OXFORD
HIGHER EDUCATION

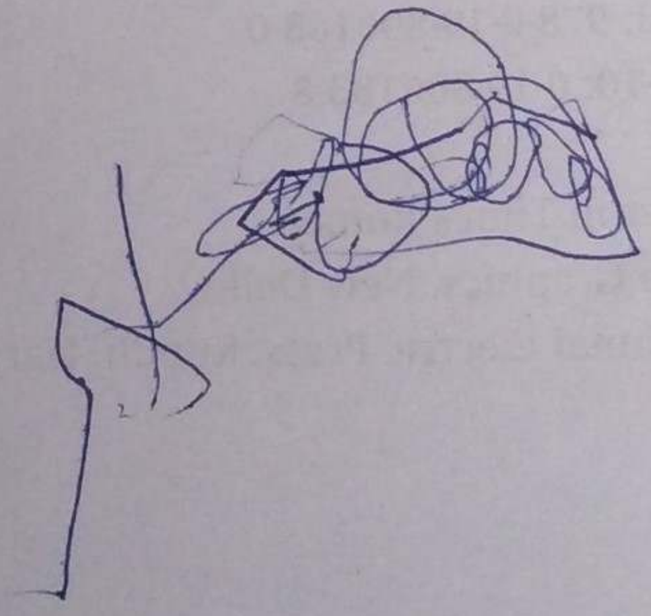
DIGITAL ELECTRONICS

G.K. KHARATE

DIGITAL ELECTRONICS

SPECIMEN COPY
NOT FOR SALE

G. K. KHARATE
Principal
Matoshri College of Engineering
and Research Centre
Nashik



OXFORD
UNIVERSITY PRESS

OXFORD
UNIVERSITY PRESS

Oxford University Press is a department of the University of Oxford.
It furthers the University's objective of excellence in research, scholarship,
and education by publishing worldwide. Oxford is a registered trademark of
Oxford University Press in the UK and in certain other countries

Published in India by
Oxford University Press
YMCA Library Building, 1 Jai Singh Road, New Delhi 110001, India

© Oxford University Press 2010

The moral rights of the author have been asserted

First Edition published in 2010
Seventh impression 2013

All rights reserved. No part of this publication may be reproduced, stored in
a retrieval system, or transmitted, in any form or by any means, without the
prior permission in writing of Oxford University Press, or as expressly permitted
by law, by licence or under terms agreed with the appropriate reprographics
rights organization. Enquiries concerning reproduction outside the scope of the
above should be sent to the Rights Department, Oxford University Press, at the
address above

You must not circulate this work in any other form
and you must impose this same condition on any acquirer

ISBN-13: 978-0-19-806183-0
ISBN-10: 0-19-806183-8

Typeset in Times Roman
by Pee-Gee Graphics, New Delhi
Printed in India by Raj Kamal Electric Press, Kundli, Haryana

To
my beloved parents
Mrs Bhagirathi and Mr Kashiram Kharate

PREFACE

The tremendous power and usefulness of digital techniques and systems can be seen in a wide variety of applications in the areas of industrial machinery, computers, microprocessors, and household appliances, among others. The areas where digital electronics can be applied are increasing at a fast pace. Several critical parameters including the level of government support, quality of labour, availability of engineering skills, cost advantage, project management skills, entrepreneurial culture, and strong customer relationships have made India the preferred destination for a booming electronics industry. This has allowed the individuals involved in this field to showcase their talent and meet the current demands of the consumers and industries. All these factors have resulted in the growing interest among students and faculty members of this subject.

Developments in the integrated circuits (IC) technology have made it possible to fabricate complex digital circuits such as microprocessors, memories, complex programmable logic devices (CPLDs), and field programmable gate arrays (FPGAs). The emergence of various programmable logic devices has resulted in significant changes in the design methodologies of digital systems. Therefore, it is essential to have a strong foundation of the basic digital techniques for making effective use of automation in digital design.

About the Book

This book is suitable for a course in digital electronics and logic design in undergraduate engineering disciplines such as electrical, electronics, instrumentation, telecommunication, computer science, and information technology. The last four chapters of the book will also be useful for postgraduate engineering students pursuing courses in electronics and computer science as also to postgraduate students of physics specializing in electronics.

The book presents the basic theory of switching circuits and their applications. The availability of various digital functions in ICs has changed the teaching of digital electronics from the good old approach of using discrete devices to a new approach of using modern digital ICs. This book adopts the new approach. The text has been systematically organized and the presentation has been kept at a level appropriate for students with the basic knowledge of circuit theory and digital electronics.

Since the study of Boolean algebra helps in analysing and synthesizing switching circuits, a complete chapter is dedicated to Boolean algebra. Separate chapters on asynchronous and synchronous circuits and K-maps have also been included.

The text is interspersed with illustrations and solved examples for all topics in the chapters. Glossary of important terms and descriptive as well as objective questions are provided at the end of each chapter to enhance the understanding of the reader.

Content and Structure

This book has been divided into 10 chapters. A brief description of these chapters is as follows:

Chapter 1 introduces the fundamental concepts of logic families and characteristics of digital ICs that are used to compare the performance of logic families. The chapter begins with resistor-transistor logic (RTL), direct coupled transistor logic (DCTC), diode-transistor logic (DTL), modified diode-transistor logic, transistor-transistor logic (TTL), emitter coupled logic, integrated injection logic (I2L), NMOS, CMOS, and ends with the interfacing problems between ICs of the same logic families and between those of different logic families to obtain maximum benefits.

Chapter 2 includes the detailed study of different number systems, their conversion, binary arithmetic codes, error detecting codes, and error correcting codes.

Chapter 3 includes Boolean algebra, DeMorgan's theorems, simplification of Boolean expression by algebraic method, Karnaugh map method, and Quine-McCluskey method.

Chapter 4 deals with combinational logic design using MSI circuits, which is important for the design of digital systems considering the simplicity in design, cost, space, power requirement, speed, and other factors.

Chapter 5 introduces the basic building blocks of sequential circuits including flip-flop conversions, application of flip-flops, shift registers and their applications. This chapter also discusses the analysis and design of synchronous sequential circuits.

Chapter 6 discusses the analysis and design of asynchronous sequential circuits.

Chapter 7 discusses various ways of representing the control sections and data sections of hardware algorithms. The chapter includes the use of algorithmic state machine (ASM) charts, register transfer language (RTL), and covers the hardware description language, VHDL.

Chapter 8 covers programmable logic arrays (PLA), programmable array logic (PAL), generic array logic (GAL), complex programmable logic device (CPLD), field programmable gate arrays (FPGAs), Xilinx XC 9500 CPLD family, and application specific integrated circuits (ASICs).

Chapter 9 explains the most common A/D and D/A conversion techniques.

Chapter 10 presents the study of memory organization and operations, classification, characteristics, and types of memories—RAM, ROM, EPROM, EEPROM, SRAM, and DRAM.

Acknowledgements

I will be forever indebted to many people who helped me during this project. First and foremost, I am deeply indebted to my family—wife, Sangeeta and children, Puja and Pujan—from whom I received continuous encouragement and support. Their loving and caring attitude has been the driving force for this endeavour and no words of gratitude are enough. I also express my deep sense of gratitude to Dr (Mrs) V. H. Patil for her invaluable and thought-provoking suggestions, constant encouragement, and undying enthusiasm throughout this project. I wish to specially thank Prof. D. D. Dighe and Mr H. D. Deshpande who helped me with my research work for this project.

Special thanks to my parents, in-laws, sisters, and brother for their unconditional love and support. Thanks are also due to all my friends, relatives, and colleagues for their support. Finally, I thank the editorial team at Oxford University Press, Delhi, for their cooperation.

Dr G. K. Kharate

CONTENTS

Preface *v*

Chapter 1: LOGIC FAMILIES **1**

1.1 Introduction *1*
1.2 Logic Families *2*
1.3 Transistor as Switch *3*
1.4 Characteristics of Digital ICs *4*
1.5 Resistor-Transistor Logic (RTL) *8*
1.6 Direct Coupled Transistor Logic (DCTL) *9*
1.7 Diode-Transistor Logic (DTL) *11*
1.8 Modified Diode-Transistor Logic *12*
1.9 Transistor-Transistor Logic (TTL) *13*
1.10 TTL Parameters *23*
1.11 Commonly Used ICs of Standard TTL *26*
1.12 Improved TTL Series *27*
1.13 Comparison of TTL Families *29*
1.14 Emitter Coupled Logic *29*
1.15 Integrated Injection Logic (I2L) *32*
1.16 MOSFET Logic *34*
1.17 NMOS *34*
1.18 CMOS *37*
1.19 Comparison of CMOS and TTL Families *43*
1.20 Interfacing CMOS and TTL Devices *43*
1.21 Interfacing ECL and TTL Devices *45*

Chapter 2: NUMBER SYSTEM AND CODES **51**

2.1 Introduction *51*
2.2 Number Systems *51*
2.3 Interconversion of Numbers *53*
2.4 Signed Binary Number *68*
2.5 Floating Point Representation of Number *76*
2.6 Binary Arithmetic *77*
2.7 Complement Binary Arithmetic *81*

2.8 Arithmetic Overflow *91*
2.9 Codes *92*

Chapter 3: BOOLEAN ALGEBRA AND LOGIC GATES 129
3.1 Introduction *129*
3.2 Boolean Algebra *129*
3.3 Overview of Logic Circuit *137*
3.4 DeMorgan's Theorems *140*
3.5 Standard Representation for Logical Functions *141*
3.6 Minterm and Maxterm *142*
3.7 Simplification of Boolean Expression *145*
3.8 Simplification of Sum of Products *156*
3.9 Simplification of Product of Sums *167*
3.10 Don't-care Condition *178*
3.11 Five- and Six-variables K-map *181*
3.12 Quine-McCluskey Method *185*

Chapter 4: COMBINATIONAL LOGIC CIRCUITS 198
4.1 Introduction *198*
4.2 Design Procedure for Combinational Logic Circuit *198*
4.3 Adders *202*
4.4 Subtractor *211*
4.5 BCD Adder *219*
4.6 BCD Subtractor *220*
4.7 Arithmetic Logic Unit *226*
4.8 Comparator *229*
4.9 Parity Generator *233*
4.10 Parity Checker *235*
4.11 Parity Generator/Checker (IC74180) *239*
4.12 Multiplexer *241*
4.13 Demultiplexer *257*
4.14 Code Converters *264*
4.15 PIN Diagrams of ICs *286*

Chapter 5: SEQUENTIAL LOGIC CIRCUITS 293
5.1 Introduction *293*
5.2 1-bit Memory Cell *294*
5.3 Clocked S-R Flip-Flop *297*
5.4 J-K Flip-Flop *299*
5.5 D Flip-Flop *303*
5.6 T Flip-Flop *303*
5.7 Edge-Triggered Flip-Flop *304*

5.8 Characteristics of Flip-Flop *306*
5.9 Flip-Flop Conversions *307*
5.10 Applications of Flip-Flops *323*
5.11 Registers and Shift Registers *327*
5.12 Applications of Shift Register *334*
5.13 Commonly used ICs as Shift Register *341*
5.14 Ripple Counter *343*
5.15 Synchronous Counter *355*
5.16 Flip-Flop Excitation Table *355*
5.17 Synchronous Counter Design *358*
5.18 UP/DOWN Counter *363*
5.19 Clocked Sequential Circuit *369*
5.20 Analysis of Clocked Sequential Circuit *371*
5.21 Design of Clocked Sequential Circuit *375*
5.22 Lockout Condition *398*
5.23 Sequence Generator *401*
5.24 Sequence Detector *406*

Chapter 6: ASYNCHRONOUS SEQUENTIAL CIRCUITS 431
6.1 Introduction *431*
6.2 Fundamental Mode Asynchronous Sequential Circuits *432*
6.3 Pulse Mode Asynchronous Sequential Circuits *440*
6.4 Incompletely Specified State Machines *447*
6.5 Problems in Asynchronous Circuits *449*
6.6 Design of Hazard-free Switching Circuits *452*

Chapter 7: ALGORITHMIC STATE MACHINES 456
7.1 Introduction *456*
7.2 Algorithmic State Machines *457*
7.3 Realization of ASM Charts *464*
7.4 Examples on ASM Charts *468*
7.5 Register Transfer Language *478*
7.6 RTL Notations *479*
7.7 Data Unit Construction from an RTL Description *486*
7.8 VHDL *491*

Chapter 8: PROGRAMMABLE LOGIC DEVICES 529
8.1 Introduction *529*
8.2 Programmable Logic Array *529*
8.3 Programmable Array Logic *539*
8.4 Generic Array Logic Devices *549*
8.5 Classification of PLDs *550*

8.6 Complex Programmable Logic Devices 551
8.7 Field Programmable Gate Array 555
8.8 Application Specific Integrated Circuits (ASICs) 561

Chapter 9: A/D AND D/A CONVERTERS 584
9.1 Introduction 584
9.2 Basic Principle of DAC 585
9.3 Basic Principle of ADC 596
9.4 DAC and ADC ICs 607
9.5 ADC 0809 (8-bit ADC) 608
9.6 ADC-7109 (12-bit binary ADC) 611
9.7 DAC 0808 (8-bit DAC) 626

Chapter 10: SEMICONDUCTOR MEMORY DEVICES 630
10.1 Introduction 630
10.2 Memory Organization 630
10.3 Functional Diagram of Memory 631
10.4 Memory Operations 632
10.5 Expanding Memory Size 635
10.6 Characteristics of Memory Devices 641
10.7 Classification of Semiconductor Memories 642
10.8 Read and Write Memory 644
10.9 Read Only Memory 652

Index

Chapter 1

LOGIC FAMILIES

Chapter Outline

- Types of logic families
- Switching characteristics of transistor
- Characteristics of digital ICs
- RTL, DCTL, TTL, ECL logic families
- CMOS NAND, NOR and Inverter
- Comparison of TTL and CMOS
- Interfacing of TTL to CMOS and CMOS to TTL
- Interfacing of TTL to ECL and ECL to TTL
- 74XX series data sheet

1.1 INTRODUCTION

Digital logic is concerned with the interconnection among digital components and modules. The best known example of a digital system is the general purpose digital computer. Most of the digital circuits are constructed on a single chip, which are referred to as *integrated circuits* (IC). Integrated circuits contain a large number of interconnected digital circuits within a single small package.

Small scale integration (SSI) and medium scale integration (MSI) devices provide digital functions and large scale integration (LSI) devices provide complete computer modules. It is very important for the logic designer to be familiar with the various digital components encountered in integrated circuit form. It is constructed by basic electronic components such as diodes, transistors, MOSFETs, registers, and capacitors. The complex digital functions can be realized using these basic electronic components in a variety of forms and each form is referred as a *logic family*.

Now-a-days digital integrated circuits are most commonly used in modern digital systems. ICs are popular due to their enormous advantages, as listed below.

- Small in size
- Low cost
- Less power consumption

8.6 Complex Programmable Logic Devices *551*
8.7 Field Programmable Gate Array *555*
8.8 Application Specific Integrated Circuits (ASICs) *561*

Chapter 9: A/D AND D/A CONVERTERS **584**
9.1 Introduction *584*
9.2 Basic Principle of DAC *585*
9.3 Basic Principle of ADC *596*
9.4 DAC and ADC ICs *607*
9.5 ADC 0809 (8-bit ADC) *608*
9.6 ADC-7109 (12-bit binary ADC) *611*
9.7 DAC 0808 (8-bit DAC) *626*

Chapter 10: SEMICONDUCTOR MEMORY DEVICES **630**
10.1 Introduction *630*
10.2 Memory Organization *630*
10.3 Functional Diagram of Memory *631*
10.4 Memory Operations *632*
10.5 Expanding Memory Size *635*
10.6 Characteristics of Memory Devices *641*
10.7 Classification of Semiconductor Memories *642*
10.8 Read and Write Memory *644*
10.9 Read Only Memory *652*

Index *664*

Chapter 1

LOGIC FAMILIES

Chapter Outline

- Types of logic families
- Switching characteristics of transistor
- Characteristics of digital ICs
- RTL, DCTL, TTL, ECL logic families
- CMOS NAND, NOR and Inverter
- Comparison of TTL and CMOS
- Interfacing of TTL to CMOS and CMOS to TTL
- Interfacing of TTL to ECL and ECL to TTL
- 74XX series data sheet

1.1 INTRODUCTION

Digital logic is concerned with the interconnection among digital components and modules. The best known example of a digital system is the general purpose digital computer. Most of the digital circuits are constructed on a single chip, which are referred to as *integrated circuits* (IC). Integrated circuits contain a large number of interconnected digital circuits within a single small package.

Small scale integration (SSI) and medium scale integration (MSI) devices provide digital functions and large scale integration (LSI) devices provide complete computer modules. It is very important for the logic designer to be familiar with the various digital components encountered in integrated circuit form. It is constructed by basic electronic components such as diodes, transistors, MOSFETs, registers, and capacitors. The complex digital functions can be realized using these basic electronic components in a variety of forms and each form is referred as a *logic family*.

Now-a-days digital integrated circuits are most commonly used in modern digital systems. ICs are popular due to their enormous advantages, as listed below.

- Small in size
- Low cost
- Less power consumption

- High noise margin
- High reliability
- High speed

This chapter includes the study of different logic families, interfacing of logic families, and data sheets of commonly used series 74XX.

1.2 LOGIC FAMILIES

The set of compatible ICs with the same logic levels and same supply voltages have been fabricated to perform the various logic functions known as logic family. Based on the fabrication technology, logic families are classified into two types:

- Bipolar logic family
- Unipolar logic family

Unipolar logic family

In unipolar logic families, unipolar devices are the key element. MOSFET (Metal Oxide Semiconductor Field Effect Transistor) is a unipolar device, in which the current flows because of only one type of charge carriers (that is, either electrons or holes). The examples of unipolar families include PMOS, NMOS, and CMOS.

Bipolar logic family

Transistors and diodes are bipolar devices, in which the current flows because of both the charge carriers (electrons and holes). In bipolar logic families, transistors and diodes are used as key elements. On the basis of operations of transistors in ICs, bipolar logic families are further classified as:

- Saturated bipolar logic families
- Unsaturated bipolar logic families

In saturated bipolar logic families, transistors operate in saturation region. The speed of saturated bipolar logic family is low, reasons of which would be discussed in forthcoming topics. Examples of saturated bipolar logic families are:

- Resistor-transistor logic
- Direct coupled transistor logic
- Integrated injection logic
- Diode-transistor logic
- High-threshold logic
- Transistor-transistor logic

In non-saturated bipolar logic families, transistors operate in active region. The speed of non-saturated bipolar logic families is high as compared to saturated logic families. Examples of unsaturated bipolar logic families are:

- Schottky transistor-transistor logic
- Emitter-coupled logic

The transistor is one of the key elements used in logic families. One of the important applications of the transistor is the *switch*.

1.3 TRANSISTOR AS SWITCH

Transistor is one of the basic elements of logic families. It operates as a switch. In switching circuits, transistors operate in cut-off or saturation region. The cut-off condition is referred to as *switch OFF* and saturation is referred to as *switch ON*.

In the cut-off region, both emitter and collector junctions are in reverse-bias condition and only reverse current flows in the transistor, which is negligible.

In the saturation region, both emitter and collector junctions are in forward bias condition. When transistor is operated in the saturation region, voltage across the emitter junction is V_{BEsat} and voltage across emitter and collector terminals is V_{CEsat}. The value of $V_{BEsat} = 0.8$ V for silicon transistor and 0.3 V for germanium transistor. The value of $V_{CEsat} = 0.2$ for silicon transistor and 0.1 V for germanium transistor. The condition to operate the transistor in saturation is that the base current should be greater than the collector current by β, that is, $[I_B > I_C/\beta]$.

The circuit of a transistor working as a switch is shown in Fig. 1.1.

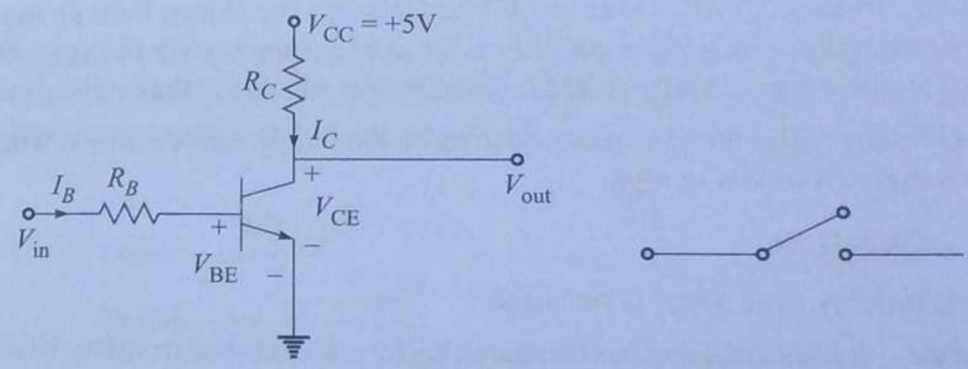


Fig. 1.1 Transistor as a switch

When input V_{in} applied to the transistor is LOW (0 V), the emitter junction is reverse biased, there is no current flowing through the base terminal and the current flowing through the collector terminal is reverse saturation current, which is negligible.

By applying KVL to the output loop,

$$V_{CC} - I_C R_C - V_{out} = 0$$

$$V_{out} = V_{CC} - I_C R_C \tag{1.1}$$

In the cut-off region, collector current $I_C = 0$

$$V_{out} = V_{CC} \tag{1.2}$$

When the transistor is operating in cut-off, the output is equal to V_{CC} and it is referred to as HIGH (logic 1).

When input V_{in} that is applied to the transistor is HIGH (+5 V), the emitter junction is forward biased, current flowing through the base terminal is sufficient and the transistor operates in saturation region. Current flows through the collector terminal and there is a considerable voltage drop across the collector resistor. And output voltage is

$$V_{out} = V_{CEsat} \tag{1.3}$$

When the transistor is operating in saturation, the output is equal to V_{CEsat} and it is referred to as LOW (logic 0). Transistor is either ON or OFF and it is controlled by the input voltage.

The response of a transistor switch for a square wave input is shown in Fig. 1.2.

