LSI Circuits 155
- M**
- Magnitude Comparator 177, 178, 179  
 M-ary signal 5  
 Mask programmed ROMs 197  
 Memory elements 213, 214, 251  
 Merger diagram 349, 352, 357, 358  
 Metastability 332

## 522 *Switching Theory*

Minimum distance 48  
Minterm and Maxterm 73  
Mod-3 Counter 288, 289, 290, 295, 296  
Mod-5 counter 289, 290, 296, 297, 301, 302, 303, 307, 311, 329, 368  
MODES OF OPERATION 334  
Modulo 272, 286, 289, 295, 307, 311, 312 328  
Modulo counter 286, 307, 328  
Modulus counter 296, 287, 295  
MOS 382, 388, 389, 392, 393, 398, 399, 400, 401, 402, 403, 407, 422, 430, 438, 439, 440, 441, 442, 443, 444, 446, 447, 448, 449, 450, 451  
MOS Logic 438, 450  
MSI 10, 224  
MSI Circuits 148  
MSI Counter 307, 313  
Multi Output Minimization 129  
MUX 156, 157, 158, 159, 160, 161, 162, 163, 165, 166, 195

## N

Nand and NOR Implementation 97  
NAND Gate 84, 85, 86, 90, 92, 95, 97, 98, 99, 104, 105, 106, 108, 109  
NAND Gate Latch 218, 219  
Next State 213, 221, 222, 224, 228, 234, 238, 241, 243, 244, 245, 248, 249, 250, 251, 253, 254, 255, 256, 257, 258, 259, 261  
Nibble 12, 15, 16, 17, 18, 37, 38, 39  
NMOS 399, 400, 402, 403, 438, 439, 440, 441, 442, 449, 451  
NMOS as Load 438  
NMOS Gates 438  
Noise immunity 406, 440, 442  
Noise Margin 406, 407, 411, 413, 414, 415, 418, 420, 422, 426, 427, 436, 441, 449  
NON Saturated LOGIC FAMILY 430, 431, 433  
Non Weighted Codes 43  
Noncritical race 345  
NOR gate 84, 86, 87, 88, 90, 91, 92, 93, 95, 96, 97, 99, 100, 104, 105, 108, 109  
NOR LATCH 216, 218

## O

Octal Number 18, 22, 58, 62  
Octal to Binary 19  
ON time 391, 398, 399 449  
Open collector output 425, 445, 449  
Open emitter output 434, 449  
OR Gate 71, 80, 81, 82, 84, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 104, 105, 106, 108, 109  
Output characteristic of transistor 395

## P

Parallel Adder 169, 170, 171, 172, 173, 174, 176  
Parallel Adder/Subtractor 173, 174  
Parity bit 46, 48, 52  
Parity Generators and Checkers 146  
Passive Pullup 414, 420, 423, 424, 425, 449, 450  
Path Sensitizing 185, 189, 192  
Permanent faults 185  
PMOS 402, 403, 441, 442, 449  
PN-junction 385  
Positive and Negative Logic 77  
Postulates 65, 66, 67  
Power Dissipation 402, 403, 404, 405, 406, 417, 418, 421, 424, 426, 427, 428, 430, 431, 436, 439, 440, 441, 442, 443, 447, 450, 451  
Present State 213, 214, 221, 222, 224, 229, 234, 238, 241, 243, 244, 245, 248, 249, 255, 256, 257, 258, 259, 261  
Preset 266, 279, 280, 283, 285, 286, 287, 307, 316, 317, 324, 329  
Previous state next state 228  
Prime and Essential Implicants 123  
Primitive flow 340, 349, 351, 352, 353, 357, 358  
Primitive flow table 340, 349, 351, 352, 357, 358  
Priority Encoder 168  
Product of Sums (POS) 72  
Product Term 71, 72, 75  
Product-of-maxterms 74  
Programmable Array Logic (PAL) 202