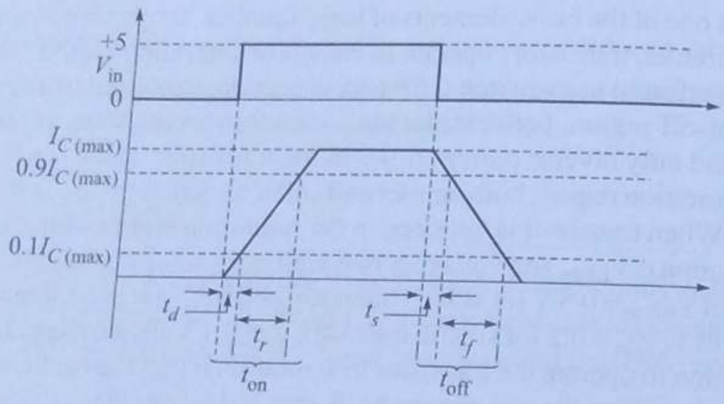


Fig. 1.2 Response of transistor switch for square wave input

When a transistor switches from OFF to ON state, the charge built up requires time to reach the steady-state condition. Similarly when it switches from ON to OFF state, the excess charge stored must be removed which takes some time.

Turn-ON time It is the time required to reach the steady state condition when the input changes from low to high.

$$t_{ON} = t_d + t_r \tag{1.4}$$

where t_d is delay time and t_r is rise time.

Delay time It is the time required for the collector current to rise from 0 to 10 percent of the maximum value of the collector current.

Rise time It is the time required for the collector current to rise from 10 to 90 percent of the maximum value of the collector current.

Turn-OFF time It is the time required to remove the excess charge stored near the junction, when the input changes from high to low.

$$t_{OFF} = t_f + t_s \tag{1.5}$$

where t_f is fall time and t_s is storage time.

Storage time It is the time required to drop the collector current to 90 percent of the maximum value of the collector current when the input changes from high to low.

Fall time It is the time required to drop the collector current from 90 to 10 percent of the maximum value of the collector current.

1.4 CHARACTERISTICS OF DIGITAL ICs

There are various logic families and the selection of a family for a particular application is based on its characteristics. Real time applications demand high speed logic families and hence it is necessary to study the characteristics of digital ICs. Following are the parameters used to compare the performance of digital ICs:

- Speed of operation
- Power dissipation

- Figure of merit
- Fan-out
- Fan-in
- Current and voltage parameters
- Noise Immunity
- Power supply requirements
- Operating temperature

1.4.1 Speed of Operation

The speed of operation of digital ICs should be high. It is specified in terms of *propagation delay time*. It is the average of the propagation delay times from high to low state and low to high state.

$$t_p = \frac{t_{PHL} + t_{PLH}}{2} \tag{1.6}$$

where t_{PHL} is the delay time measured, when output changes from high to low state, and t_{PLH} is the delay time measured, when output changes from low to high state. The input and output voltage waveforms of a logic gate are shown in Fig. 1.3.

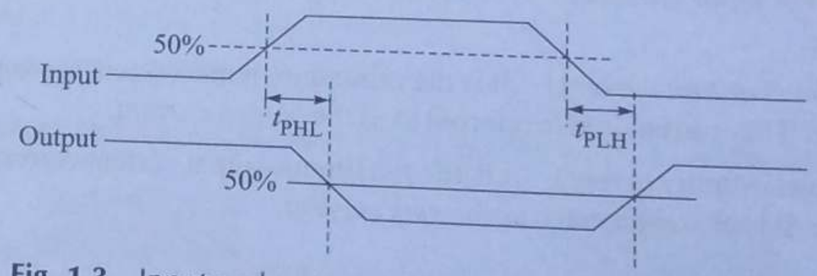


Fig. 1.3 Input and output voltage waveforms of logic gate

The delay times are measured between 50 percent voltage levels of input and output waveforms.

The propagation delay between input and output should be as minimum as possible so that the operating speed of IC remains high.

1.4.2 Power Dissipation

Every electronic circuit requires a certain amount of electric power for its operation. When the power is supplied by an external source, some of it is dissipated in electronic circuits. It is wastage of power across the circuit components and devices. Requirement of power is less, if the dissipation of power is less. Hence power dissipation should be as minimum as possible. Power dissipation of an IC is expressed in terms of milli Watt (mW).

1.4.3 Figure of Merit

The power dissipation of logic families should be minimum to reduce power requirements. But it is important to note that in logic families, if the power dissipation is reduced, the speed of operation gets reduced. Hence the *figure of merit* is a parameter considered for comparison instead of using the speed of operati

power dissipation. Figure of merit is a product of propagation delay and power dissipation. It is measured in terms of Pico-Joules (ns × mW = pJ).

1.4.4 Current and Voltage Parameters

Current and voltage parameters define the minimum and maximum limit of current and voltage for input and output of a logic family.

V_{IH} (High level input voltage) It is the minimum input voltage corresponding to logic 1 state.

V_{IL} (Low level input voltage) It is the maximum input voltage corresponding to logic 0 state.

V_{OH} (High level output voltage) It is the minimum output voltage corresponding to logic 1 state.

V_{OL} (Low level output voltage) It is the maximum output voltage corresponding to logic 0 state.

I_{IH} (High level input current) It is the minimum input current corresponding to logic 1 state.

I_{IL} (Low level input current) It is the maximum input current corresponding to logic 0 state.

I_{OH} (High level output current) It is the minimum output current corresponding to logic 1 state. This current is also referred to as the *source current*.

I_{OL} (Low level output current) It is the maximum output current corresponding to logic 0 state. It is also referred to as the *sink current*.

1.4.5 Fan-Out

Fan-out is the capability of a logic gate to drive the maximum number of similar gates. High fan-out is advantageous, because it reduces the need of additional gates to drive more gates. Consider Fig.1.4.

In Fig. 1.4, the driver gate drives all the N gates (N is fan-out). If more than N gates are connected to a load of the driver gate, then the current supply by the driver gate is less than the current required to drive the gates, or the current sink by the driver gate is more than the rating of the driver gate which may damage the gate.

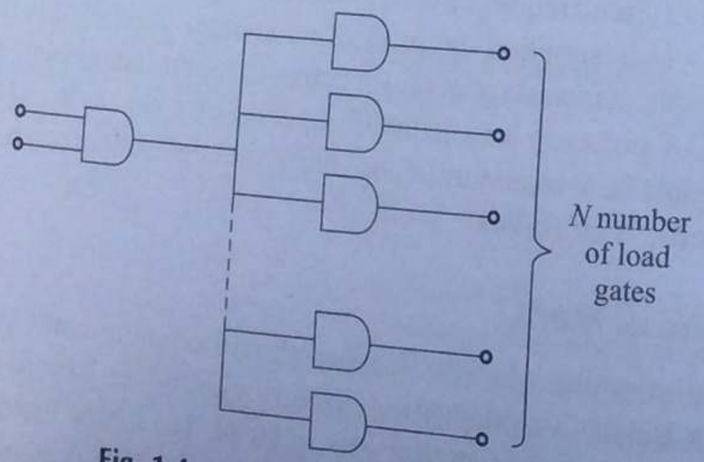


Fig. 1.4 AND gate driving N gates

The fan-out of a logic family can be calculated as

$$\text{Fan-out} = \text{minimum of} \left\{ \frac{I_{OH}}{I_{IH}}, \frac{I_{OL}}{I_{IL}} \right\} \tag{1.7}$$

1.4.6 Fan-In

Fan-in is the number of inputs to a gate. For a two-inputs gate, fan-in is two; and for a four-inputs gate, fan-in is four.

1.4.7 Noise Immunity

Unwanted signals are known as *noise*. The stray electric and magnetic fields may induce some noise at an input of digital circuit. Because of noise, the input voltage may drop below V_{IH} or may be raise above V_{IL}, which results in undesired operations. The circuit should have the ability to tolerate the noise signal. The noise immunity of digital circuit is defined as the ability of a digital circuit to tolerate the noise signal. A quantitative measure of noise immunity is known as *noise margin*. Logic 1 state noise margin and Logic 0 state noise margin can be calculated as:

$$\text{Logic 1 state noise margin } \Delta 1 = V_{OH} - V_{IH} \tag{1.8}$$

$$\text{Logic 0 state noise margin } \Delta 0 = V_{IL} - V_{OL} \tag{1.9}$$

It is important that for logic families, $V_{OH} > V_{IH}$ and $V_{IL} > V_{OL}$ as shown in Fig. 1.5.

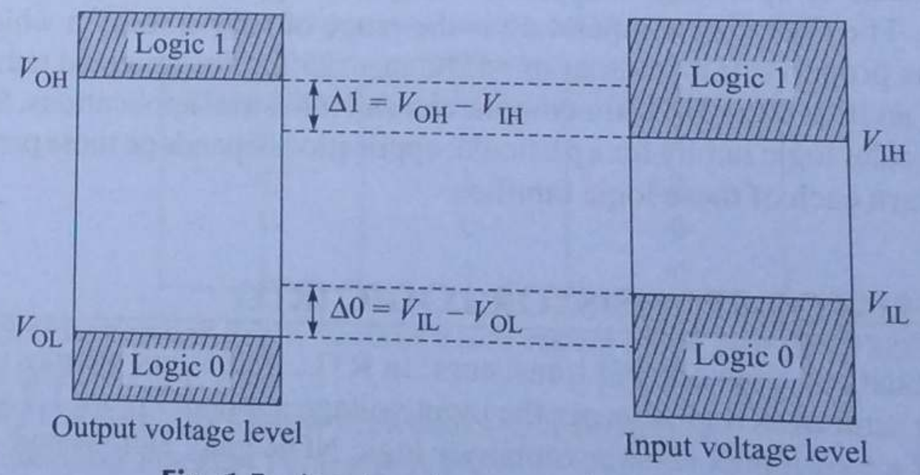


Fig. 1.5 Input and output voltage levels

Let us see how the noise leads to an undesired output. Take the example of a TTL AND gate. A detailed study of TTL AND gate is covered in Section 1.9. The TTL gate has $V_{OH} = 2.4$ V, $V_{OL} = 0.4$ V, $V_{IH} = 2$ V, and $V_{IL} = 0.8$ V. The noise introduced in the signal (V_{noise}) is shown in Fig. 1.6.

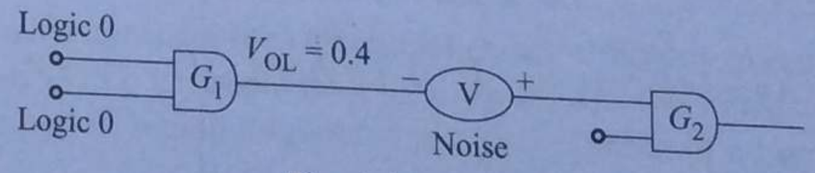


Fig. 1.6 Noise

Let the inputs of gate$_1$ be such that the output of gate$_1$ is in logic 0 state (V_{OL} = 0.4 V). This output is given as an input to gate$_2$. Because of the noise present, the actual input given to gate$_2$ is:

$$V_{i2} = V_{OL} + V_{noise} \quad (1.10)$$

$$V_{i2} = 0.4 + V_{noise}$$

$$V_{noise} = V_{i2} - 0.4$$

The maximum low level input of the gate is 0.8 V. When the noise introduced in the signal is greater than (0.8 – 0.4) = 0.4 V, then the input to gate$_2$ will be greater than 0.8 V, which transits to an invalid input level and the output will be unpredictable.

Similarly for high state output of gate$_1$

$$V_{i2} = 2.4 - V_{noise}$$

$$V_{noise} = 2.4 - V_{i2}$$

The minimum high level input of the gate is 2 V. When the noise signal is greater than (2.4 – 2.0) = 0.4 V, the input of gate$_2$ will be less than 2 V, which transits to an invalid input state and the output will be unpredictable.

1.4.8 Power Supply Requirements

Every electronic circuit requires supply voltage to operate. The required supply voltage and power by the IC should be as less as possible.

1.4.9 Operating Temperature

On the basis of operating temperature range, the application of the ICs will be decided. The operating temperature is the range of temperature in which an IC functions properly. It is in order of –55°C to +125°C. The accepted temperature range of an IC is 0 to +70°C for commercial and industrial applications. Selection of a particular logic family for a particular application depends on these parameters. Let us learn each of these logic families.

1.5 RESISTOR-TRANSISTOR LOGIC (RTL)

RTL consists of resistors and transistors. In RTL, transistors operate in cut-off region or saturation region as per the input voltage applied. Figure 1.7 shows the circuit of a two-inputs resistor-transistor logic NOR gate. Here A and B are the inputs of the gate and Y is the output.

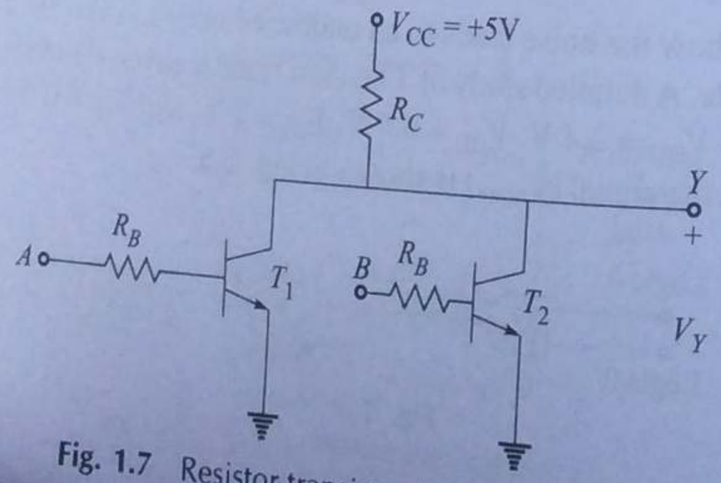


Fig. 1.7 Resistor-transis

Operation

- When the transistor operates in saturation region, maximum current flows through resistor R_C. The output voltage $V_Y = V_{CEsat}$ (V_{CEsat} = 0.2 V for silicon and 0.1 V for germanium); it is logic 0 level voltage. When the transistor operates in cut-off, no current flows through resistor R_C and the output voltage $V_Y = V_{CC}$ = +5 V; it is logic 1 level voltage.
- When both the inputs are in logic 0, transistors T_1 and T_2 operate in cut-off, and the output is $+V_{CC}$, i.e. +5 V (logic 1).
- When any one of the inputs is at logic 1 level, the corresponding transistor operates in saturation, and the output is V_Y = 0.2 V (logic 0).
- When both the inputs are at logic 1 level, both the transistors operate in saturation and the output is V_Y = 0.2 V (logic 0). The operation of circuit is summarized in Table 1.1(a).

Table 1.1(a) Operation of RTL NOR gate (Fig. 1.7)

V_A	V_B	*Transistor* T_1	*Transistor* T_2	V_Y
Logic 0	Logic 0	Cut-off	Cut-off	Logic 1
Logic 0	Logic 1	Cut-off	Saturation	Logic 0
Logic 1	Logic 0	Saturation	Cut-off	Logic 0
Logic 1	Logic 1	Saturation	Saturation	Logic 0

In terms of 0 and 1, the above table can be written as in Table 1.1(b).

Table 1.1(b) Operation of RTL NOR gate (Fig. 1.7)

V_A	V_B	V_Y
0	0	1
0	1	0
1	0	0
1	1	0

The circuit shown in Fig.1.7 acts as a two-inputs NOR gate and Table 1.1(b) is the truth table of NOR gate.

The RTL suffers from a few drawbacks as listed below:

1. Low noise margin (Typically 0.1 V)
2. Fan-out is poor (Typically 5)
3. Propagation delay is high and the speed of operation is low (Typically 12 ns)
4. High power dissipation (Typically 12 mW)

1.6 DIRECT COUPLED TRANSISTOR LOGIC (DCTL)

In direct coupled transistor logic, the input signal is directly given to the base of the transistor. DCTL is simple than RTL. In DCTL, the transistor operates in saturation or cut-off region. Figure 1.8 shows the circuit of a two-inputs DCTL NOR gate.

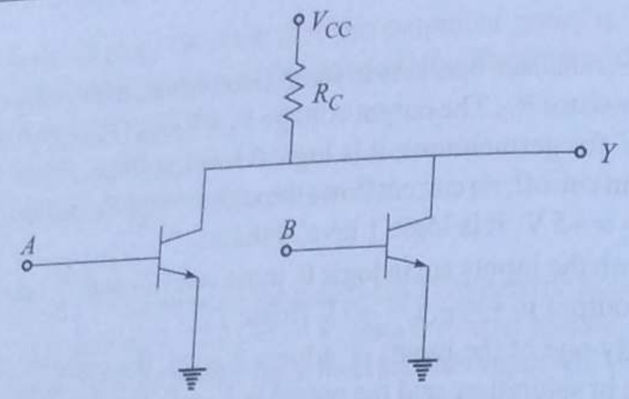


Fig. 1.8 Two-inputs DCTL NOR gate

Operation

The operation of DCTL is same as the operation of RTL. When both the inputs are in logic 0, transistors operate in cut-off, and the output is logic 1. When any one of the inputs or both the inputs are in logic 1, the corresponding transistor or transistors operate in saturation and the output is logic 0.

Although DCTL is simpler than RTL, it is not popular because of the *current hogging problem*. Let us see what the current hogging problem is.

Current hogging problem

Figure 1.9 shows a DCTL gate driving a three-inputs NOR gate.

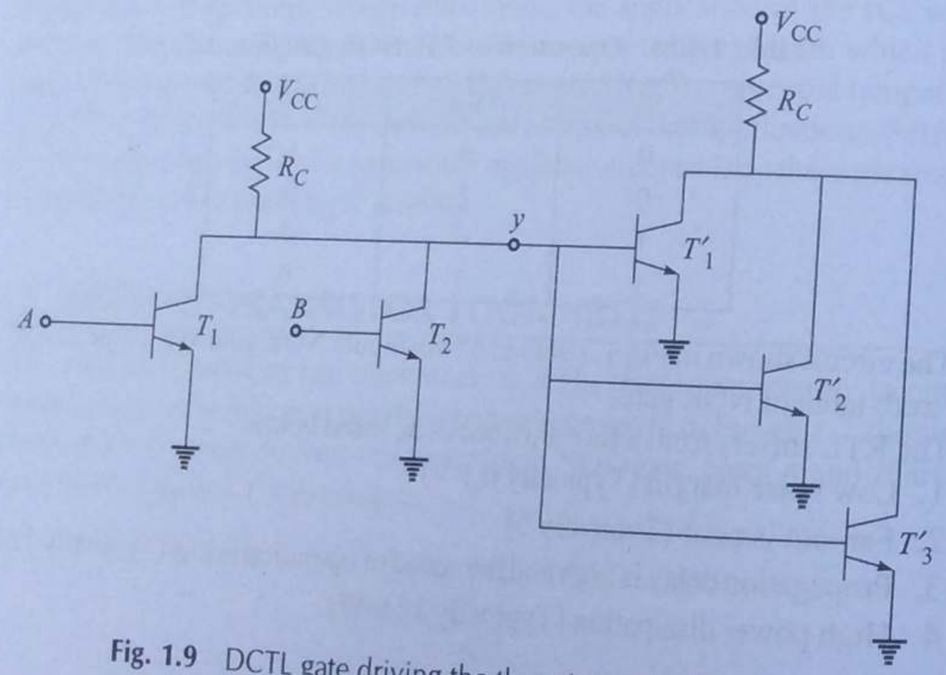


Fig. 1.9 DCTL gate driving the three-inputs NOR gate

Initially the input of transistor T_1 is logic 0, and the input of transistor T_2 is logic 1, T_1 operates in cut-off and T_2 operates in saturation, the output of driving gate is in logic 0 ($V_{CEsat} = 0.2$ V) which drives all the transistors in cut-off. When the input of T_2 changes from logic 1 to logic 0, T_2 operates in cut-off. The output of the driving gate starts to change from 0.2 V to V_{CC}.

Due to manufacturing tolerance, the base to emitter voltage of transistors T'_1, T'_2 and T'_3 are different. Assume V_{BEsat} of transistors T'_1, T'_2 and T'_3 are 0.8 V, 0.79 V and 0.78 V, respectively. When the voltage at the output of driving gate reaches 0.78 V, T'_3 goes in saturation and will not allow other transistors to enter in saturation. The whole current flows through the base of T'_3 and the transistor may damage. Once T'_3 damages, T'_2 goes in saturation and will not allow other transistors to enter in saturation. The whole current flows through the base of T'_2 and it may damage. Similarly, all the transistors of the driven circuit are damaged. It is known as the *current hogging problem*.

1.7 DIODE-TRANSISTOR LOGIC (DTL)

The circuit of a DTL consists of diodes and transistors. Figure 1.10 shows the circuit of a two-inputs diode-transistor logic NAND gate.

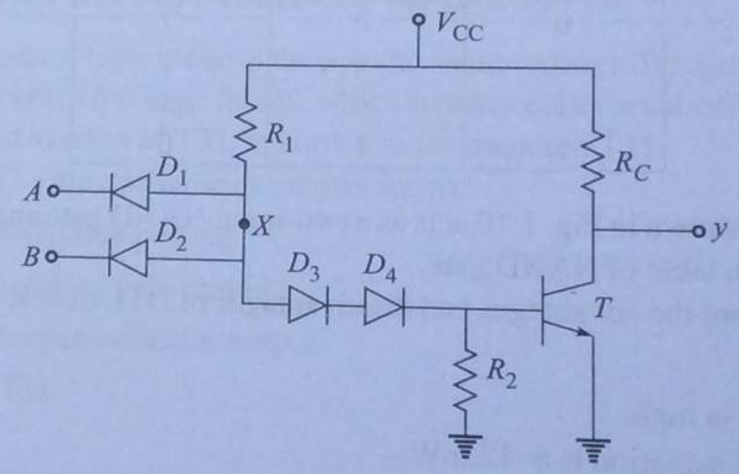


Fig. 1.10 Two-inputs DTL NAND gate

Operation

- When the transistor operates in saturation, the output voltage $V_{(0)} = V_{CEsat} = 0.2$ V; and when it operates in cut-off, the output voltage $V_{(1)} = V_{CC} = +5$ V.
- When both the inputs are in logic 0, $V_{(0)} = V_{CEsat} = 0.2$ V, the input diodes are forward biased, voltage at point x is $V_x = V_{(0)} + V_D = 0.2 + 0.7 = 0.9$ V which is not sufficient to drive the transistor in saturation, because the voltage desired at point x to drive the transistor in saturation should be $V_{BEsat} + V_{D4} + V_{D3} = 0.8 + 0.7 + 0.9 = 2.2$ V. The transistor operates in cut-off and the output voltage is in logic 1 state.
- When any one of the inputs is in logic 1, the corresponding diode is forward biased. Voltage at point x is $V_x = 0.2$ V $+ 0.7$ V $= 0.9$ V; the transistor operates in cut-off and the output voltage is in logic 1 state.
- When all the inputs are in logic 1 state, the diodes D_1 and D_2 are reverse biased. The resistances R_1 and R_2 are selected such that the transistor operates in saturation and the output voltage is in logic 0 state.

The operation of the circuit is summarized in Table 1.2(a).

Table 1.2(a) Operation of DTL NAND gate (Fig. 1.10)

Inputs		*Diodes*		*Transistor*	*output*
A	*B*	D_1	D_2	*T*	*Y*
Logic 0	Logic 0	Forward biased	Forward biased	Cut-off	Logic 1
Logic 0	Logic 1	Forward biased	Reverse biased	Cut-off	Logic 1
Logic 1	Logic 0	Reverse biased	Forward biased	Cut-off	Logic 1
Logic 1	Logic 1	Reverse biased	Reverse biased	Saturation	Logic 0

In terms of 0 and 1, Table 1.2(a) can be written as in Table 1.2(b).

Table 1.2(b) Operation of DTL NAND gate (Fig. 1.10)

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

The circuit shown in Fig. 1.10 acts as a two-input NAND gate and Table1.2(b) shows the truth table of NAND gate.

Following are the advantages and disadvantages of DTL over RTL.

Advantages

1. Fan-out is high.
2. Power dissipation is 8–12 mW.
3. Noise immunity is good.

Disadvantages

1. More elements are required.
2. Propagation delay is more (typically 30 ns) and hence the speed of operation is less.

1.8 MODIFIED DIODE-TRANSISTOR LOGIC

More fan-out gates are preferred for most of the applications. Fan-out of the gate is a function of source current. The fan-out of a logic gate is increased by increasing the current supply of the gate (source current). When the base current of a transistor is increased, then the source current in the diode-transistor logic increases and this in turn increases the fan-out. Replacement of the diode D_3 in Fig. 1.10 by the transistor T_1 increases the base current of the transistor and the circuit is referred as a modified diode-transistor logic. The modified diode-transistor logic has more fan-out as compared to DTL. The circuit diagram of a modified diode-transistor logic is shown in Fig. 1.11.

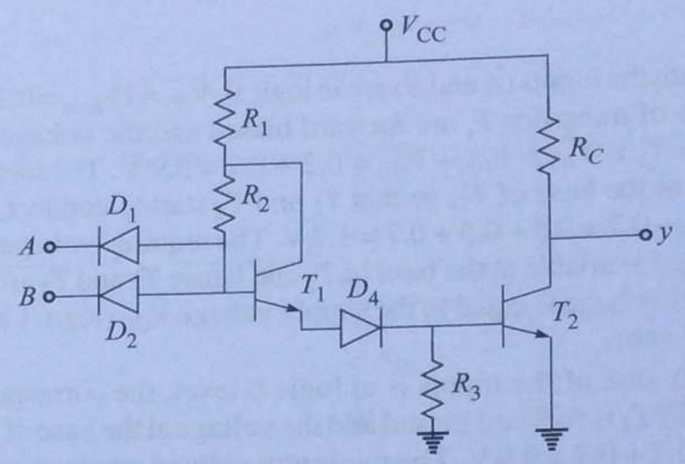


Fig. 1.11 Modified diode-transistor logic

1.9 TRANSISTOR-TRANSISTOR LOGIC (TTL)

Transistor-transistor logic is one of the popular saturated logic families. Transistor is the basic element of this logic family, which operates either in cut-off or saturation region. The first version of TTL is known as the standard TTL.

Standard TTLs are available in various forms:

1. TTL with passive pull-up
2. TTL with totem-pole output
3. TTL with open collector output
4. Tristate TTL

1.9.1 TTL with Passive Pull-Up

Figure 1.12 shows a two-input TTL NAND gate with passive pull-up. Transistor T_1 has two emitter terminals. These terminals act as the inputs of the gate, that is, input *A* and input *B*. The input voltages are logic 0 or logic 1, where logic 0 corresponds to 0.2 V and logic 1 corresponds to +5 V.

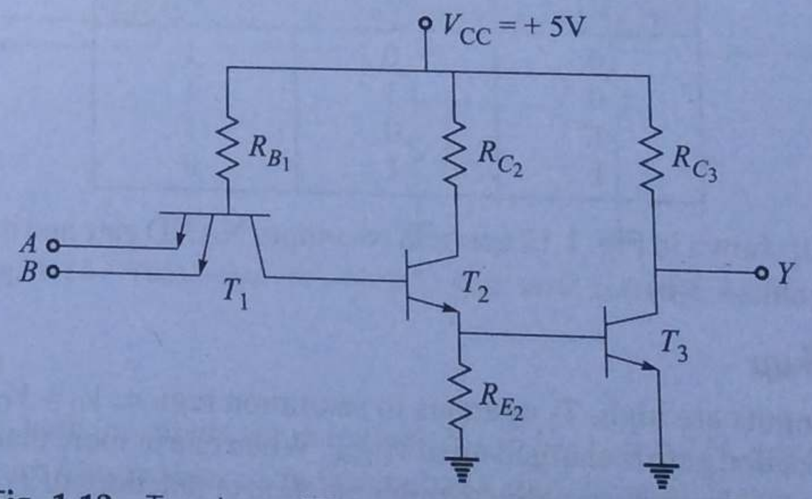


Fig. 1.12 Two-input TTL NAND gate with passive pull-up

Operation

- When both the inputs (A and B) are in logic 0, $V_{(0)} = V_{CEsat} = 0.2$ V, the emitter junctions of transistor T_1 are forward biased and the voltage at the base of transistor T_1 is $V_{B1} = V_{(0)} + V_{BE} = 0.2 + 0.7 = 0.9$ V. The minimum voltage required at the base of T_1, so that T_2 and T_3 start to conduct, is $V_{BEcut\ (in)} + V_{BEcut\ (in)} + 0.7 = 0.5 + 0.5 + 0.7 = 1.7$ V. The required voltage is greater than the voltage available at the base of T_1 and hence T_2 and T_3 are in cut-off and the output voltage is equal to the supply voltage V_{CC} (logic 1 level), output is in logic 1 state.
- When any one of the inputs is at logic 0 level, the corresponding emitter junction of T_1 is forward biased and the voltage at the base of T_1 is $V_{B1} = V_{(0)} + V_{BE} = 0.2 + 0.7 = 0.9$ V. The minimum voltage required at the base of T_1, so that T_2 and T_3 start to conduct, is $V_{BEcut\ (in)} + V_{BEcut\ (in)} + 0.7 = 0.5 + 0.5 + 0.7 = 1.7$ V. The required voltage is greater than the voltage available at the base of T_1 and hence T_2 and T_3 are in cut-off and the output voltage is equal to the supply voltage V_{CC}, output is in logic 1 state.
- When all the inputs are in logic 1 state, the emitter junctions of T_1 are reverse biased and the current supply by the source is sufficient to operate T_2 and T_3 in saturation and the output is in logic 0 state.

The operation of the circuit is summarized in Table 1.3(a).

Table 1.3(a) Operation of TTL NAND gate (Fig. 1.12)

Inputs		*Transistor T_1*		*Transistors T_2 and T_3*	*Output*
A	*B*	*Emitter junction A*	*Emitter junction B*		
Logic 0	Logic 0	Forward biased	Forward biased	Cut-off	Logic 1
Logic 0	Logic 1	Forward biased	Reverse biased	Cut-off	Logic 1
Logic 1	Logic 0	Reverse biased	Forward biased	Cut-off	Logic 1
Logic 1	Logic 1	Reverse biased	Reverse biased	Saturation	Logic 0

In terms of 0 and 1, Table 1.3(a) can be written as follows:

Table 1.3(b) Operation of TTL NAND gate (Fig. 1.12)

V_A	V_B	V_Y
0	0	1
0	1	1
1	0	1
1	1	0

The circuit shown in Fig. 1.12 acts as a two-input NAND gate and its truth table is given in Table 1.3(b).

Passive pull-up

When both inputs are high, T_3 operates in saturation region, $V_0 = V_{CEsat}$ and the capacitor of loaded gate is charged up to V_{CEsat}. When one or more than one inputs change to logic 0, the corresponding emitter junction or junctions of T_1 are forward biased and T_3 goes into cut-off. The capacitor of the loaded gate starts charging towards V_{CC} through the resistor R_{C3} as shown in Fig. 1.13.

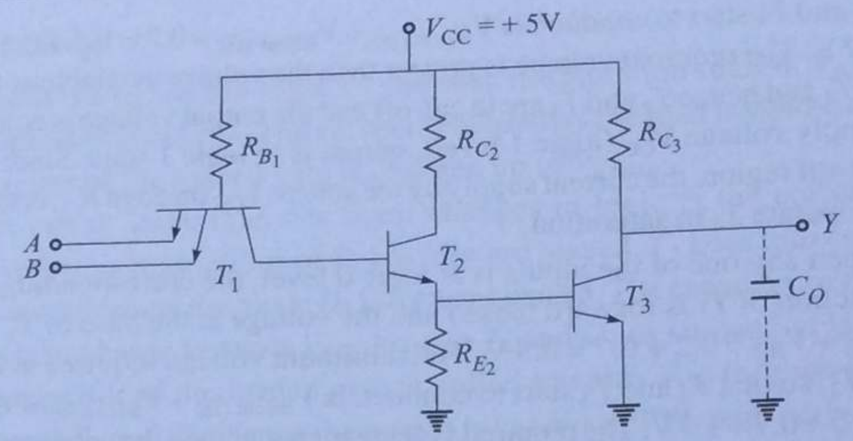


Fig. 1.13 TTL with passive pull-up

The capacitor of the loaded gate is pulled towards V_{CC} through the passive component R_{C_3} and hence the circuit is known as TTL with passive pull-up.

In TTL with passive pull-up, the time constant is $R_{C_3} \times C_O$. The speed of the circuit can be improved by decreasing R_{C_3}, which decreases the time constant. By decreasing the value of R_{C_3}, the power dissipation will go up due to the increased collector current of transistor T_3. This problem of TTL with passive pull-up is overcome in TTL with active pull-up.

1.9.2 TTL with Totem-Pole Output

Figure 1.14 shows the circuit of a two-input TTL NAND gate with totem-pole output. It is possible in TTL to improve the speed of operation by reducing the time constant without increasing power dissipation with the help of active pull-up. TTL with active pull-up is known as TTL with totem-pole output.

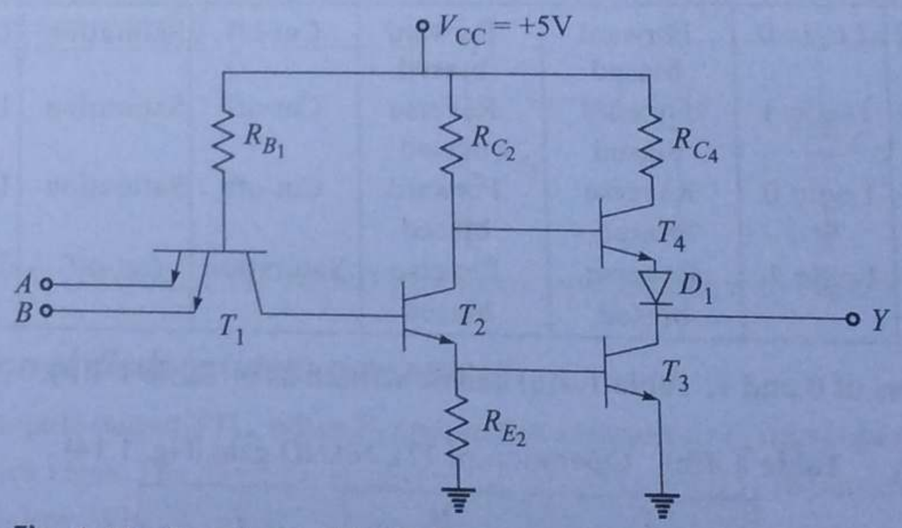


Fig. 1.14 Two-input TTL NAND gate with totem-pole output

Operation

- When both the inputs are in logic 0, $V_{(0)} = V_{CEsat} = 0.2$ V, emitter junctions of T_1 are forward biased and the voltage at the base of T_1 is $V_{B1} = V_{(0)} + V_{BE} = 0.2 + 0.7 = 0.9$ V. The minimum voltage required at the base of T_1, so that

T_2 and T_3 start to conduct, is $V_{BEcut\ (in)} + V_{BEcut\ (in)} + 0.7 = 0.5 + 0.5 + 0.7 =$ 1.7 V. The required voltage is greater than the voltage available at the base of T_1 and hence T_2 and T_3 are in cut-off and the output voltage is equal to the supply voltage V_{CC} (logic 1 level), output is in logic 1 state. Since T_2 is in cut-off region, the current supply by the source V_{CC} through R_{C_2} is sufficient to operate T_4 in saturation.

- When any one of the inputs is at logic 0 level, the corresponding emitter junction of T_1 is forward biased and the voltage at the base of T_1 is $V_{B1} = V_{(0)} + V_{BE} = 0.2 + 0.7 = 0.9$ V. The minimum voltage required at the base of T_1, so that T_2 and T_3 start to conduct, is $V_{BEcut\ (in)} + V_{BEcut\ (in)} + 0.7 = 0.5 + 0.5 + 0.7 = 1.7$ V. The required voltage is greater than the voltage available at the base of T_1 and hence T_2 and T_3 are in cut-off and the output voltage is equal to the supply voltage V_{CC} (logic 1 level), output is in logic 1 state. Since T_2 is in cut-off region, the current supply by the source V_{CC} through R_{C_2} is sufficient to operate T_4 in saturation.
- When all the inputs are in logic 1 state, the emitter junctions of T_1 are reverse biased and the current supply by the source is sufficient to operate T_2 and T_3 in saturation and the output is logic 0 state. Since T_2 is in saturation region, the voltage at the collector of T_2 is low and T_4 operates in cut-off.

The operation of the circuit is summarized in Table 1.4(a).

Table 1.4(a) Operation of TTL NAND gate (Fig. 1.14)

Inputs		*Transistor T_1*		*Transistors T_2 and T_3*	*Transistor T_4*	*Output Y*
A	*B*	*Emitter junction A*	*Emitter junction B*			
Logic 0	Logic 0	Forward biased	Forward biased	Cut-off	Saturation	Logic 1
Logic 0	Logic 1	Forward biased	Reverse biased	Cut-off	Saturation	Logic 1
Logic 1	Logic 0	Reverse biased	Forward biased	Cut-off	Saturation	Logic 1
Logic 1	Logic 1	Reverse biased	Reverse biased	Saturation	Cut-off	Logic 0

In terms of 0 and 1, Table 1.4(a) can be written as in Table 1.4(b).

Table 1.4(b) Operation of TTL NAND gate (Fig. 1.14)

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

The circuit shown in Fig. 1.14 acts as a two-input NAND gate and its truth table is given in Table 1.4(b).

Active pull-up

When both the inputs are high, T_3 operates in saturation region, $V_0 = V_{CEsat}$, and T_4 operates in cut-off. The current provided by the load is sinked by T_3 and the capacitor of the loaded gate is charged up to V_{CEsat}.

When one or more than one input changes to logic 0, the corresponding emitter junction or junctions of T_1 are forward biased. T_3 goes into cut-off and T_4 is in saturation and the diode D_1 is forward biased. The capacitor of the loaded gate starts to charge towards V_{CC} through T_4 and D_1 as shown in Fig. 1.15.

The capacitor of the loaded gate is pulled towards V_{CC} through the active components T_4 and D_1. Hence the circuit is known as TTL with active pull-up.

In totem-pole output, T_4 acts as an emitter follower. The output impedance of emitter follower is low. This means that the output voltage can change quickly from low state to high state.

When the output changes from high to low, T_3 operates in saturation and the capacitor of the loaded gate discharges quickly through T_3.

Due to the current spike problem, wired-AND connection must not be used for totem-pole output circuits, as discussed in the next section.

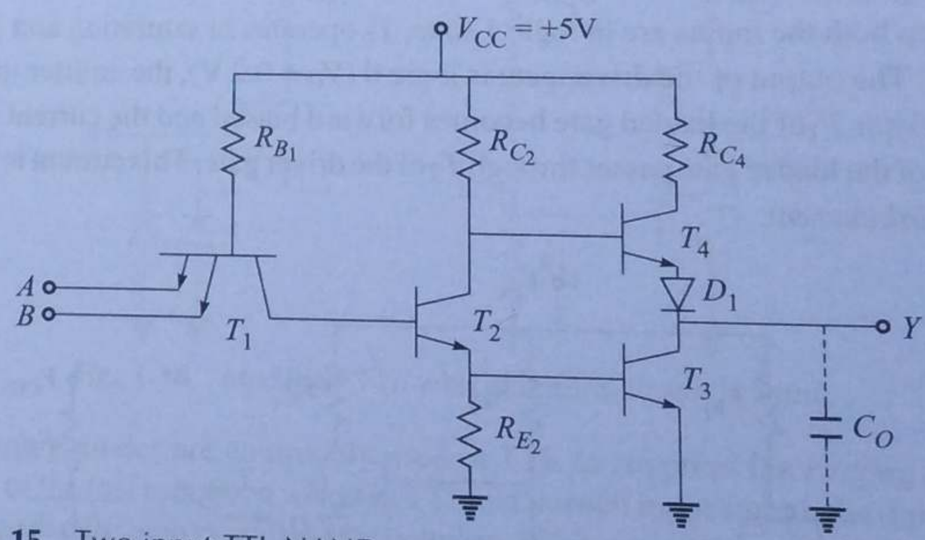


Fig. 1.15 Two-input TTL NAND gate totem-pole output with capacitive load

Function of diode in totem-pole output

In totem-pole output TTL, when T_3 operates in saturation, T_4 must operate in cut-off or vice versa. The diode D_1 is used in the circuit to keep T_4 in cut-off when the output is low. When T_2 and T_3 are in saturation, the voltage available at the base of T_4 is

$$V_{B_4} = V_{BE3sat} + V_{CE2sat} = 0.8 + 0.2 = 1 \text{ V} \quad (1.11)$$

In the absence of a diode, the voltage required at the base of T_4 so that it starts to conduct is

$$V_{B_4} = V_0 + V_{BEcut\ (in)}$$
$$= 0.2 + 0.5 = 0.7\ V \quad (1.12)$$

The voltage available at the base of T_4 is greater than the voltage required and hence both T_3 and T_4 are in saturation. To avoid this situation, D_1 is used in the circuit.

In the presence of a diode, the voltage required at the base of T_4, for the transistor to conduct, is

$$V_{B_4} = V_{(O)} + V_D + V_{BEcut\ (in)}$$
$$= 0.2 + 0.7 + 0.5 = 1.4\ V \quad (1.13)$$

The voltage available at the base of T_4 is less than the voltage required and hence T_4 is operating in cut-off.

Sink current

It is the current supplied by the load. Figure 1.16 shows a two-input modified TTL NAND gate driving a similar gate.

When both the inputs are in logic 1 state, T_3 operates in saturation and T_4 is in cut-off. The output of the driver gate is logic 0 ($V_O = 0.2$ V), the emitter junction of transistor T'_1 of the loaded gate becomes forward biased and the current supply by V_{CC} of the loaded gate passes through T_3 of the driver gate. This current is known as the *sink current*.

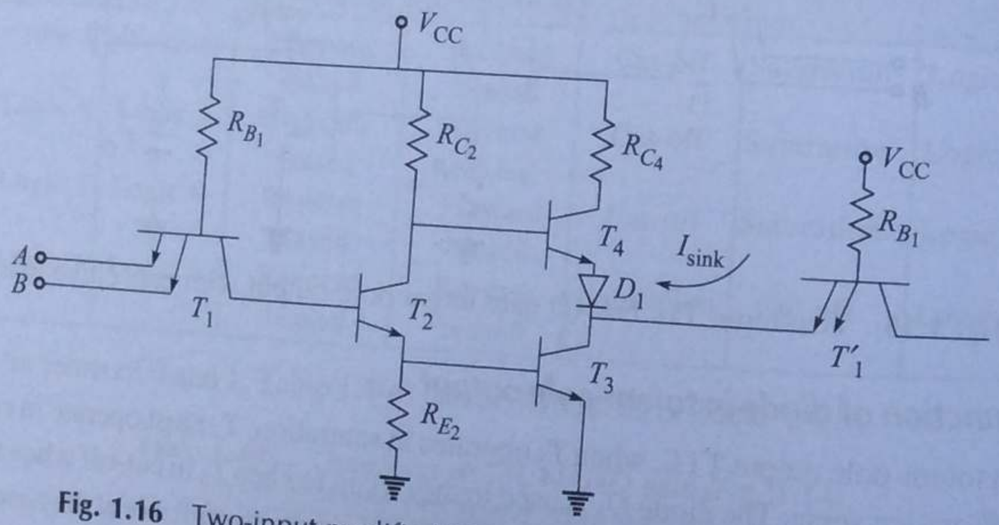


Fig. 1.16 Two-input modified TTL NAND gate driving same gate

Source current

It is the current supplied by the driver gate to the loading gate(s). Figure 1.17 shows a two-input modified TTL NAND gate driving a similar gate.

When both or any one of the inputs is in logic 0 state, T_3 operates in cut-off, and T_4 is in saturation. The source V_{CC} supplies the current to the loading gate through R_{C_4}, T_4 and D_1 as shown in Fig. 1.17. This current is known as the *source current*.

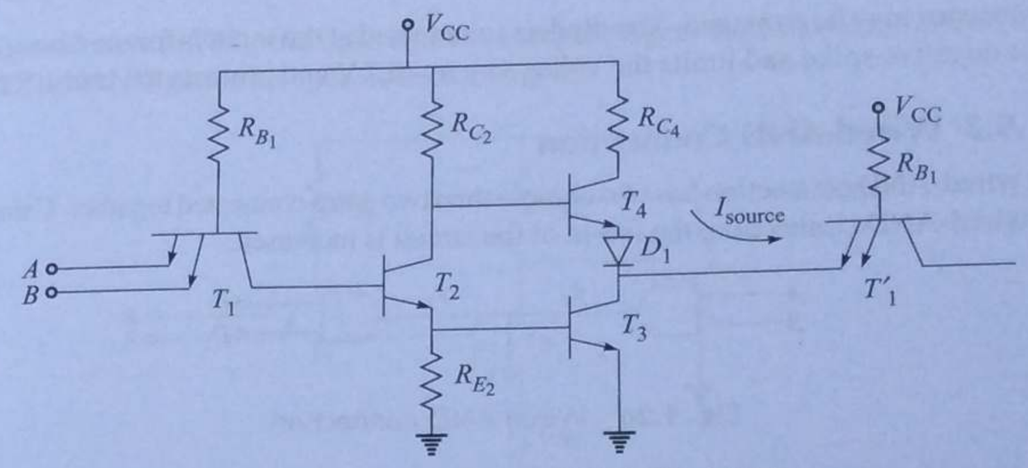


Fig. 1.17 Two-input modified TTL NAND

Clamped Diode

The circuit of a modified TTL with a clamping diode at input is shown in Fig. 1.18.