- Programmable logic devices (PLDs) 196  
 Programmable Read-Only Memory (PROM) 199  
 Propagation delay 404, 405, 407, 414, 420, 422, 423, 424, 425, 427, 430, 431, 438, 442, 449, 450  
 Pull down resistor 424  
 Pullup Resistor 407, 408, 420, 424, 446, 447  
 Pullup transistor 428, 442, 447, 450  
 Pulse mode asynchronous sequential circuit 334, 335
- Q**
- Quine-McCluskey (Tabular) Method 130
- R**
- r's Complement 35, 36  
 Race 332, 345, 346, 347, 349, 353, 354, 359, 361  
 Race around 230, 231, 233, 234, 235  
 Race-around condition 230  
 Read Only Memory (ROM) 196  
 Redundancy 45  
 Redundant Bits 49, 52, 53  
 Reflective code 44  
 Register 364, 365, 366, 371, 372, 373, 374, 375  
 Reset 266, 268, 269, 273, 274, 277, 279, 280, 281, 282, 283, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 300, 307, 309, 312, 315, 316, 317, 318, 319, 320, 324, 329  
 Reset State 216, 217, 218, 221, 225, 229, 230, 248  
 Resistor Transistor Logic (RTL) 410  
 Response time 391  
 Reverse biasing 386, 424  
 Reversible shift register 271  
 Ring counter 271, 316, 317, 318, 319, 322, 323, 330  
 Ripple counter 272, 273, 274, 275, 280, 293  
 Rise time 398  
 RS Flip-Flop 216, 228, 264
- Saturation current 389, 393, 396, 421  
 Saturation region 395, 396, 397, 399, 431  
 Scale Current 388  
 Schottky Diode 391, 399, 449  
 Schottky Transistor 399, 430, 449  
 Schottky TTL 403, 430, 450  
 Self complementing 43  
 Self complementing codes 43  
 Sensitized 189, 190, 191  
 Sequence Detector 369  
 Sequence Detector 369, 370  
 Sequence generators 322, 323, 325, 327  
 Sequential Circuits 213, 214, 219, 238, 241, 246, 247, 265, 271, 272  
 Sequential Logic 10, 11  
 Serial Adder 170  
 Serial and Parallel Adders 169  
 Set state 216, 217, 218, 221, 225, 229, 230, 248  
 Shift Register 265, 266, 267, 268, 270, 271, 316, 320, 321, 323, 326, 329, 330  
 Signed Complement Representation 31  
 Signed Magnitude Representation 31  
 Simple PLDs (SPLDs) 202  
 Six variable K-Map 127  
 Specifications of Standard TTL 425, 431  
 Speed power product 404  
 Spikes 179, 180, 185  
 SPLDs 202, 204, 206, 207, 209, 210  
 SSI 1, 2, 4, 6, 8, 9, 10  
 SR latch 219, 220, 223, 228, 261  
 stable state 214, 216, 217, 222  
 Standard Forms 75, 77  
 State 213, 214, 215, 216, 217, 218, 220, 221, 222, 223, 224, 225, 227, 228, 229, 230, 231, 232, 233, 234, 235, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264  
 State box 362, 363, 364, 365, 368, 370, 371  
 State Diagram 366, 369, 370, 381  
 State variables 337, 346, 348, 354, 356  
 Static hazard 181, 182, 184, 194  
 Static Power Dissipation 441
- S**



## 524 *Switching Theory*

Storage time 390, 391, 399, 430, 449

Subtractors 146, 148

Sum of minterms 74, 75, 104, 111

Sum of Products (SOP) 72, 199

Sum Term 72, 75, 76

Switching Circuits 8, 9, 10

Switching time 391, 424

Synchronous 265, 266, 271, 272, 273, 280, 281, 282, 283, 288, 291, 293, 295, 296, 298, 299, 300, 301, 303, 307, 328, 329, 330, 331, 332, 333, 334, 335, 336, 339, 341, 345, 346, 347, 349, 351, 353, 354, 356, 360, 361

Synchronous counter 272, 273, 280, 282, 288, 291, 293, 299, 300, 301, 307, 329, 330

Synchronous Sequential Circuits 214

### T

Tautology Law 67

Temporary faults 185

Three state buffer 445, 446

Three State Logic (TSL) 444

Threshold voltage 401, 441, 449

Toggle Switch 214, 229, 230, 231, 233

Toggling 6

Totem pole output 428

Transfer characteristics 395, 441, 442, 451

Transistor as a Switch 397

Transistor Transistor Logic (TTL) 423

Transition Table 338, 339, 340, 341, 342, 343, 345, 348, 349, 359, 360, 391, 449

Truth table 8, 9, 64, 65, 66, 74, 80, 83, 85, 86, 87, 92, 216, 217, 218, 219, 220, 221, 222, 224, 227, 228, 229, 232, 233, 238, 247, 253, 259, 264

TSL Inverter 424, 443, 445

TTL Series 430

TTL to CMOS Interface 446

Types of Hazards 181

### U

Union Law 67

Unipolar logic families 403

Unipolar transistors 399

Unit Distance code 44

Universal gates 84, 90, 97

Unused Codes 42

Up-counter 272, 273, 275, 276, 277, 278, 282, 298, 328, 329

Up-Down Counters 278

### V

VLSI 10

### W

Wired Logic 421, 425, 430, 437, 450

Words 12, 15, 17, 18, 40, 41, 42, 46, 48, 53, 54

### X

Xilinx 208

XS-3 to BCD code converter 151, 153