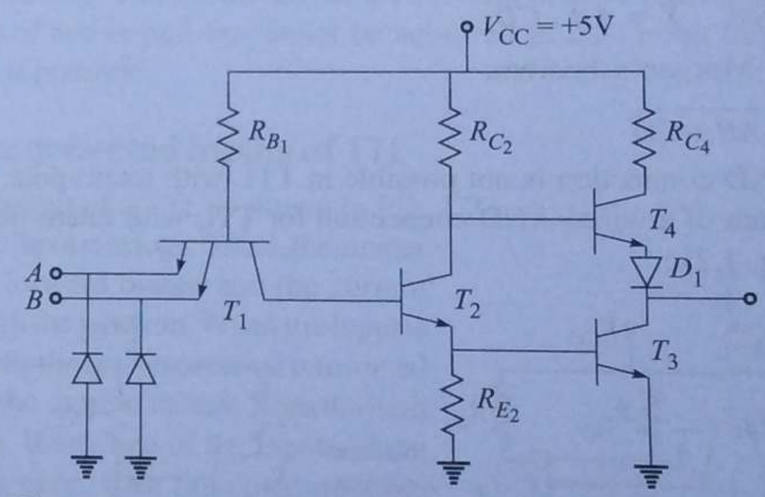


Fig. 1.18 Modified TTL with clamping diode at input

Clamping diodes are commonly used in TTL to suppress the *ringing* caused because of the fast transition found in TTL. At normal input signal, the diodes are reverse biased for high as well as low voltages. When transition occurs, the reactive component associated with the load causes ringing as shown in Fig. 1.19.

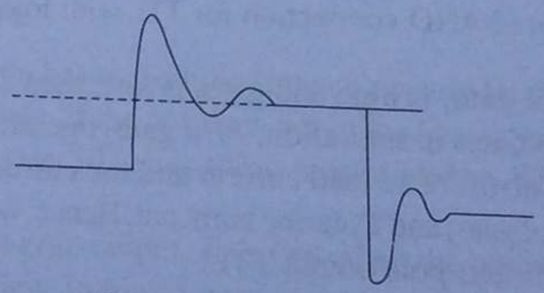


Fig. 1.19 Ringing effect

The rising on the positive side do not create any problem, but it makes the emitter junction reverse biased. Due to rising on the negative side, the emitter junction is more forward biased, the current flow through the emitter is very high, and the

transistor may be damaged. The diode(s) connected at the input is forward biased for negative spike and limits the voltage up to –0.7 V and protects the transistor.

1.9.3 Wired-AND Connection

A wired-AND connection has two or more than two gates connected together. Using a wired-AND connection, the fan-in of the circuit is increased.

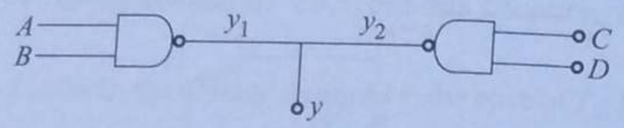


Fig. 1.20 Wired-AND connection

Here the outputs of two NAND gates are connected together.

$$Y = Y_1 \cdot Y_2$$

$$Y_1 = \overline{AB} \text{ and } Y_2 = \overline{CD}$$

Hence $Y = \overline{AB} \cdot \overline{CD}$

Using De-Morgan's theorem,

$$Y = \overline{AB + CD} \qquad (1.14)$$

Wired-AND connection is not possible in TTL with totem-pole output. The circuit diagram of a wired-AND connection for TTL with totem-pole output is shown in Fig. 1.21.

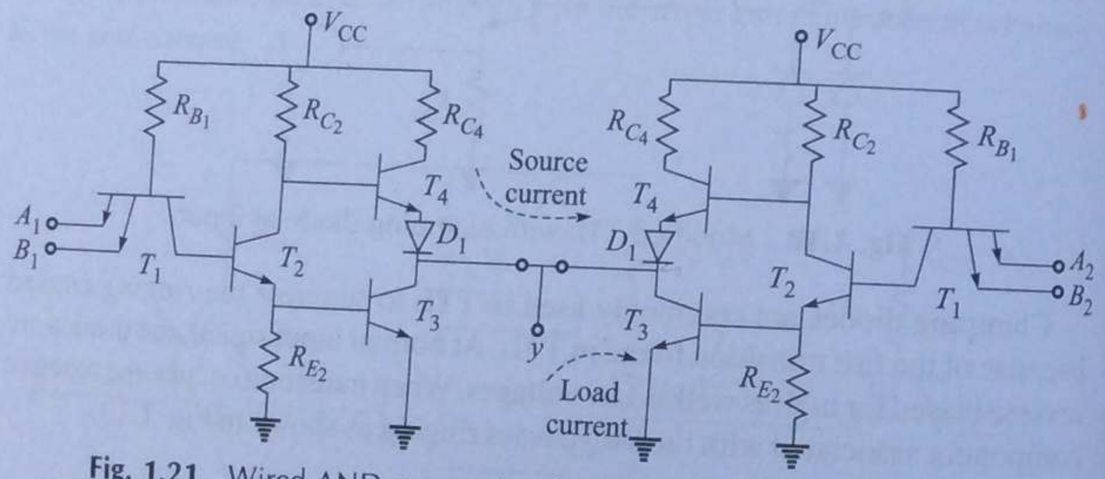


Fig. 1.21 Wired-AND connection for TTL with totem-pole output

When the output of gate$_1$ is high and gate$_2$ is low, then T_3 of gate$_1$ operates in cut-off, T_4 of gate$_1$ operates in saturation, T_3 of gate$_2$ operates in saturation and T_4 of gate$_2$ operates in cut-off. The load current and the current supplied by V_{CC} of gate$_1$ flow through T_3 of gate$_2$ and T_3 can be burnt out. Hence, wired-AND connection must not be used for totem-pole output TTL.

1.9.4 TTL with Open Collector Output

TTL with totem-pole output has a major problem that the two outputs of the two gates cannot be connected together. This problem of TTL with totem-pole output is overcome in TTL with open collector output. Figure 1.22 shows the circuit of a TTL NAND gate with open collector output.

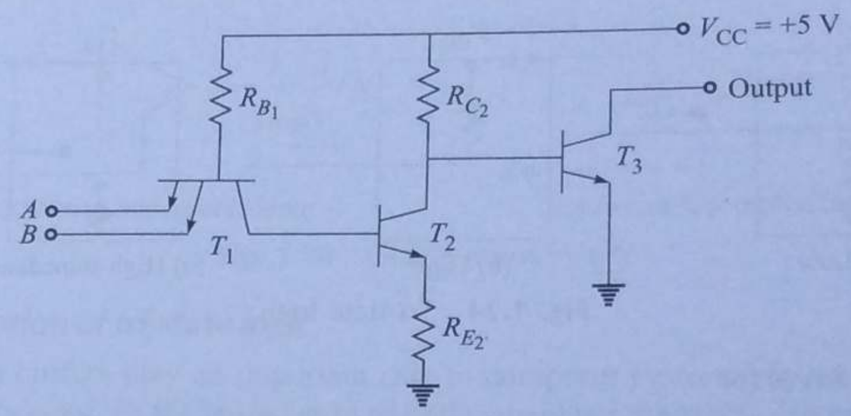


Fig. 1.22 TTL NAND gate with open collector output

The collector terminal of T_3 is available outside the IC where the external resistor is to be connected. The circuit acts as a TTL with passive pull-up and hence the advantages of active pull-up cannot be achieved in the circuit but wired-AND connection is possible.

1.9.5 Unconnected Inputs of TTL

The input circuit of a TTL is shown in Fig. 1.23.

When the input is in logic 0 state, the emitter junction is forward biased and the current flows through the junction. When the input is in logic 1 state, the emitter junction is reversed biased and the current cannot flow through the junction. If any one of the inputs of the TTL gate is open, then the corresponding junction cannot be forward biased, and the current cannot flow. The input acts exactly in the same way, as in case when logic 1 is applied to that input. Therefore in TTL ICs, all unconnected inputs are treated as logical 1s.

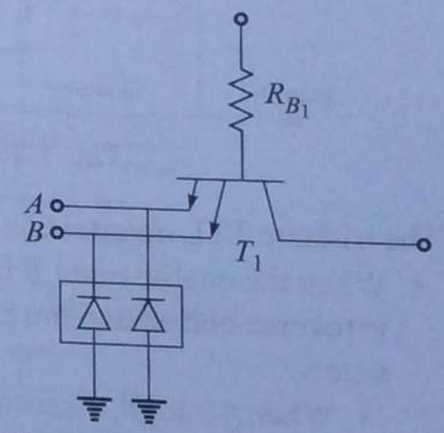


Fig. 1.23 Input circuit of TTL

1.9.6 Tri-state TTL

A normal digital circuit has two output states: Low and High. The output is either in high state or low state. If the output is not in the low state, it is definitely in the high state. The tri-state TTL has three output states: High, Low, and High-impedance.

In TTL with totem-pole output, T_3 is ON when the output is low and T_4 is ON when the output is high. In high-impedance state, both T_3 and T_4 in totem-pole arrangement are turned OFF and as a result, the output is open or floating.

When the output is low, the driver gate sinks the load current as shown in Fig. 1.24(a). When the output is high, the driver gate supplies the current to the

load as shown in Fig. 1.24(b). When the output is in high-impedance state, it acts as open or floating and there is no sink and source current as shown in Fig. 1.24(c).

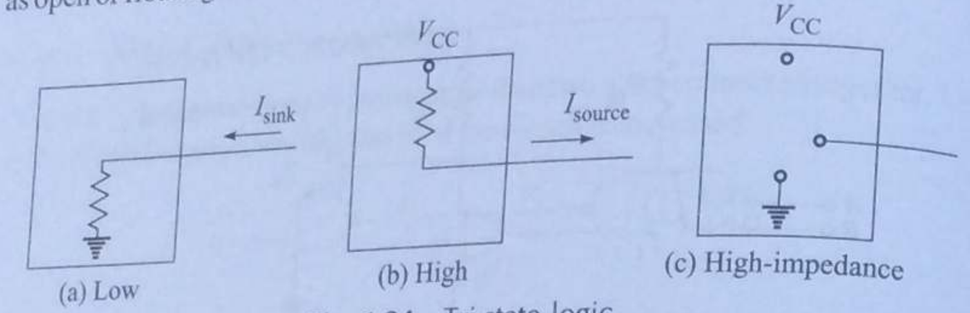


Fig. 1.24 Tri-state logic

Tri-state inverter

The circuit of a tri-state TTL inverter is shown in Fig. 1.25.

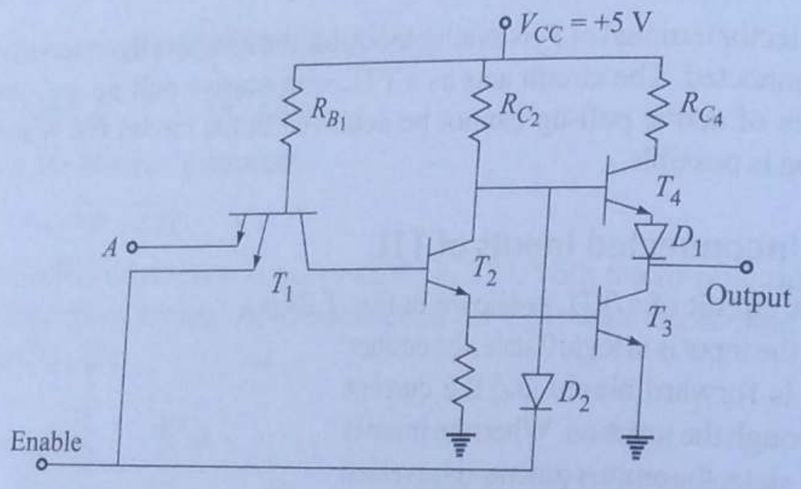


Fig. 1.25 Tri-state TTL inverter

The tri-state TTL inverter has two inputs—normal input A and enable input E.

- When the enable input E is High, the corresponding emitter junction of T_1 is reverse biased and the circuit operates as a normal inverter as explained below.
  - When A is high, the second emitter junction of T_1 is reverse biased. Current supplied by the source flows through the collector terminal of T_1, which forces T_2 and T_3 to be in saturation and the output is low. T_4 operates in cut-off.
  - When A is low, the corresponding emitter junction of T_1 is forward biased, current supplied by the source flows through the emitter terminal of T_1, transistors T_2 and T_3 operate in cut-off and the output is high. T_4 operates in ON state.
- When the enable input E is low, the corresponding emitter junction of T_1 is forward biased. Current supplied by the source flows through the emitter terminal of T_1, transistors T_2 and T_3 operate in cut-off. Because of low enable input, the diode D_2 is forward biased and the current supplied by the source flows through the diode D_2, transistor T_4 is operated in cut-off and thus the output is in high-impedance state.

The logic symbols for the active high and active low enable input inverters are shown in Figs 1.26(a) and (b), respectively.

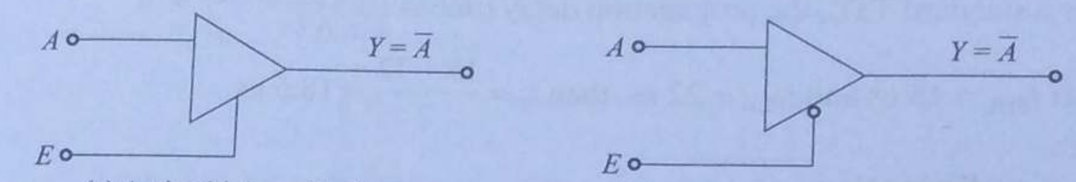


Fig. 1.26 Enable input inverter

Application of tri-state logic

Tri-state buffers play an important role in computer systems. In case of 8085 microprocessor, all the buses are in tri-state whenever the 8085 is in reset mode.

Figure 1.27 shows the common bus connected to four output devices, where A, B, C and D are the enabled inputs of tri-state buffer. Data present over the bus is given to the device as per the enabled input signal of the tri-state buffer. The enabled signal of the input device is active low.

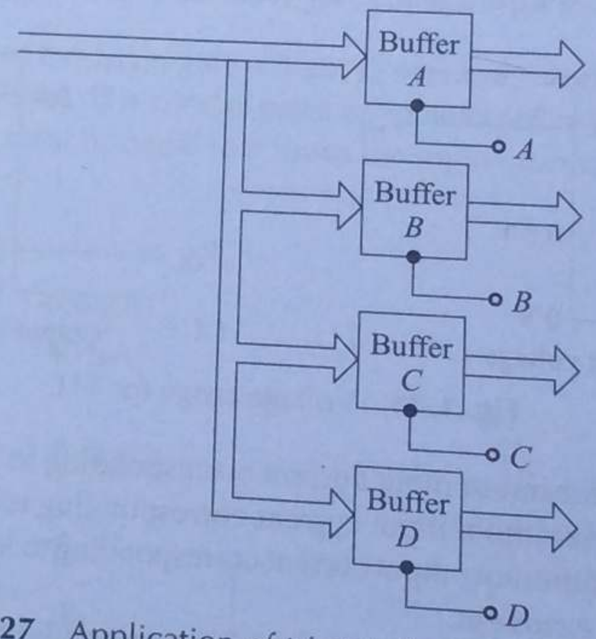


Fig. 1.27 Application of tri-state logic for data bus

When $A = 0, B = 1, C = 1, D = 1$, buffer A is enabled and others are disabled, and the data present over the line is given to device 1. When $A = 1, B = 0, C = 1, D = 1$, buffer B is enabled and others are disabled, and the data present over the line is given to device 2. When $A = 1, B = 1, C = 0, D = 1$, buffer C is enabled and others are disabled, and the data present over the line is given to device 3. When $A = 1, B = 1, C = 1, D = 0$, buffer D is enabled and others are disabled, and the data present over the line is given to device 4.

1.10 TTL PARAMETERS

Now let us study the typical values of TTL family parameters.

Speed of operation

The speed of operation of a TTL is specified in terms of propagation delay time. For a standard TTL, the propagation delay time is 18.5 ns.

$$\text{If } t_{PHL} = 15 \text{ ns and } t_{PLH} = 22 \text{ ns, then } t_p = \frac{15+22}{2} = 18.5 \text{ ns}$$

Power dissipation

It signifies the wastage of power in a digital circuit. It should be as minimum as possible. For a standard TTL, power dissipation is 19 mW.

Current and voltage parameters

V_{IH} : It is the minimum input voltage to be recognized as logic 1 state.
V_{IL} : It is the maximum input voltage to be recognized as logic 0 state.
V_{OH} : It is the minimum output voltage corresponding to logic 1 state
V_{OL} : It is the maximum output voltage corresponding to logic 0 state.

For a standard TTL family:

$$V_{IH} = 2\,V, \quad V_{OH} = 2.4\,V, \quad V_{IL} = 0.8\,V, \quad \text{and} \quad V_{OL} = 0.4\,V$$

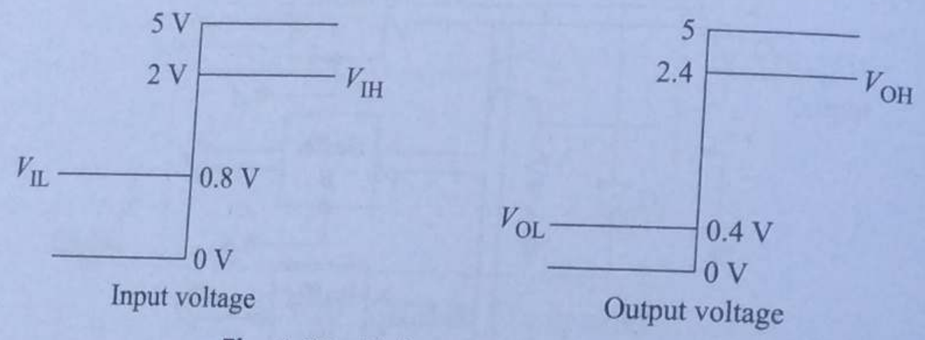


Fig. 1.28 Voltage range for TTL

I_{IH} : It is the minimum input current corresponding to logic 1 state.
I_{IL} : It is the maximum input current corresponding to logic 0 state
I_{OH} : It is the minimum output current corresponding to logic 1 state. It is called the *source current*.
I_{OL} : It is the maximum output current corresponding to logic 0 state. It is called the *sink current*.

For a standard TTL family:

$$I_{IH} = 40\,\mu A, \quad I_{OH} = -400\,\mu A, \quad I_{IL} = 1.6\,mA, \quad \text{and} \quad I_{OL} = 16\,mA$$

Fan-out

In digital applications, the output of a logic gate is given to one or more than one inputs of other logic gates. Fan-out is nothing but the capacity of the driver gate to drive a number of similar gates.

When the output of the driver gate is low, T_3 sinks current from the forward biased emitter junction of transistor of the load as shown in Fig. 1.29(a). If n similar gates are connected at the output, then the total sink current I_{OL} is n times the input current I_{IL}, where n is the fan-out of TTL.

$$I_{OL} = nI_{IL}$$

$$n = \frac{I_{OL}}{I_{IL}} = \frac{16\,mA}{1.6\,mA} = 10$$

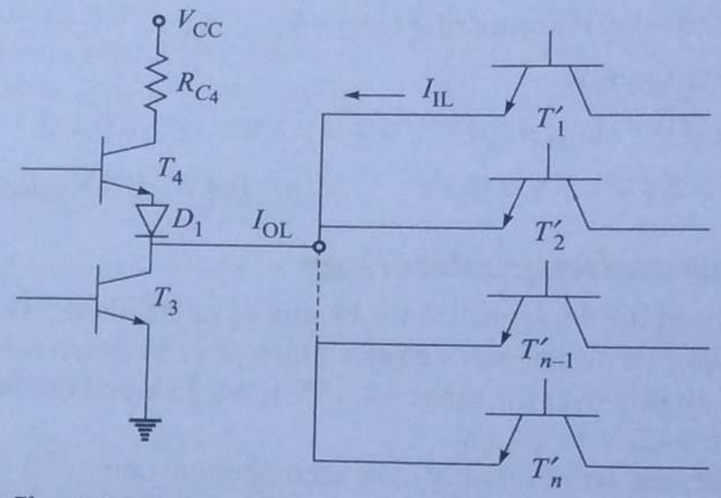


Fig. 1.29(a) TTL sinking the current from n gates

When the output of the driver gate is high, T_4 acts as a current source to the load as shown in Fig. 1.29(b). If n similar gates are connected at the output, then the total source current must be equal to n times the input current I_{IH}, where n is the fan-out of TTL.

$$I_{OH} = n\,I_{IH}$$

$$n = \frac{I_{OH}}{I_{IH}} = \frac{400\,\mu A}{40\,\mu A} = 10$$

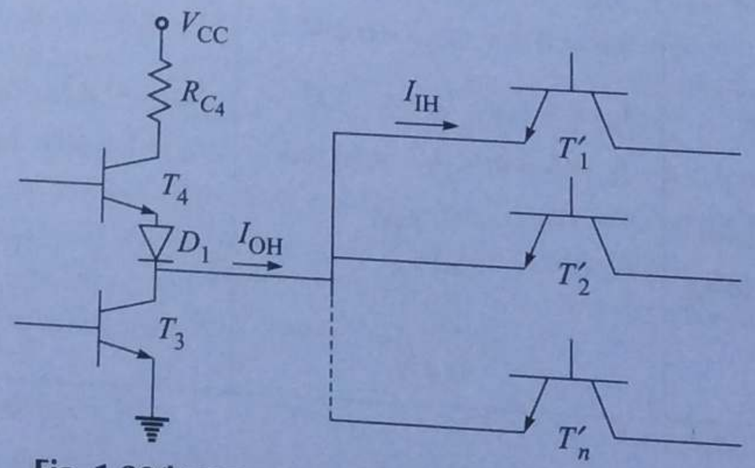


Fig. 1.29(b) TTL supplying the current to n gates

$$\text{Fan-out} = \text{minimum of } \left\{\frac{I_{OH}}{I_{IH}}, \frac{I_{OL}}{I_{IL}}\right\}$$

Fan-out for a standard TTL = minimum of {10, 10} = 10

Noise Margin

The noise immunity of a digital circuit is its ability to tolerate a noise signal. A quantitative measure of noise immunity is known as the *noise margin*.

Logic 1 level noise margin $(\Delta 1) = V_{OH} - V_{IH}$

Logic 0 level noise margin $(\Delta 0) = V_{IL} - V_{OL}$

For a standard family,

$V_{IH} = 2\,V, \quad V_{OH} = 2.4\,V, \quad V_{IL} = 0.8\,V, \quad V_{OL} = 0.4\,V$

$(\Delta 1) = 2.4\,V - 2\,V = 0.4\,V \qquad (\Delta 0) = 0.8\,V - 0.4\,V = 0.4\,V$

Supply voltage and temperature range

The 74 series and the 54 series are the examples of standard TTL logic families. These series operate on a power supply voltage of +5 V. But, it is found that the 74 series works reliably over the range +4.75 V to +5.25 V and the 54 series operates over the range +4.5 V to +5.5 V.

The 74 series can work reliably over a temperature range of 0°C to 70°C, while the 54 series can work over a temperature range of – 55°C to +125°C.

Summary of standard TTL

Table 1.5 summarizes the typical values of standard TTL parameters.

Table 1.5 Typical values for standard TTL parameters

Characteristics	*74 series*	*54 series*
Supply voltage	4.75 V to 5.25 V	4.5 V to 5.5 V
Temperature range	+0°C to 70°C	– 55°C to 125°C
Voltage levels	$V_{IH} = 2$ V, $V_{OH} = 2.4$ V $V_{IL} = 0.8$ V, $V_{OL} = 0.4$ V	$V_{IH} = 2$ V, $V_{OH} = 2.4$ V $V_{IL} = 0.8$ V, $V_{OL} = 0.4$ V
Current levels	$I_{IH} = 40\ \mu A$, $I_{OH} = -400\ \mu A$ $I_{IL} = 1.6$ mA, $I_{OL} = 16$ mA	$I_{IH} = 40\ \mu A$, $I_{OH} = -400\ \mu A$ $I_{IL} = 1.6$ mA, $I_{OL} = 16$ mA
Power dissipation	10 mW	10 mW
Propagation delay	10 ns	10 ns
Fan-out	10	10
Noise margin	0.4 V	0.4 V

1.11 COMMONLY USED ICs OF STANDARD TTL

Multiple emitter inputs and totem-pole output TTL circuits are used for different applications with a few exceptions. The 74 series is an example of a standard TTL family. It is available in different packages with different operations. The commonly used ICs of standard TTL are given in Table 1.6.

Table 1.6 Commonly used ICs of standard TTL

IC number	*Description*	*IC number*	*Description*
7400	Quad 2-inputs NAND gates	7411	Triple 3-inputs AND gates
7402	Quad 2-inputs NOR gates	7420	Dual 4-inputs NAND gates
7404	Hex Inverter	7424	Dual 4-inputs AND gates
7408	Quad 2-inputs AND gates	7432	Quad 2-inputs OR gates
7410	Triple 3-inputs NAND gates	7486	Quad 2-inputs EX-OR gates

1.12 IMPROVED TTL SERIES

TTL 54 series/74 series are the most popular and commonly used series of digital ICs. These series have the limitation of speed and power dissipation. These limitations are overcome up to a certain limit in the improved TTL series. The improved TTL series are as follows:

1. 74L series (low power TTL)
2. 74H series (high speed TTL)
3. 74S series (Schottky TTL)
4. 74LS series (low-power Schottky TTL)
5. 74AS series (advanced Schottky TTL)
6. 74ALS series (advanced low power Schottky TTL)
7. 74F series (fast TTL)

1.12.1 Low Power and High Speed TTL

The 74L series were developed to provide low power dissipation. The 74H series were developed to provide high speed. It is seen that the power dissipation goes down by increasing the charging resistance of the gate. Due to an increase in the value of the resistor, the delay is increased and the speed of operation is decreased. The 74L series has 1mW power dissipation and 33 ns propagation delay.

The speed of operation can be increased by reducing the value of the charging resistor. Due to low resistance, the power dissipation is increased. The high speed TTL, 74H series has 6 ns delay and 23 mW power dissipation.

The circuits of 74L series and 74H series are same as the circuit of a standard TTL, but they differ in circuit component values.

1.12.2 Schottky TTL

All the transistors in the circuits of standard TTL, low power TTL, and high speed TTL operate in saturation or cut-off region. When the transistor is in saturation, it stores the charge and the operation causes a storage-time delay during the transistor transition from ON to OFF; and this limits the circuit's switching speed.

In Schottky TTL families, Schottky transistors are used instead of normal transistors. The Schottky transistor is operated in active region or cut-off region, it never goes into saturation and the storage time

The Schottky transistor is obtained by using a Schottky barrier diode between the base and the collector terminals of the transistor as shown in Fig.1.30(a).

The Schottky diode has a forward biased voltage of 0.25 V. Because of this diode connected between the base and the collector terminals of the transistor, the collector junction of the transistor cannot get forward biased and the transistor never goes in saturation; the transistor operates in cut-off or active region. The symbol of Schottky transistor is shown in Fig. 1.30(b).

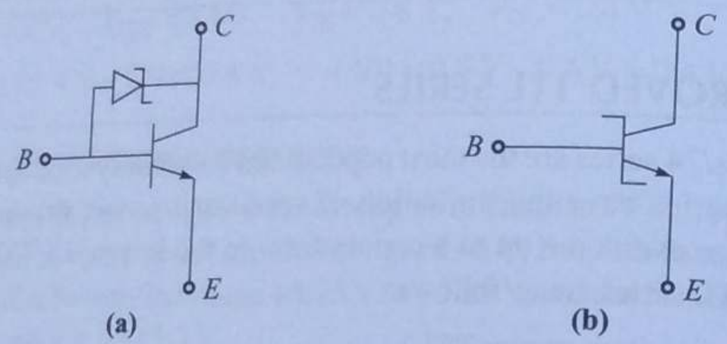


Fig. 1.30 Schottky transistor and Schottky symbol

The 74S series is an example of Schottky TTL. The propagation delay of Schottky TTL is 3 ns only, which is twice as fast as the 74H series. Figure 1.31 shows a basic NAND gate in Schottky TTL series.

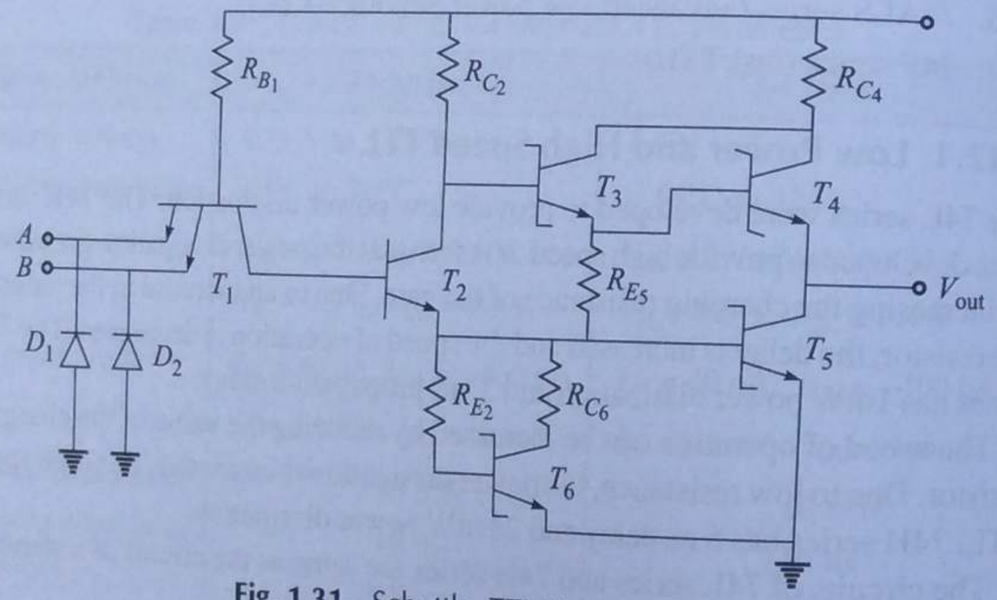


Fig. 1.31 Schottky TTL NAND gate

1.12.3 Low Power Schottky TTL

The 74LS series is a low power Schottky TTL. It uses Schottky transistors. It is similar to 74S, but it has a large value of charging resistor than 74S series. The large resistor values reduce the circuit power requirement but increase the time delay. The 74LS series has 9.5 ns propagation delay and 2 mW power dissipation. The 74AS series is an advanced Schottky TTL. The 74ALS series is an advanced low power Schottky TTL and the 74F series is a fast TTL.

1.13 COMPARISON OF TTL FAMILIES

A comparison of TTL families with respect to their common characteristics is given in Table 1.7.

Table. 1.7 Comparison of TTL families

Character-istics	*74*	*74 LS*	*74 S*	*74 ALS*	*74 AS*	*74 F*
Min. V_{OH}/V_{OL}	2.4/0.4	2.7/0.5	2.7/0.5	2.7/0.5	2.7/0.5	2.7/0.5
Max. V_{IH}/V_{IL}	2/0.8	2/0.8	2/0.8	2/0.8	2/0.8	2/0.8
Min. I_{OH}/I_{OL}	–0.4/16	–0.4/16	–1/20	–0.4/16	–2/20	–1/20
Max. I_{IH}/I_{IL}	40/(–1.6)	20/(–0.4)	50/(–2)	20/(– 0.2)	0.2/(–2)	20/(– 0.6)
T_{pd} (ns)	10	10	3	4	1.5	2.5
P.D. per gate (mW)	10	2	20	1	20	4

1.14 EMITTER COUPLED LOGIC

Emitter coupled logic (ECL) is faster than TTL family. The transistors of an emitter coupled logic are operated in cut-off or active region, it never goes in saturation and therefore the storage time is eliminated. Emitter coupled logic family is an example of unsaturated logic family. Figure 1.32 shows the circuit of an emitter-coupled logic OR/NOR gate.

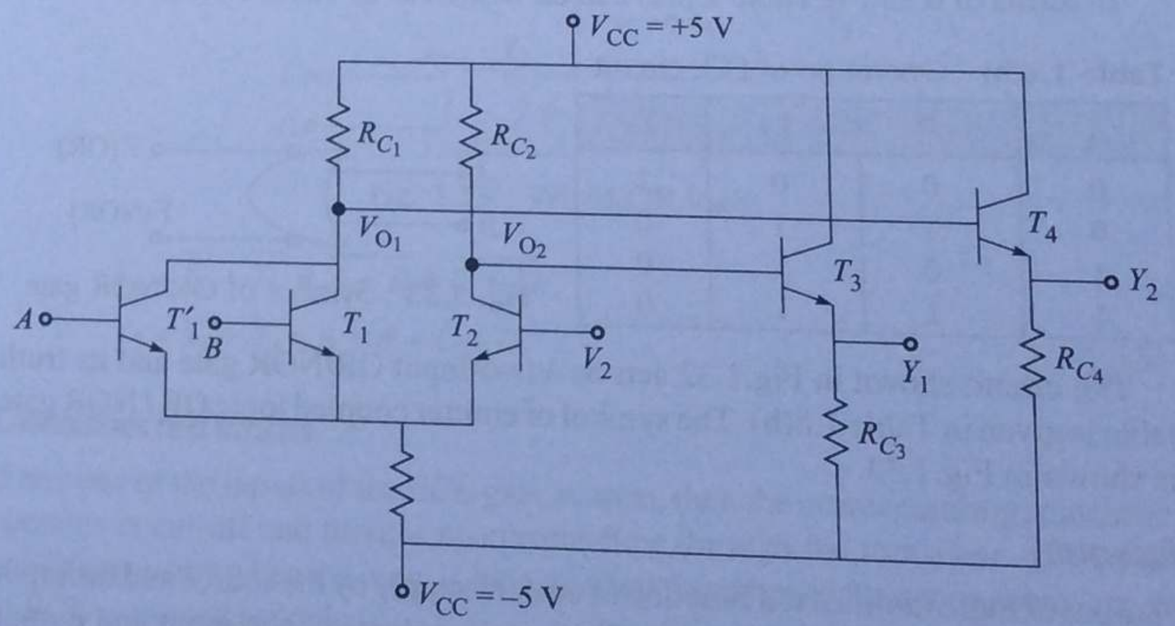


Fig. 1.32 Emitter coupled logic OR/NOR gate

The circuit consists of difference amplifiers and emitter followers. Emitter terminals of the two transistors are connected together and hence it is called as emitter coupled logic.

The emitter followers are used at the output of difference amplifier to shift the DC level. The circuit has two outputs Y_1 and Y_2, which are complementary. Y_1 corresponds to OR logic and Y_2 corresponds to NOR logic.

Operation

The input voltage of T_2 is $V_2 = V_{ref} = -1.15$ V.

- When both the inputs are in logic 0, T_1 and T'_1 operate in cut-off and T_2 operates in active region, voltage V_{O_1} is high, T_3 is ON, and the output at Y_2 is logic 1, voltage V_{O_2} is low, T_4 operates in cut-off and the output at Y_1 is logic 0.
- When any one of the inputs is in logic 1 level, the corresponding transistors T_1 or T'_1 are operated in active region and T_2 operates in cut-off, voltage V_{O_1} is low, T_3 operates in cut-off and Y_2 is logic 0, voltage V_{O_2} is high, T_4 operates in active region and Y_1 is logic 1.
- When both the inputs are in logic 1 state, T_1 and T'_1 operate in active region and T_2 operates in cut-off, voltage V_{O_1} is low, T_3 operates in cut-off and Y_2 is logic 0, voltage V_{O_2} is high, T_4 operates in active region and Y_1 is logic 1.

The operation of the circuit is summarized in Table 1.8(a).

Table 1.8(a) Operation of ECL circuit

Inputs		*Transistors*			*Transistors*		*Output*	
A	*B*	T_1	T'_1	T_2	T_3	T_4	Y_1	Y_2
Logic 0	Logic 0	Cut-off	Cut-off	Active	Active	Cut-off	Logic 0	Logic 1
Logic 0	Logic 1	Cut-off	Active	Cut-off	Cut-off	Active	Logic 1	Logic 0
Logic 1	Logic 0	Active	Cut-off	Cut-off	Cut-off	Active	Logic 1	Logic 0
Logic 1	Logic 1	Active	Active	Cut-off	Cut-off	Active	Logic 1	Logic 0

In terms of 0 and 1, Table 1.8(a) can be written as in Table 1.8(b).

Table 1.8(b) Operation of ECL circuit

A	*B*	Y_1	Y_2
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

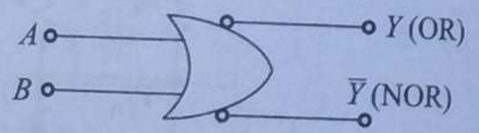


Fig. 1.33 Symbol of OR/NOR gate

The circuit shown in Fig.1.32 acts as a two-input OR/NOR gate and its truth table is given in Table 1.8(b). The symbol of emitter coupled logic OR / NOR gate is shown in Fig.1.33.

Fan-out

Fan-out of logic families is a function of current supply by the source and the input current of individual loaded gates. These are functions of the input and output resistances of the logic family. If the input resistance of a loaded gate is high, then the required current is less. If the output resistance is low, then the supply current is more. For emitter coupled logic, the transistor is used in the emitter follower mode due to which the input resistance is high and the output resistance is low. Thus, current supply by the driving gate is more and the current required to drive the individual gate is less and therefore the fan-out is large.

Wired-OR logic

The ECL circuit has two outputs Y_1 and Y_2. Y_1 is the output of OR logic and Y_2 is the output of NOR logic. ($Y_1 = A + B$, $Y_2 = \overline{Y} = \overline{A + B}$). When the outputs of two or more than two gates are connected, then an additional logic is realized without using any additional hardware. Consider the circuit shown in Fig. 1.34.

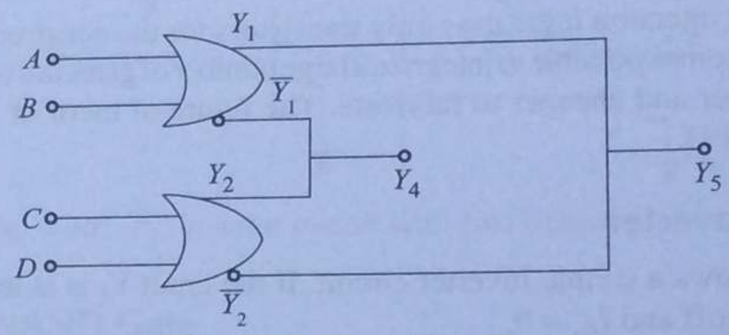


Fig. 1.34 Wired-OR logic

$$Y_4 = \overline{Y_1} + Y_2 = \overline{A + B} + C + D$$

$$Y_5 = Y_1 + \overline{Y_2} = A + B + \overline{C + D}$$

Consider another circuit shown in Fig. 1.35.

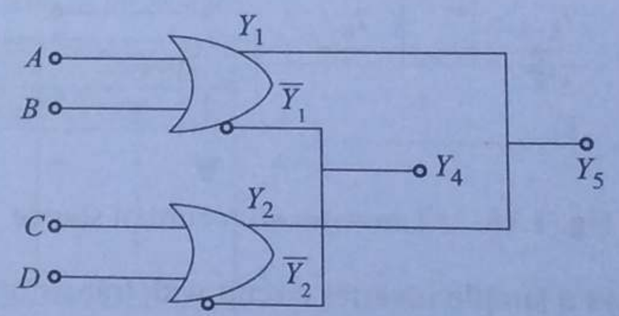


Fig. 1.35 Wired-OR logic

$$Y_4 = \overline{Y_1} + \overline{Y_2} = \overline{A + B} + \overline{C + D}$$

$$Y_5 = Y_1 + Y_2 = A + B + C + D \qquad (1.15)$$

Unconnected inputs

If any one of the inputs of the ECL gate is open, then the corresponding transistor operates in cut-off and there is no current flow through the transistor. The same condition occurs when the input is in logic 0 level and hence the unconnected input of ECL is treated as logic 0.

ECL characteristics

Following are the characteristics of an ECL circuit:

1. Less propagation delay time ($t_d = 1$ ns).
2. The logic low-level voltage is –1.7 V and the logic high level voltage is –0.8 V.

3. Poor noise margin.
4. Power dissipation within the range of 40–55 mW.
5. Fan-out is 25.

1.15 INTEGRATED INJECTION LOGIC (I^2L)

The integrated injection logic uses only transistors for the construction of a gate and hence it becomes possible to integrate a large number of gates in a single package. This IC is easier and cheaper to fabricate. The figure of merit of I^2L circuits is quite small (4 PJ).

1.15.1 I^2L Inverter

Figure 1.36 shows a simple inverter circuit. If the input V_1 is at low logic level, transistor T_1 is off and $I_{B_1} = 0$.

The input source acts as a sink for the current supplied by the current source I_1 and the output is at high logic level. If the input is high, the base current $I_{B_1} = I_S + I_1$ and T_1 operates in saturation. The output is at low logic level.

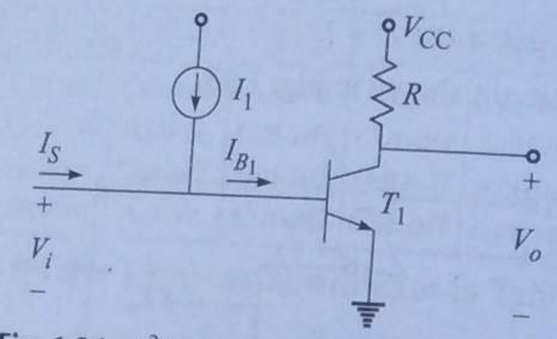


Fig. 1.36 I^2L Inverter with current source

Figure 1.37 shows a simple inverter circuit with transistor T_2 as the constant current source, hence T_2 is in series as a constant current source.

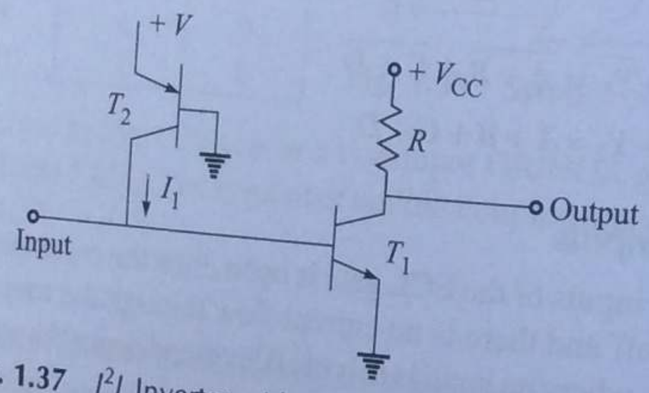


Fig. 1.37 I^2L Inverter with transistor as current source

When the input is low, the source sinks the current and T_1 is off and the output is high. When the input is high, the base current of T_1 is the sum of currents I_1 and the current supplied by the source, T_1 is ON and the output is low.

Figure 1.38 shows the I^2L inverter circuit with two output terminals. Transistor T_1 has two collector terminals. The outputs are connected directly to the inputs of other I^2L gate.

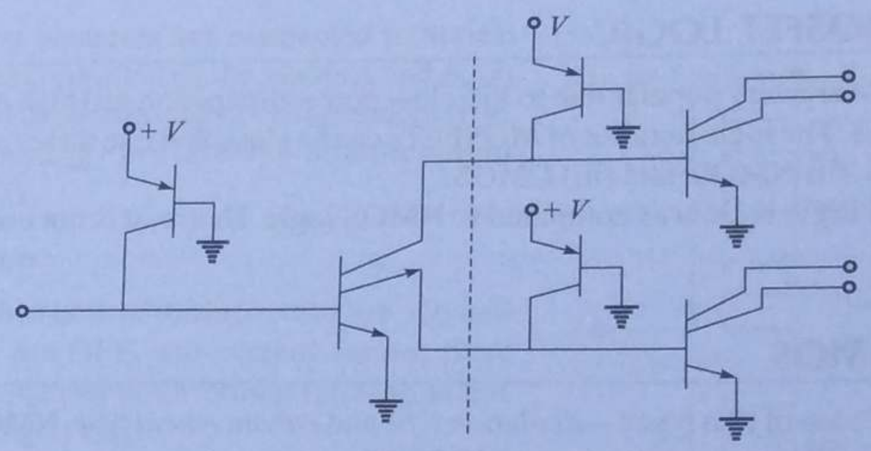


Fig. 1.38 I^2L inverter circuit with two output terminals

1.15.2 I^2L NAND Gate

Figure 1.39 shows the I^2L NAND gate. When inputs A and B are low or any one of the inputs is low, the current provided by T_2 is sinked by the source, T_1 is OFF, and the output is high. When both the inputs are high, the base current of T_1 is the sum of currents provided by the source and T_2, transistor T_1 is ON and the output is low.

Table 1.9 Truth Table of NOR gate

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

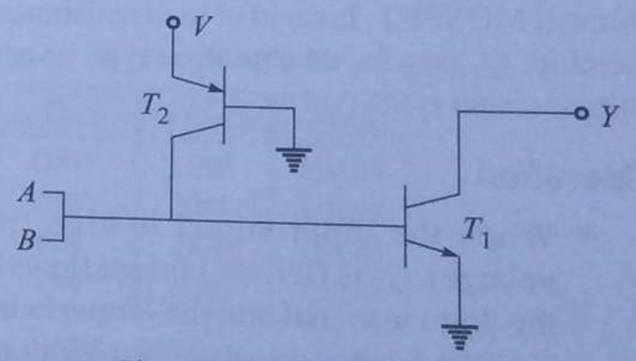


Fig. 1.39 I^2L NAND gate

1.15.3 I^2L NOR Gate

Figure 1.40 shows the I^2L NOR gate. The circuit has two inverters with their outputs connected together. When both or any one of the inputs is high, the output of the corresponding inverter is low and the resulting output is low.

When both inputs are low, the output of both the inverters is high and the result is also high.

Table 1.10 Truth Table of NOR gate

Inputs		*Outputs*		*Y*
A	*B*	Y_1	Y_2	
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

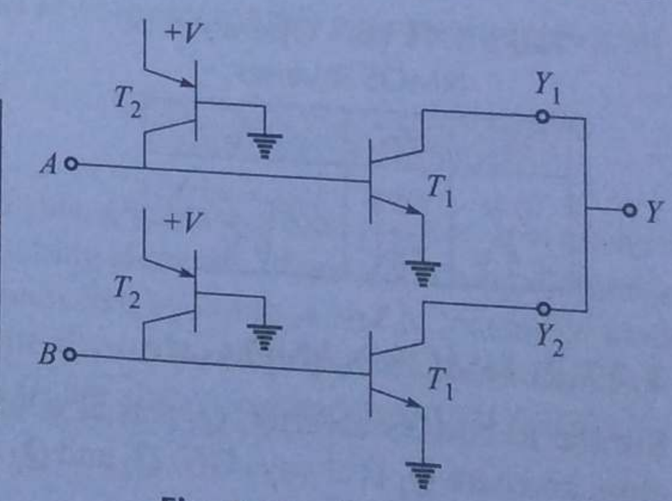


Fig. 1.40 I^2L NOR gate

1.16 MOSFET LOGIC

MOSFETs are very popular due to their low power dissipation and high density of fabrication. The logic families of MOSFETs can be classified into three categories: (i) PMOS, (ii) NMOS, and (iii) CMOS.

PMOS logic is slow as compared to NMOS logic. Hence, it is not used in new designs.

1.17 NMOS

MOSFETs are of two types—*depletion type* and *enhancement type*. NMOS family uses only *n*-channel enhancement MOSFETs.

1.17.1 NMOS Inverter

Figure 1.41 shows the circuit of an NMOS inverter consisting of *n*-channel MOSFETs. When the drain and gate terminals of MOSFETs are short-circuited, then it acts as a resistor.

In Fig. 1.41, MOSFET Q_1 acts as a load resistor and MOSFET Q_2 acts as a switching element. Q_1 is always ON; the load resistance is equal to R_{ON} of the *n*-channel MOSFET. Instead of load resistance, Q_1 is used, which reduces the size of the chip. Q_1 may be of depletion type or enhancement type. But Q_2 is always of enhancement type.

Operation

- When the input signal is high (positive voltage), Q_2 is ON, the current flows through the drain terminal and the output is low.
- When the input signal is low (0 V or negative voltage), Q_2 is OFF, there is no current flow through the circuit and the output is high (V_{DD}).

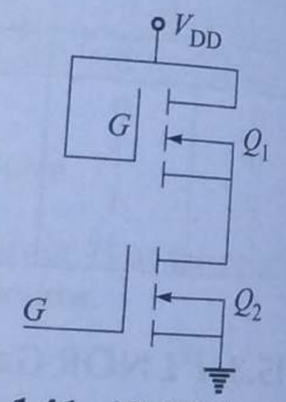


Fig. 1.41 NMOS inverter

The operation of the circuit is summarized in Table 1.11 (a). In terms of 0 and 1, we can write it as in Table 1.11(b). The circuit in Fig. 1.41 acts as a NOT gate and its truth table is given in Table 1.11(b).

Table 1.11 (a) Operation of NMOS inverter

V_{in}	Q_2	V_O
0 V	OFF	V_{DD} = +5 V
+5 V	ON	0 V

Table 1.11 (b) Operation of NMOS inverter

V_{in}	V_O
0	1
1	0

1.17.2 NMOS NAND Gate

Similar to NMOS inverter, Q_1 acts as a load resistor that reduces the size of the chip; and here Q_1 is always ON. Q_2 and Q_3 are the switching elements. These two switching elements are connected in series, which are controlled by the inputs *A* and *B*. Q_1 may be of depletion type or enhancement type, but Q_2 and Q_3 are always enhancement MOSFETs.

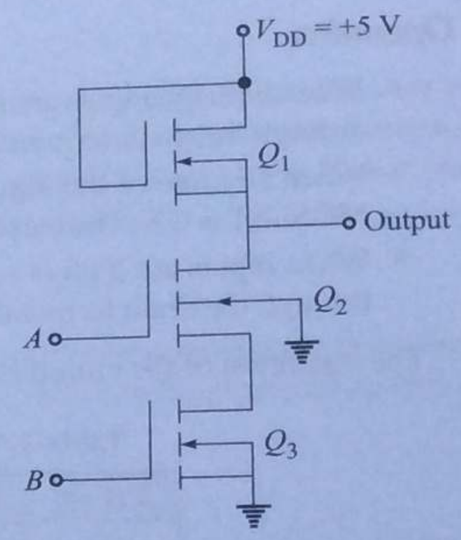


Fig. 1.42 A two-input NAND gate using NMOS

Operation

- When both of the inputs are low, Q_2 and Q_3 are OFF, the current cannot flow through the drain terminal and the output is high (V_{DD}).
- When any one of the inputs is low (0V or negative), then the corresponding MOSFET is OFF. There is no current flow through the circuit and the output is high (V_{DD}).
- When inputs are high (+ve voltage), Q_2 and Q_3 are ON. The current flows through the drain terminal and the output is low.

The operation of the circuit is summarized in Table 1.12(a).

Table 1.12(a) Operation of NMOS NAND gate

A	*B*	Q_2	Q_3	V_o
LOW	LOW	OFF	OFF	HIGH
LOW	HIGH	OFF	ON	HIGH
HIGH	LOW	ON	OFF	HIGH
HIGH	HIGH	ON	ON	LOW

In terms of 0 and 1, Table 1.12(a) can be written as in Table 1.12(b).

Table 1.12(b) Operation of NMOS NAND gate

A	*B*	V_O
0	0	1
0	1	1
1	0	1
1	1	0

The circuit shown in Fig.1.42 acts as a two-input NAND gate and its truth table is given in Table 1.12(b).

1.17.3 NMOS NOR Gate

Figure 1.43 shows a two-input NMOS gate. Q_1 acts as a load resistor. It is always ON. MOSFETs Q_2 and Q_3 are the switching elements. These switching elements are connected in parallel, which are controlled by inputs *A* and *B*. Instead of load resistance, Q_1 is used which reduces the size of the chip. Q_1 may be of depletion type or enhancement type, but Q_2 and Q_3 are always enhancement MOSFETs.

Operation

- When both the inputs are low, Q_2 and Q_3 are OFF. The current cannot flow through the drain terminal and the output is high (V_{DD}).
- When any one of the inputs is high (0 V or –ve), then the corresponding MOSFET is ON. The current flows through the circuit and the output is low.
- When inputs are high (+ve voltage), Q_2 and Q_3 are ON. The current flows through the drain terminal and the output is low.

The operation of the circuit is summarized in Table 1.13(a).

Table 1.13(a) NMOS NOR gate

A	*B*	Q_2	Q_3	V_o
LOW	LOW	OFF	OFF	HIGH
LOW	HIGH	OFF	ON	LOW
HIGH	LOW	ON	OFF	LOW
HIGH	HIGH	ON	ON	LOW

In terms of 0 and 1, Table 1.13(a) can be written as in Table 1.13(b).

Table 1.13(b) NMOS NOR gate

A	*B*	V_o
0	0	1
0	1	0
1	0	0
1	1	0

The circuit shown in Fig. 1.43 acts as a two-inputs NAND gate and its truth table is given in Table 1.13(b).

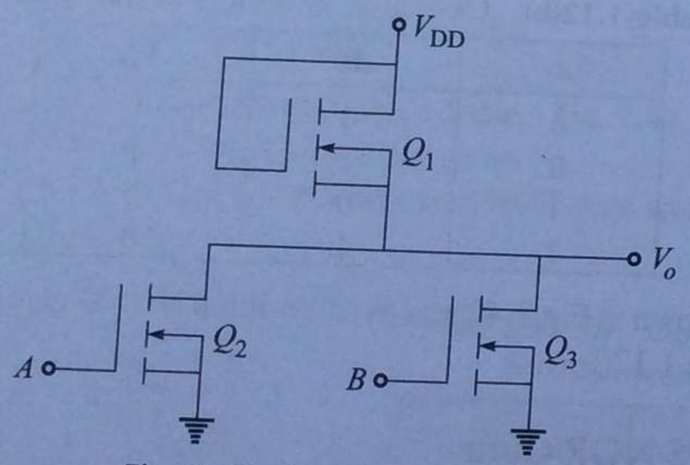


Fig. 1.43 NMOS NOR gate

1.17.4 Fan-out

We have seen that the fan-out of a logic family is the function of its input and output impedance. For high input impedance, the fan-out is large. MOSFET devices have very high input impedance, therefore fan-out is large. Since the number of driving gates is more, the capacitor at the driving gate output considerably reduces the speed of MOSFET gate.

1.17.5 Propagation Delay Time

It is a function of the capacitor of loaded gate and the charging resistor. In case of MOS devices, the large capacitor is present at input and output and the resistor through which the capacitor gets charged or discharged is also high. Hence, the propagation delay is large and the speed of operation is low.

1.17.6 Power Dissipation

It is a function of current supply by the source and resistance of the load. The power supply by the source in MOS logic family is small and hence the power dissipation is low.

1.17.7 Characteristics of NMOS

Table 1.14 summarizes the characteristics of NMOS.

Table 1.14 Characteristics of NMOS

Parameter	*Value*	*Parameter*	*Value*	*Parameter*	*Value*
V_{IH}	2.0 V	I_{OH}	– 400 μA	t_{PHL}	60 ns
V_{IL}	0.8 V	I_{OL}	2 mA	t_{PLH}	45 ns
V_{OH}	2.4 V	Fan-out	30	P.D.	0.1 mW
V_{OL}	0.45 V	Noise margin	1.5 V		

1.18 CMOS

CMOS family uses *n*-channel and *p*-channel MOSFETs. In CMOS, *p*-channel and *n*-channel MOS devices are fabricated on the same chip, which makes its fabrication complicated but it reduces the packaging density, and has small power consumption. Hence, CMOS is ideally suited for battery-operated systems.

1.18.1 CMOS Inverter

Figure 1.44 shows a CMOS logic inverter. For the circuit, the logic levels are 0 V and V_{CC}. It is important to note that the *p*-channel MOSFET is ON, when the input is 0 V and the *n*-channel MOSFET is ON, when the input is V_{CC}. Q_1 is *p*-channel and Q_2 is *n*-channel. When Q_1 is ON, the output voltage is equal to V_{CC} and when Q_2 is ON, the output voltage is equal to 0 V.

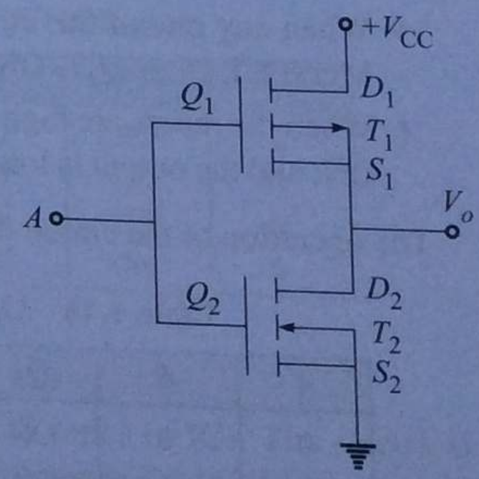


Fig. 1.44 CMOS inverter

Operation

- When the input is low, Q_1 is ON and Q_2 is OFF, output is high.
- When the input is high, Q_1 is OFF and Q_2 is ON, output is low.

Table 1.15 shows the operation of CMOS inverter.

Table 1.15 Operation of CMOS inverter

A	Q_1	Q_2	V_O
0	ON	OFF	1
1	OFF	ON	0

1.18.2 CMOS NAND Gate

Figure 1.45 shows a two-inputs CMOS logic NAND gate. It consists of two *p*-channel and two *n*-channel MOSFETs. *p*-channel MOSFETs are connected in parallel and *n*-channel MOSFETs are connected in series. Here, Q_1 and Q_2 are *p*-channel MOSFETs and Q_3 and Q_4 are *n*-channel MOSFETs.

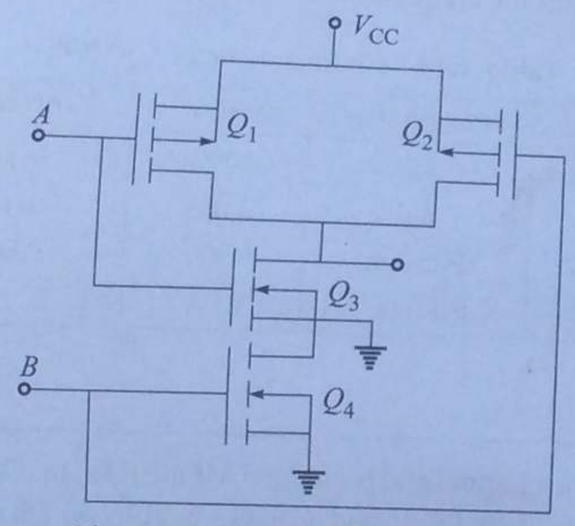


Fig. 1.45 CMOS NAND gate

Operation

- When the inputs are low, Q_1 and Q_2 are ON, Q_3 and Q_4 are OFF, and the output is high (V_{DD}).
- When any one of the inputs is low (0 V or –ve), then the corresponding MOSFET Q_1 or Q_2 is ON, Q_3 or Q_4 is ON, and the output is high.
- When the inputs are high (+ve voltage), Q_1 and Q_2 are OFF, Q_3 and Q_4 are ON, and the output is low.

The operation of the circuit is summarized in Table 1.16.

Table 1.16 Operations of CMOS NAND gate

A	B	Q_1	Q_2	Q_3	Q_4	V_O
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

1.18.3 CMOS NOR Gate

Figure 1.46 shows a two-inputs CMOS logic NOR gate. It consists of two *p*-channel and two *n*-channel MOSFETs. *n*-channel MOSFETs are connected in parallel and *p*-channel MOSFETs are connected in series. Here, Q_1 and Q_2 are *p*-channel MOSFETs and Q_3 and Q_4 are *n*-channel MOSFETs. When the input is low, *p*-channel MOSFETs are ON and *n*-channel MOSFETs are OFF. When the input is high, *p*-channel is OFF and *n*-channel is ON.

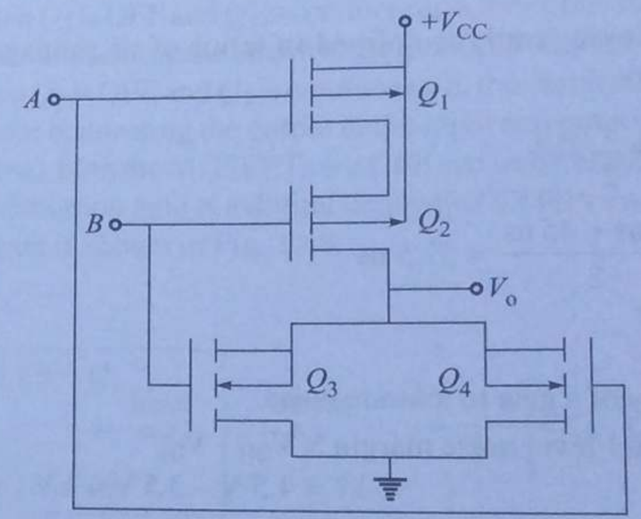


Fig. 1.46 CMOS NOR gate

Operation

- When inputs are low, Q_1 and Q_2 are ON, Q_3 and Q_4 are OFF, and the output is high (V_{DD}).
- When any one of the inputs is low (0 V or –ve), then the corresponding MOSFET Q_1 or Q_2 is ON, Q_3 or Q_4 is ON, and the output is low.
- When inputs are high (+ve voltage), Q_1 and Q_2 are OFF, Q_3 and Q_4 are ON, and the output is low.

Table 1.17 Operations of CMOS NOR gate

A	B	Q_1	Q_2	Q_3	Q_4	V_O
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	0
1	0	OFF	ON	ON	OFF	0
1	1	OFF	OFF	ON	ON	0

1.18.4 Characteristics of CMOS

The 54C/74C series is the commonly used CMOS series of ICs. The electrical characteristics of 54C/74C CMOS logic family are given in Table 1.18 for a supply voltage of 5 V.

Table 1.18 Electrical characteristics of 54C/74C CMOS logic family

Parameter	Value	Parameter	Value	Parameter	Value
V_{IH}	3.5 V	I_{IH}	1 μA	t_{PHL}	60 ns
V_{IL}	1.5 V	I_{IL}	−1 μA	t_{PHL}	45 ns
V_{OH}	4.5 V	I_{OH}	−100 μA	P.D.	10 mW
V_{OL}	0.5 V	I_{OL}	−360 μA		

Operation speed

The speed of a logic family is defined in terms of its propagation delay. The propagation delay,

$$t_p = \frac{t_{PHL} + t_{PLH}}{2}$$

$$= \frac{60\text{ ns} + 45\text{ ns}}{2} = 52.5\text{ ns}$$

Noise margin

It is the capability of a gate to tolerate noise.

Logic 1 level noise margin = $V_{OH} - V_{IH}$

$\Delta 1 = 4.5\text{ V} - 3.5\text{ V} = 1\text{ V}$

Logic 0 level noise margin = $V_{IL} - V_{OL}$

$\Delta 0 = 1.5\text{ V} - 0.5\text{ V} = 1\text{ V}$

Fan-out

MOS devices have a very high input impedance; therefore, the fan-out is large. Fan-out of a CMOS is 50 for low frequency and less than 50 for high frequency inputs.

Power dissipation

It is a function of current supply by the source. The current drawn from the supply in CMOS logic is less. Hence the power consumption in MOS circuit is less. Power dissipation for CMOS logic family is 1 mW at 1 MHz. It is less than 1 mW for frequencies less than 1 MHz.

Unused inputs

The IC of a logic family may have more than one gate. CMOS inputs have to be connected with a fixed voltage level or to another input. If inputs of unused CMOS gates are open, they are susceptible to noise and static charge that could bias both *p* and *n*-channel MOSFETs in the conductive state and power dissipation is increased.

1.18.5 Buffered and Unbuffered Gates

CMOS circuits are available in two versions, namely (*i*) CMOS with buffered output and (ii) CMOS with unbuffered output.

We have discussed unbuffered outputs of CMOS inverter, CMOS NAND gate, and CMOS NOR gate. To...

the output, the propagation delay of the gate is increased and it reduces the speed of operation.

1.18.6 Transmission Gates

Figure 1.47 shows the circuit of a transmission gate where Q_1 is PMOS and Q_2 is NMOS. Gates of Q_1 and Q_2 are controlled by the controlled inputs C and $\overline{C}$, respectively.

When $C = 1$ (high), Q_1 and Q_2 are ON or OFF depending upon the input A. When input A is high, then Q_2 is OFF and Q_1 is conducting in the ohmic region; Q_1 behaves as a small resistance connecting the output to the input and output $Y = A =$ high. When input A is low, then Q_1 is OFF and Q_2 is conducting in the ohmic region; Q_2 behaves as a small resistance connecting the output to the input and output $Y = A =$ low.

When $C = 0$ (low), both the MOSFETs are OFF and transmission is not possible.

In short, a transmission gate is a digital controlled CMOS switch. The symbol of transmission gate is shown in Fig. 1.48.

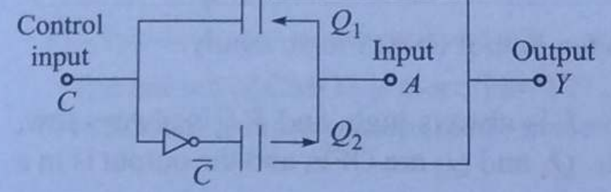


Fig. 1.47 Circuit of transmission gate

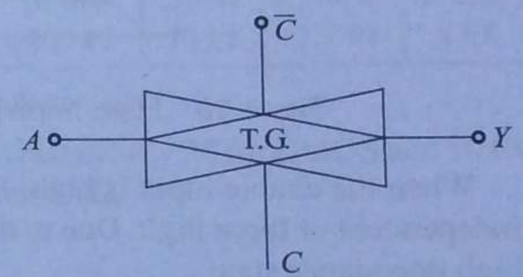


Fig. 1.48 The symbol of transmission gate

1.18.7 Open Drain Outputs

Different outputs are available in CMOS logic family as in the TTL family we had discussed earlier. In CMOS, open drain output is possible by replacing the transistor with a diode. See Fig. 1.49.

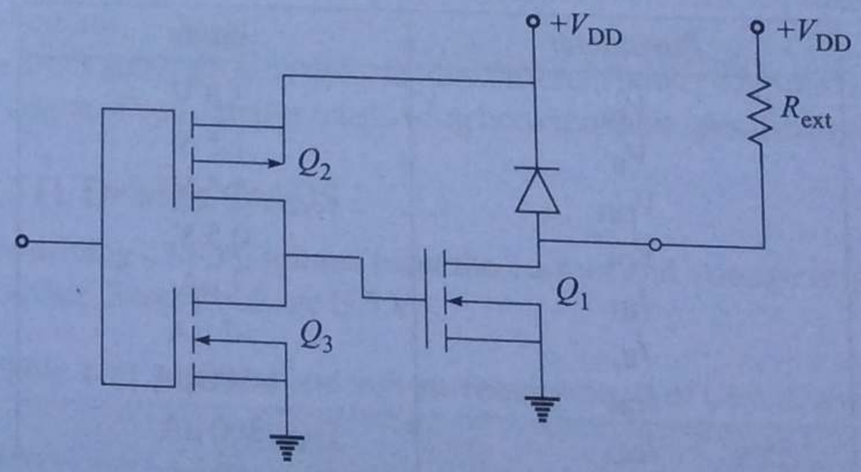


Fig. 1.49 Open drain outputs

The diode D_1 provides protection from electrostatic discharge. As in TTL, an external pull-up resistor is required to take the output. The open drain outputs CMOS

1.18.8 High Impedance Outputs

The high impedance output CMOS logic family is similar to the tri-state output in TTL family. Figure 1.50 shows the high impedance output CMOS logic family.

The enable input E is active low input. When E is low, Q_1 will be ON as the input to Q_1 is $(E + I)$, and I (input) is low. Now at the same time, Q_2 is off. Therefore the output is high. When the enable input is low and the input is high, $(E + I)$ and $\overline{E}I$ inputs to Q_1 and Q_2, respectively, are high. So Q_1 will be OFF and Q_2 will be ON.

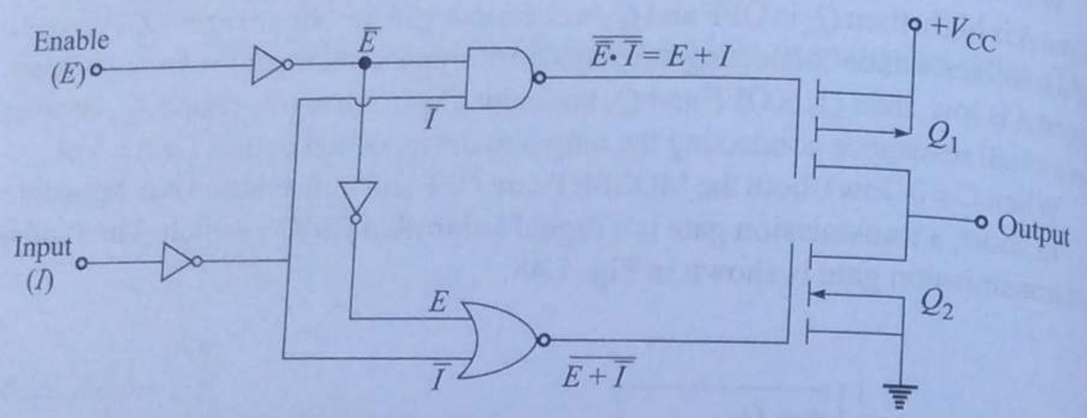


Fig. 1.50 High impedance output CMOS logic family

When the enable input is high, $E + I$ is always high, and $\overline{E}I$ is always low, independent of input high. Due to this, Q_1 and Q_2 are OFF, and the output is in a high impedance state.

1.18.9 Specifications and Standards

The EIA [Electronic Industries Association] has established certain standard specifications for CMOS circuits. In standards to differentiate buffered CMOS, it is identified as B-types and unbuffered are identified as VB types.

Table 1.19 Electrical characteristics

(Supply voltage V_{DD} = 5 V)

Parameter	*Value*
V_{IH}	3.5 V
V_{IL}	1.5 V
V_{OH}	4.5 V
V_{OL}	0.5 V
I_{IH}	1 μA
I_{IL}	–1 μA
I_{OH}	–100 μA
I_{OL}	360 μA
t_{PHL}	60 ns
t_{PLH}	45 ns
Static power dissipation per gate	10 mW

1.19 COMPARISON OF CMOS AND TTL FAMILIES

The CMOS and TTL families are compared and the comparison is given in Table 1.20.

Table 1.20 Comparison of CMOS and TTL families

Parameter	*CMOS*		*TTL*			
	Silicon gateCMOS	*Metal gate CMOS*	*74*	*74L*	*74AS*	*74ALS*
V_{IH}(min)	3.5 V	3.5 V	2.0 V	2.0 V	2.0 V	2.0 V
V_{IL}(max)	1.0 V	1.5 V	0.8 V	0.8 V	0.8 V	0.8 V
V_{OH}(min)	4.9 V	4.95 V	2.4 V	2.7 V	2.7 V	2.7 V
V_{OL}(max)	0.1 V	0.05 V	0.4 V	0.5 V	0.5 V	0.4 V
V_{NH}	1.4 V	1.45 V	0.4 V	0.7 V	0.7 V	0.7 V
V_{NL}	0.9 V	1.45 V	0.4 V	0.3 V	0.3 V	0.4 V
t_{pd}	8 ns	105 ns	10 ns	10 ns	1.5 ns	4 ns
P.D. (per gate)	0.17 mW	0.1 mW	10 mW	2 mW	8.5 mW	1 mW
$I_{pd} \infty$ P.D.	1.4 PJ	10.5 PJ	100 PJ	20 PJ	12.8 PJ	4 PJ

PJ – Pico-Joule, ns – nanosecond, and mW – milliWatts

The fan-out of CMOS is more than TTL. It is typically 50 for CMOS and 10 for TTL. CMOS is more susceptible to noise than TTL.

1.20 INTERFACING CMOS AND TTL DEVICES

Interfacing means connecting two different systems or devices, having different electrical characteristics. In such a case, direct connection is not possible. The circuit used to connect driver and load circuits is called as the *interface circuit*. The interface circuit takes the input from the driver and converts it, so that it is compatible with the requirements of the load.

Following are the important factors to be considered:

- The driver output must satisfy the voltage and current requirements of the load circuit.
- The driver and load circuit may require different power supplies. In such cases, the output of both circuits must swing between their specified voltage ranges.

1.20.1 TTL Driving CMOS

As TTL is driving CMOS, it must meet the current and voltage requirements of the load device. Supply voltage is 5 V.

Table 1.21 Current and voltage requirements of CMOS and TTL

Parameter	*CMOS*		*TTL*		
	4000 B	*74HCT*	*74*	*74L*	*74AS*
I_{IH} (min)	1 μA	1 μA	40 μA	20 μA	200 μA
I_{IL} (max)	1 μA	1 μA	1.6 mA	0.4 mA	2 mA
I_{OH} (min)	0.4 mA	4 mA	0.4 mA	0.4 mA	2 mA

From Table 1.21, it is clear that the TTL has no problem to drive the CMOS, as the input current requirement of CMOS is very low as compared to the output current capabilities of TTL.

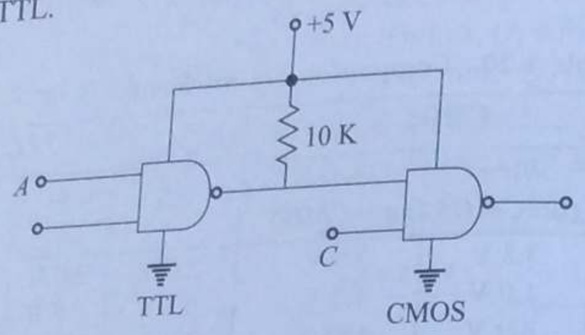


Fig. 1.51 TTL driving CMOS

1.20.2 CMOS Driving TTL

It is necessary to check the CMOS output capacity and the TTL input requirements. Table 1.22 compares the two.

Table 1.22 CMOS output capacity and TTL input requirements

CMOS (4000 B)	TTL
V_{OH} (min) = 4.95 V	V_{IH} (min) = 2.0 V
V_{OL} (max) = 0.05 V	V_{IL} (max) = 0.8 V
I_{OH} (min) = 0.4 mA	I_{IH} (min) = 40 μA
I_{OL} (max) = 0.4 mA	I_{IL} (max) = 1.6 mA

* μA – microampere and mA – milliampere

From the above table, it is proved that the CMOS driving a TTL in the high state do not need any special consideration such as,

V_{OH} (CMOS) > V_{IH} (TTL)

I_{OH} (CMOS) > I_{IH} (TTL)

For the CMOS driving a TTL in low state, see the parameter

V_{OL} (max) CMOS < V_{IL} (max) TTL

This satisfies the requirement in the low state. Now see the current parameter.

I_{OL} (CMOS) must be greater than I_{IL} (max) for TTL.

I_{OL} (CMOS) < I_{IL} (max)

This does not satisfy the condition. In this situation, some buffer circuit is required to be connected between the two. Figure 1.52 shows what it means.

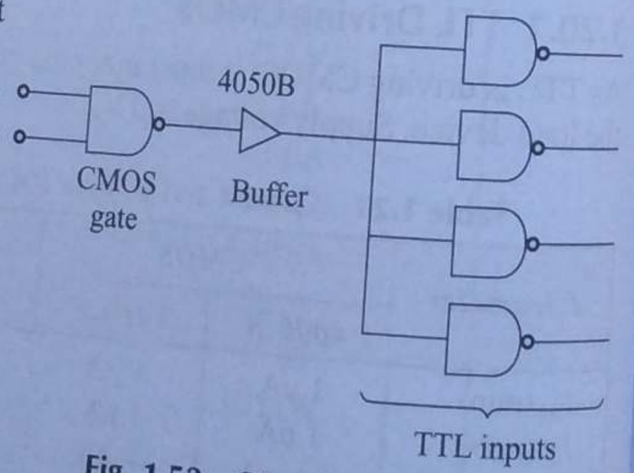


Fig. 1.52 CMOS driving TTL

The above discussion of CMOS and TTL interfacing is only for standard CMOS and standard TTL logic families. For other families, like high speed, low power, it is necessary to compare the output capabilities of driver and the input requirements of load on the same lines as above.

1.21 INTERFACING ECL AND TTL DEVICES

Interfacing means connecting two different systems or devices, having different electrical characteristics. In such a case, direct connection is not possible. The circuit used to connect driver and load circuits is called as the *interface circuit*. The interface circuit takes the input from the driver and converts it, so that it is compatible with the requirements of the load.

1.21.1 TTL Driving ECL

As TTL is driving ECL, it must meet the current and voltage requirements of the load device. A TTL cannot interface directly with an ECL; it requires a translator. The MC10H124 is a TTL to ECL translator. The logic diagram of MC10H124 is shown in Fig. 1.53; it is a 16-pin IC and it uses two power supplies.

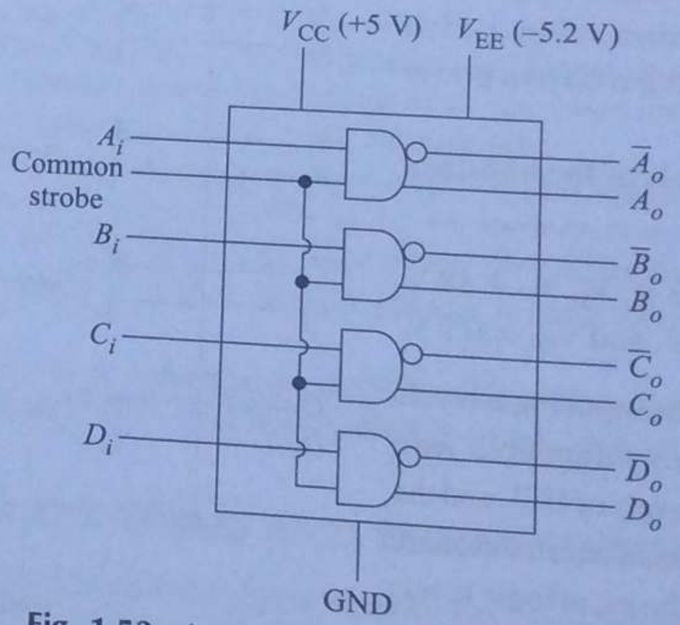


Fig. 1.53 Logic diagram of MC10H124

The logic levels of the translator are:

V_{IH} = 2 V, V_{IL} = 0.8 V, V_{OH} = –0.98 V and V_{OL} = –1.63 V

The logic levels of the TTL are: V_{OH} = 2.4 V and V_{OL} = 0.4 V

The logic levels of the ECL are: V_{IH} = –1.4 V and V_{IH} = –1.2 V

It is observed that

V_{IH} (Translator) < V_{OH} (TTL), V_{IL}(Translator) > V_{OL} (TTL)

and V_{IH} (ECL) < V_{OH} (Translator), V_{IL}(ECL) > V_{OL} (Translator).

It shows that the input logic levels of a translator are compatible with the [illegible] logic levels of a TTL and the output logic levels of a translator are compatible [illegible] the input logic levels of an ECL. Figure 1.54 shows the TTL gate driving an [illegible] gate.

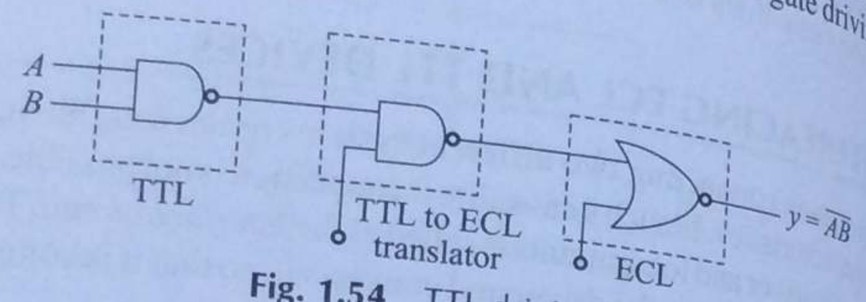


Fig. 1.54 TTL driving ECL

1.21.2 ECL Driving TTL

An ECL cannot interface directly with a TTL; it requires a translator. The MC10H125 is an ECL to TTL translator. The logic diagram of MC10H124 is shown in Fig. 1.55; it is a 16-pin IC and it uses two power supplies.

The logic levels of the translator are:

$$V_{IH} = -1.13\text{ V},\ V_{IL} = -1.48\text{ V},$$
$$V_{OH} = 2.5\text{ V, and } V_{OL} = 0.5\text{ V}$$

It shows that the input logic levels of a translator are compatible with the output logic levels of ECL and the output logic levels of a translator are compatible with the input logic levels of a TTL. Figure 1.56 shows the ECL gate driving a TTL gate.

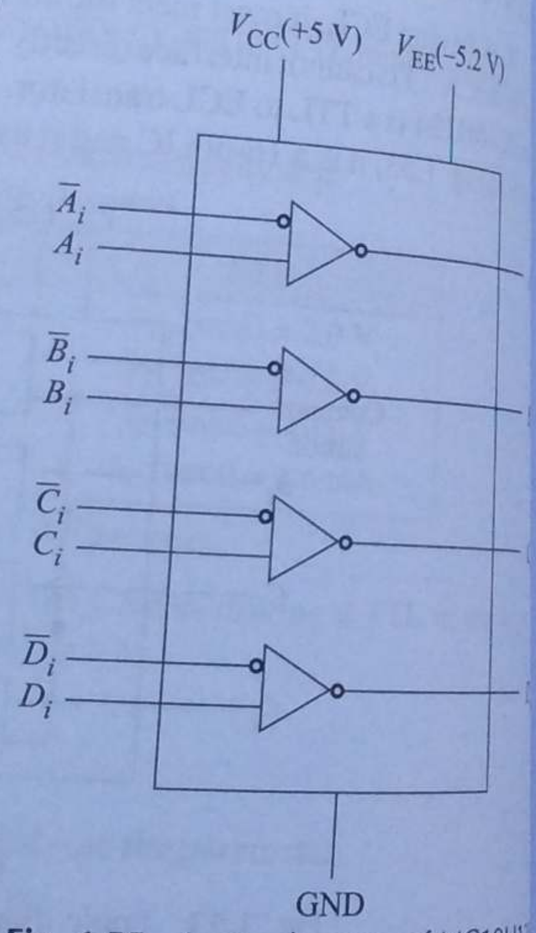
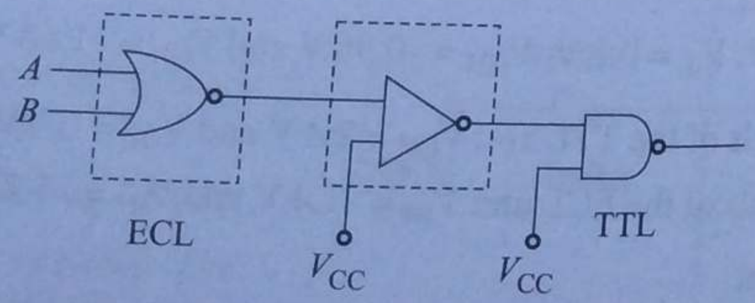


Fig. 1.55 Logic diagram of MC10H1[illegible]



ECL to TTL translated

SUMMARY

- Modern digital systems use integrated circuits because of the following advantages:
  1. Overall size of digital system gets reduced, because ICs consist of a large number of digital circuits.
  2. The cost of ICs is very low.
  3. Power consumption is less.
  4. They have high noise margin.
  5. They have high reliability.
  6. The operating speed is high.
- The logic families are classified into two types: (a) Bipolar logic families, and (b) Unipolar logic families.
- Bipolar logic families are further classified as: (a) Saturated bipolar logic families, and (b) Unsaturated bipolar logic families.
- The examples of saturated bipolar logic families are: (a) Resistor-transistor logic, (b) Direct coupled transistor logic, (c) Integrated injection logic, (d) Diode-transistor logic, (e) High threshold logic, and (f) Transistor-transistor logic.
- The examples of unsaturated bipolar logic families are: (a) Schottky transistor-transistor logic, and (b) Emitter-coupled logic.
- The examples of unipolar families are: (a) PMOS, (b) NMOS, and (c) CMOS.
- The various parameters of digital ICs used to compare their performance are: (a) Speed of operation, (b) Power dissipation, (c) Figure of merit, (d) Fan-out, (e) Fan-in, (f) Current and voltage parameters, (g) Noise immunity, (h) Power supply requirements, and (i) Operating temperature.
- The RTL family is no more used because of: (a) Low noise margin (typically 0.1 V), (b) Fan-out is poor (typically 5), (c) Propagation delay is high and speed of operation is low (12 ns), and (d) High power dissipation (typically 12 mW).
- The DCTL is simple than RTL, it is not popular because of the current hogging problem.
- A standard TTL can be classified as: (a) TTL with passive pull-up, (b) TTL with totem-pole output (c) TTL with open collector output, and (d) Tri-state TTL.

KEY TERMS AND DEFINITIONS

Integrated circuit Most of the digital circuits are constructed on a single chip, which are referred to as integrated circuits (IC).

Logic family The set of compatible ICs with the same logic levels and same supply voltages, fabricated to perform various logic functions, are known as logic family.

Unipolar logic family The logic family having unipolar devices like MOSFETs as its key element is referred to as unipolar logic family.

Bipolar logic family The logic family with transistors and diodes as its key element is referred to as bipolar logic family.

Resistor-transistor logic (RTL) The logic family that consists of resistors and transistors is called as resistor-transistor logic.

Direct coupled transistor logic (DCTL) The logic family in which [illegible]

Diode-transistor logic (DTL) The logic family with diodes and transistors is called as diode-transistor logic.

Transistor-transistor logic (TTL) The saturated logic family consisting transistor as its basic element, which operates either in cut-off or saturation region, is called transistor-transistor logic.

Emitter coupled logic (ECL) The transistors of emitter coupled logic is operated in cut-off or active region, it never goes in saturation and therefore the storage time is eliminated.

Integrated injection logic (I^2L) It uses only transistors for the construction of gate and hence, it becomes possible to integrate a large number of gates in a single package.

MOSFET logic MOSFETs are very popular for logic circuit because of their low power dissipation and high density of fabrication.

CMOS logic CMOS family uses *n*-channel and *p*-channel MOSFETs. In CMOS, *p*-channel and *n*-channel MOS devices are fabricated on the same chip, which makes its fabrication complicated but it reduces the packaging density. It has small power consumption.

EXERCISES

Review Questions

1. With respect to logic gate, explain the following terms: I_{OL}, I_{IL}, I_{OH}, I_{IH}. Specify the values of these parameters for a standard TTL gate. Define fan-out. How do you obtain fan-out from the above four parameters?
2. Explain CMOS inverter gate with a circuit diagram.
3. Explain the totem-pole output configuration for a TTL gate.
4. With the help of a neat diagram, explain the working of a two-inputs CMOS NAND gate. What is the advantage of active load?
5. Define the following parameters of digital IC families and give their typical values for TTL and CMOS families.
   (a) Propagation delay (b) Noise margin
   (c) Fan-out (d) Figure of merit
6. Explain the working of a two-inputs TTL NAND gate with the help of a circuit diagram.
7. Explain the following parameters:
   (a) Propagation delay (b) Fan-out
   (c) Noise margin (d) Power dissipation
8. Write a short note on CMOS logic family.
9. How is propagation delay improved in totem-pole TTL logic?
10. Give typical values of the following parameters for TTL and CMOS logic gates:
    (a) Propagation delay (b) Fan-out
    (c) Noise margin (d) Power dissipation
11. Describe the difference between current sinking and current sourcing in TTL logic.

12. Explain the working of basic TTL NAND gate with a neat diagram. Explain the following three types of output configuration.
    (a) Open collector output (b) Totem-pole output
    (c) Tri-state output
13. Explain the basic ECL OR/NOR gate with a neat circuit diagram. Why does the ECL family have the lowest propagation delay of all logic families?
14. Give typical values of the following parameters for CMOS logic gates:
    (a) Propagation delay (b) Fan-out
15. With a neat circuit diagram, explain the operation of a two-inputs CMOS NOR gate.
16. Explain interfacing of a TTL gate driving CMOS gates and vice versa.
17. Define the following parameters of logic families and give their typical values for a standard TTL gate:
    (a) Propagation delay (b) Fan-out
    (c) Noise margin (d) Figure of merit
18. Explain with a neat circuit diagram the tri-state TTL gate. How can the tri-state circuit outputs be connected together to form a data bus so that each output can be switched onto the bus wire?
19. Draw neat circuit diagrams of:
    (a) Two-inputs TTL NAND gate with totem-pole output
    (b) Two-inputs CMOS NAND gate

Multiple Choice Questions

Select the correct option.

1. Positive logic in a logic circuit is one in which.....
   (a) logic 0 and 1 are represented by negative and positive voltages respectively
   (b) logic 0 voltage level is higher than logic 1 voltage level
   (c) logic 0 voltage level is lower than logic 1 voltage level
   (d) logic 0 and 1 are represented by 0 and positive voltages respectively
2. In standard TTL gates, the totem–pole output stage is primarily used to.....
   (a) increase the noise margin of the gate
   (b) decrease the output switching delay
   (c) facilitate a wired-OR logic connection
   (d) increase the output impedance of the circuit
3. The switching speed of ECL is very high, because.....
   (a) the transistors are switched between cut-off and saturation regions
   (b) the transistors are switched between active and saturation regions
   (c) the transistors are switched between cut-off and active regions
   (d) the transistors may operate in any of the three regions
4. In standard TTL, the totem-pole stage refers to.....
   (a) the multi-emitter input stage
   (b) the phase-splitter
   (c) the output buffer
   (d) the open collector output stage

5. The figure of merit of a logic family is given by.....
   (a) gain-bandwidth product
   (b) product of propagation delay time and power dissipation
   (c) product of fan-out and propagation delay time
   (d) product of noise margin and power dissipation
6. Which of the following uses the least power?
   (a) TTL (b) ECL
   (c) CMOS (d) all use same power
7. Which of the following logics is the fastest?
   (a) TTL (b) ECL
   (c) CMOS (d) LSI
8. In Schottky-clamped TTL, the purpose of Schottky diode is.....
   (a) to increase propagation delay
   (b) to achieve efficient non-saturated switching
   (c) to improve noise margin
   (d) to decrease dissipation

Answers

1. (d)	2. (b)	3. (c)	4. (c)
5. (b)	6. (c)	7. (b)	8. (b)

Chapter 2

NUMBER SYSTEM AND CODES

Chapter Outline

- Number systems and their interconversion
- Sign binary numbers
- Binary arithmetic
- 2's complement arithmetic
- BCD code and its arithmetic
- Excess-3 code and its arithmetic
- Gray code
- Parity code, linear block code, and hamming code

2.1 INTRODUCTION

In case of digital systems, data is represented by the binary digits 0 and 1 which are known as *bits*. A group of eight bits is known as *byte* and a group of four bits is known as *nibble*. In digital systems, the information is represented, stored and transmitted as a group of bits. This group of bits is known as *binary code*. In other words, the binary code is a binary representation of numbers, alphabets, special characters, and control functions. This chapter includes the study of different codes and their applications.

The digital system supports only binary data. In binary number data, HIGH is represented as logic 1 and LOW as logic 0 in positive logic; and in negative logic, HIGH is represented as logic 0 and LOW is represented as logic 1.

Whenever binary data is transmitted over different paths, there is a possibility of change in the data which leads to transmission of wrong information to a destination. To avoid the trasmission of incorrect information, error detecting and correcting codes are used.

2.2 NUMBER SYSTEMS

A number is a collection of symbols. Each symbol's value is a function of the type of the system and its position. Number systems are positional because the value of a symbol depends upon its position.

The number is represented as,

$$\underbrace{(S_{n-1}\, S_{n-2} \ldots S_1\, S_0}_{\text{Integer part}} \underbrace{\overset{\bullet}{\uparrow}}_{\text{Radix point}} \underbrace{S_{-1}\, S_{-2} \ldots S_{-m})_b}_{\text{Fractional part}} \qquad (2.1)$$

where b = radix or base of the number system
n = number of digits in the integer part
m = number of digits in fractional part
S_{n-1} = most significant digit (msb)
S_{-m} = least significant digit (lsb)

On the basis of number of symbols and radix, number systems are classified as (i) decimal number system, (ii) binary number system, (iii) octal number system and (iv) hexadecimal number system.

2.2.1 Decimal Number System

In the decimal number system, the total number of symbols are ten (0, 1, 2, 3, 4, 5, 6, 7, 8, 9); base is 10, and the radix point is known as the decimal point. The digits at the left side of the decimal point constitute the integer part and the digits at the right side of the decimal point constitute the fractional part. The value of a digit is the function of its position in a number. The leftmost digit has the maximum value, hence it is known as the most significant digit (MSD). The rightmost digit has the minimum value and hence it is known as the least significant digit (LSD).

For example, $(555.55)_{10} = 5\times10^2 + 5\times10^1 + 5\times10^0 + 5\times10^{-1} + 5\times10^{-2}$

2.2.2 Binary Number System

In the binary number system, the total number of symbols are two (0, 1); base is 2 and the radix point is known as the binary point. These symbols are known as bits (binary digits). It is a positional number system; the weight of a bit is defined by its position with base 2.

For example, $(1111.111)_2 = 1\times2^3 + 1\times2^2 + 1\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2} + 1\times2^{-3}$

2.2.3 Hexadecimal Number System

The number system with base 16 is known as the hexadecimal number system. In the hexadecimal number system, 16 symbols are used to represent the numbers and these symbols are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. The equivalent decimal numbers of the hexadecimal symbols are given in Table 2.1.

Table 2.1 Hexadecimal symbols and equivalent decimal digit

Hexa-decimal	*Decimal*	*Hexa-decimal*	*Decimal*	*Hexa-decimal*	*Decimal*	*Hexa-decimal*	*Decimal*
0	0	4	4	8	8	C	12
1	1	5	5	9	9	D	13
2	2	6	6	A	10	E	14
3	3	7	7	B	11	F	15

In the hexadecimal number system, the radix point is known as the hexadecimal point. It is a positional number system; the weight of a digit is defined by its position with base 16.

For example, $(215A.268)_{16} = 2\times16^3 + 1\times16^2 + 5\times16^1 + 10\times16^0 + 2\times16^{-1} + 6\times16^{-2} + 8\times16^{-3}$

This number system is used to pass the data to the computer system by using 16 keys. It is the most commonly used number system in a microprocessor kit. The keyboard of the microprocessor kit is known as the HEX keyboard.

2.2.4 Octal Number System

The number system with base 8 is known as the octal number system. Eight symbols are used to represent numbers in this system, and these are: 0, 1, 2, 3, 4, 5, 6, and 7. In the octal number system, the radix point is known as the octal point.

It is also a positional number system; the weight of a digit is defined by its position with base 8.

For example, $(576.217)_8 = 5\times8^2 + 7\times8^1 + 6\times8^0 + 2\times8^{-1} + 1\times8^{-2} + 7\times8^{-3}$

2.3 INTERCONVERSION OF NUMBERS

Computer systems process binary data. The information given by the user may be in the form of decimal number, hexadecimal number, or octal number. So an understanding of the system operation requires the ability to convert the numbers from one number system to another.

2.3.1 Binary to Decimal Number Conversion

The binary number system is also a positional system in which each binary bit carries a certain weight based on its position relative to the binary point. The binary number can be converted to its decimal equivalent by summing the weights of various positions in the binary number which contain a symbol 1. The generalized method for the conversion of a binary number to its decimal equivalent is explained below.

Consider the binary number: $(B_4\, B_3\, B_2\, B_1\, B_0 \bullet B_{-1}\, B_{-2}\, B_{-3})_2$ (2.2)

The first left bit from the binary point is denoted by B_0. Subscript 0 defines the position of the bit. The value of B_0 in decimal is $B_0\times2^0$. Similarly the value of B_1 is $B_1\times2^1$ and so on. The first right bit from binary point is denoted by B_{-1}. The value of B_{-1} is $B_{-1}\times2^{-1}$. The decimal equivalent of the binary number is the sum of all these bit values.

Decimal equivalent of $(B_4\, B_3\, B_2\, B_1\, B_0 \bullet B_{-1}\, B_{-2}\, B_{-3})_2$
$= B_4\times2^4 + B_3\times2^3 + B_2\times2^2 + B_1\times2^1 + B_0\times2^0 + B_{-1}\times2^{-1} + B_{-2}\times2^{-2} + B_{-3}\times2^{-3}$

Examples Let us compute the decimal equivalents of the following binary numbers.

(i) $(11011)_2 = 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$
$= 16 + 8 + 2 + 1 = (27)_{10}$

(ii) $(0.1101)_2 = 1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 1\times2^{-4}$

$$= \frac{1}{2} + \frac{1}{4} + 0 + \frac{1}{16}$$

$$= 0.5 + 0.25 + 0.0625$$

$$= (0.8125)_{10}$$

(iii) $(1101.1101)_2 = 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 1\times2^{-4}$

$$= 8 + 4 + 0 + 1 + 0.5 + 0.25 + 0 + 0.0625$$

$$= (13.8125)_{10}$$

2.3.2 Decimal to Binary Conversion

There are two methods to convert a decimal number to its equivalent binary number, namely (i) sum of power method, and (ii) repeated division and multiplication method.

Sum of power method

In the sum of power method, the decimal number is expressed as a sum of powers of 2 and then 1s and 0s are written in the appropriate bit positions. It is just the opposite of binary to decimal conversion.

Examples

(i) $(26)_{10} = 16 + 8 + 0 + 2 + 0$

$$= 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0$$

$$= (11010)_2$$

(ii) $(27)_{10} = 16 + 8 + 0 + 2 + 1$

$$= 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$$

$$= (11011)_2$$

(iii) $(0.75)_{10} = 0.5 + 0.25$

$$= \frac{1}{2} + \frac{1}{4}$$

$$= 1\times2^{-1} + 1\times2^{-2}$$

$$= (0.11)_2$$

(iv) $(21.625)_{10} = 16 + 0 + 4 + 0 + 1 + 0.5 + 0 + 0.125$

$$= 1\times2^4 + 0\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$$

$$= (10101.101)_2$$

This method is not commonly used, as it needs the number to be represented as a sum of numbers that are in power of 2 form. It may be easy for a few numbers, but it is difficult for larger numbers.

Repeated division and multiplication method

The integer and fractional parts of a decimal number are treated separately for the conversion. The repetitive division method is used for the integer part of decimal numbers and the repetitive multiplication method is used for the fractional part of decimal numbers.

(a) Repeated division (for integer decimal number)

In the repeated division method, the decimal number is divided by 2 and the remainder is found after each division, until the quotient 0 is obtained. Note that the binary result is obtained by writing the first remainder as the least significant bit and the last remainder as the most significant bit.

For example, $(25)_{10} = (?)_2$

	Quotient	*Remainder*	
$25 \div 2$	12	1	♦ LSB
$12 \div 2$	6	0	
$6 \div 2$	3	0	
$3 \div 2$	1	1	
$1 \div 2$	0	1	♦ MSB

Ans: $(25)_{10} = (11001)_2$

(b) Repeated multiplication (for fractional decimal number)

In the repeated multiplication method, the fractional part of the decimal number is multiplied by 2; the integer part of the multiplication is found after each multiplication operation, until the fractional part of a decimal number becomes zero. The fractional part of the binary result is obtained by writing the first bit as the MSB and the last bit as the LSB.

For example, $(0.625)_{10} = (?)_2$

Multiplication	*Integer part*	
$0.625 \times 2 = 1.25$	1	♦ MSB
$0.25 \times 2 = 0.5$	0	
$0.5 \times 2 = 1.0$	1	♦ LSB

Ans: $(0.625)_{10} = (0.101)_2$

Now, let us consider a decimal number that includes both an integer part as well as a fractional part. Such a decimal number is converted to its binary equivalent by using separate treatments over the integer part and the fractional part of the decimal number. The binary representation of such a decimal number is the sum of binary representation of the integer decimal number and the fractional decimal number.

For example, consider $(57.825)_{10}$. The integer part of this decimal number is 57 and the fractional part is 0.825.

Conversion of integer part:

	Quotient	*Remainder*	
$57 \div 2$	28	1	♦ LSB
$28 \div 2$	14	0	
$14 \div 2$	7	0	
$7 \div 2$	3	1	
$3 \div 2$	1	1	
$1 \div 2$	0	1	♦ MSB

$(57)_{10} = (111001)_2$

Conversion of fractional part:

Multiplication	*Integer part*	
$0.825 \times 2 = 1.650$	1	♦ MSB
$0.650 \times 2 = 1.30$	1	
$0.30 \times 2 = 0.6$	0	
$0.6 \times 2 = 1.2$	1	♦ LSB

$(0.825)_{10} = (0.1101)_2$

Ans: $(57.825)_{10} = (111001)_2 + (0.1101)_2 = (111001.1101)_2$

Example 2.1 Convert the following binary numbers to their decimal equivalents:
(a) 10010111 (b) 10111.0110 (c) 101111.11

Solution

(a) $(10010111)_2 = 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$= 128 + 16 + 4 + 2 + 1$

$= (151)_{10}$

(b) $(10111.0110)_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4}$

$= 16 + 4 + 2 + 1 + \frac{1}{4} + \frac{1}{8}$

$= (23.375)_{10}$

(c) $(101111.11)_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$

$= 32 + 8 + 4 + 2 + 1 + \frac{1}{2} + \frac{1}{4}$

$= (47.75)_{10}$

Example 2.2 Convert the following decimal numbers to their binary equivalents:
(a) $(83)_{10}$ (b) $(79.515)_{10}$ (c) $(109.125)_{10}$

Solution

(a) $(83)_{10}$

	Quotient	*Remainder*	
$83 \div 2$	41	1	♦ LSB
$41 \div 2$	20	1	
$20 \div 2$	10	0	
$10 \div 2$	5	0	
$5 \div 2$	2	1	
$2 \div 2$	1	0	
$1 \div 2$	0	1	♦ MSB

Ans: $(83)_{10} = (1010011)_2$

(b) $(79.515)_{10}$

The integer part of the given decimal number is 79.

	Quotient	*Remainder*	
$79 \div 2$	39	1	♦ LSB
$39 \div 2$	19	1	
$19 \div 2$	9	1	
$9 \div 2$	4	1	
$4 \div 2$	2	0	
$2 \div 2$	1	0	
$1 \div 2$	0	1	♦ MSB

$(79)_{10} = (1001111)_2$

The fractional part of the given decimal number is 0.515.

Multiplication	*Integer part*	
$0.515 \times 2 = 1.030$	1	♦ MSB
$0.03 \times 2 = 0.060$	0	
$0.06 \times 2 = 0.12$	0	
$0.12 \times 2 = 0.24$	0	
$0.24 \times 2 = 0.48$	0	
$0.48 \times 2 = 0.96$	0	
$0.96 \times 2 = 1.96$	1	♦ LSB

$(0.515)_{10} = (0.1000001)_2$

Ans: $(79.515)_{10} = (1001111)_2 + (0.1000001)_2 = (1001111.1000001)_2$

(c) $(109.125)_{10}$

The integer part of the given decimal number is 109.

	Quotient	*Remainder*	
$109 \div 2$	54	1	♦ LSB
$54 \div 2$	27	0	
$27 \div 2$	13	1	
$13 \div 2$	6	1	
$6 \div 2$	3	0	
$3 \div 2$	1	1	
$1 \div 2$	0	1	♦ MSB

$(109)_{10} = (1101101)_2$

The fractional part of the given number is 0.125.

Multiplication	*Integer part*	
$0.125 \times 2 = 0.250$	0	♦ MSB
$0.25 \times 2 = 0.50$	0	
$0.5 \times 2 = 1.0$	1	♦ LSB

$(0.125)_{10} = (0.001)_2$

Ans: $(109.125)_{10} = (1101101)_2 + (0.001)_2 = (1101101.001)_2$

2.3.3 Octal to Decimal Conversion

The octal number system is also a positional number system, where each octal digit carries a certain weight based on its position relative to the octal point. The octal number can be converted to its decimal equivalent by summing the weights of various positions. Generalized method for the conversion of an octal number to its decimal equivalent is explained below.

Consider the octal number: $O_4O_3O_2O_1O_0 \cdot O_{-1}O_{-2}O_{-3}$

The first left digit from the octal point is denoted by O_0. Subscript 0 defines the position of the digit. The value of O_0 in decimal is $O_0 \times 8^0$. Similarly the value of O_1 is $O_1 \times 8^1$ and so on. The first right digit from the octal point is denoted by O_{-1}. The value of O_{-1} is $O_{-1} \times 8^{-1}$. The decimal number of the octal number is the sum of all these digit values.

$$\text{Decimal} = O_4 \times 8^4 + O_3 \times 8^3 + O_2 \times 8^2 + O_1 \times 8^1 + O_0 \times 8^0 + O_{-1} \times 8^{-1} + O_{-2} \times 8^{-2} + O_{-3} \times 8^{-3}$$

Examples

(i) $(753)_8 = 7 \times 8^2 + 5 \times 8^1 + 3 \times 8^0$
$= 448 + 40 + 3$
$= (491)_{10}$

(ii) $(0.235)_8 = 2 \times 8^{-1} + 3 \times 8^{-2} + 5 \times 8^{-3}$
$= 0.25 + 0.0468 + 0.00976$
$= (0.30664)_{10}$

(iii) $(265.731)_8 = 2 \times 8^2 + 6 \times 8^1 + 5 \times 8^0 + 7 \times 8^{-1} + 3 \times 8^{-2} + 1 \times 8^{-3}$
$= 128 + 48 + 5 + 0.875 + 0.0458 + 0.00195$
$= (181.92375)_{10}$

2.3.4 Decimal to Octal Conversion

The integer and fractional parts of a decimal number are treated separately for the conversion.

(a) Integer number

The integer part of a decimal number can be converted to octal by using the repeated division method. The integer part of the decimal number is divided by 8, and the remainder is found after each division until the quotient becomes zero. The octal number is obtained by writing the first remainder as the LSD and the last remainder as the MSD.

For example, $(266)_{10} = (?)_8$

	Quotient	*Remainder*	
$266 \div 8$	33	2	♦ LSD
$33 \div 8$	4	1	
$4 \div 8$	0	4	♦ MSD

Ans: $(266)_{10} = (412)_8$

(b) Fractional number

The fractional part of a decimal number can be converted to its octal equivalent by using the repeated multiplication method. The fractional part of the decimal number is multiplied by 8, and the integer part after each multiplication operation is found until the fractional part of the decimal number becomes zero. The fractional part of the octal result is obtained by writing the first integer as the MSD and the last integer as the LSD.

For example: $(0.256)_{10}$

Multiplication	*Integer part*	
$0.256 \times 8 = 2.048$	2	♦ MSD
$0.048 \times 8 = 0.384$	0	
$0.384 \times 8 = 3.072$	3	
$0.072 \times 8 = 0.576$	0	
$0.576 \times 8 = 4.608$	4	♦ LSD

Ans: $(0.256)_{10} = (0.20304)_8$

The decimal number that includes an integer part as well as a fractional part is converted to the octal number by using separate treatments over the integer part and the fractional part of the decimal number. The octal representation of such a decimal number is the sum of octal representation of the integer decimal number and the fractional decimal number.

For example, $(125.201)_{10} = (?)_8$

Conversion of integer part

	Quotient	*Remainder*	
$125 \div 8$	15	5	♦ LSD
$15 \div 8$	1	7	
$1 \div 8$	0	1	♦ MSD

$(125)_{10} = (175)_8$

Conversion of fractional part

Multiplication	*Integer part*	
$0.201 \times 8 = 1.608$	1	♦ MSD
$0.608 \times 8 = 4.864$	4	
$0.864 \times 8 = 6.912$	6	
$0.912 \times 8 = 7.296$	7	
$0.296 \times 8 = 2.368$	2	♦ LSD

$(0.201)_{10} = (0.14672)_8$

Ans: $(125.201)_{10} = (175)_8 + (0.14672)_8 = (175.14672)_8$

2.3.5 Octal to Binary Conversion

An octal number is also known as a 3-bit binary number, because each digit of an octal number is represented by three binary digits as given in Table 2.2.

Table 2.2 Octal digit and its binary equivalent

Octal	*Binary*
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Converting each octal digit to its 3-bit binary equivalent performs the conversion from octal to binary.

Examples

(i) $(452)_8 = (?)_2$

$4 = 100, \quad 5 = 101, \quad 2 = 010$

Ans: $(452)_8 = (100101010)_2$

(ii) $(0.275)_8 = (?)_2$

$2 = 010, \quad 7 = 111, \quad 5 = 101$

Ans: $(0.275)_8 = (0.010111101)_2$

(iii) $(746.421)_8 = (?)_2$

$7 = 111, \quad 4 = 100, \quad 6 = 110, \quad 4 = 100, \quad 2 = 010, \quad 1 = 001$

Ans: $(746.421)_8 = (111100110.100010001)_2$

2.3.6 Binary to Octal Conversion

There are different methods used to convert the integer and fractional parts of binary numbers into their octal equivalent.

(a) Integer part

The integer binary numbers can be converted into their octal equivalent by making groups of three bits starting from LSB and moving towards MSB and then replacing each group of three bits by its equivalent octal representation. If the number of bits is less than three in the last group, then the required number of zeros are added to the left side.

For example,

$(100011110)_2 = 100\ 011\ 110$

$= (436)_8$

(b) Fractional part

The fractional binary numbers can be converted into their equivalent octal numbers by making groups of three bits starting from MSB and moving towards LSB; and then replacing each group of three bits by its equivalent octal representation. If the number of bits is less than three in the last group, then the required number of zeros are added to the right side.

For example,

$(0.010111101)_2 = 0\ .\ 010\ 111\ 101$

$= (0.275)_8$

Now, let us consider a binary number with both integer and fractional parts. The integer and fractional parts of a binary number are treated separately for the conversion. The octal representation of such a binary number is the sum of octal representation of the integer binary number and the fractional binary number. For example, $(10010111.0110101)_2 = (?)_8$

Integer part $= (10010111)$

$= 10\ 010\ 111$

$= 010\ 010\ 111$ (by adding 0 to the left)

$= 227 = (227)_8$

Fractional part $= (0.0110101)$

$= 0.011\ 010\ 1$

$= 0.011\ 010\ 100$ (by adding 0 0 to the right)

$= 0.324 = (0.324)_8$

Ans: $(10010111.0110101)_2 = (227)_8 + (0.324)_8 = (227.324)_8$

Example 2.3 Convert the following decimal numbers to their octal equivalents:

(a) $(99)_{10}$ (b) $(66.38)_{10}$

Solution

(a) Given number is $(99)_{10}$

	Quotient	*Remainder*	
$99 \div 8$	12	3	♦ LSD
$12 \div 8$	1	4	
$1 \div 8$	0	1	♦ MSD

Ans: $(99)_{10} = (143)_8$

(b) Given number is $(66.38)_{10}$

The integer part of the number $(66.38)_{10} = 66$ and the fractional part is 0.38

Conversion of integer part

	Quotient	*Remainder*	
$66 \div 8$	8	2	♦ LSD
$8 \div 8$	1	0	
$1 \div 8$	0	1	♦ MSD

Ans: $(66)_{10} = (102)_8$

Conversion of fractional part

Multiplication	*Integer part*	
$0.38 \times 8 = 3.04$	3	♦ MSD
$0.04 \times 8 = 0.32$	0	
$0.32 \times 8 = 2.56$	2	
$0.56 \times 8 = 4.48$	4	

$0.48 \times 8 = 3.84$ 3

$0.84 \times 8 = 6.72$ 6 ♦ LSD

Ans: $(0.38)_{10} = (0.302436)_8$

Ans: $(66.38)_{10} = (102)_8 + (0.302436)_8 = (102.302436)_8$

Example 2.4 Convert the following octal numbers to their decimal equivalents:
(a) $(762)_8$ (b) $(0.231)_8$ (c) $(762.231)_8$

Solution

(a) $(762)_8 = 7 \times 8^2 + 6 \times 8^1 + 2 \times 8^0$
$= 448 + 48 + 2$
$= (498)_{10}$

(b) $(0.231)_8 = 2 \times 8^{-1} + 3 \times 8^{-2} + 1 \times 8^{-3}$
$= 0.25 + 0.04687 + 0.001953$
$= (0.2988)_{10}$

(c) $(762.231)_8 = 7 \times 8^2 + 6 \times 8^1 + 2 \times 8^0 + 2 \times 8^{-1} + 3 \times 8^{-2} + 1 \times 8^{-3}$
$= 448 + 48 + 2 + 0.25 + 0.04687 + 0.001953$
$= (498.2988)_{10}$

Example 2.5 Convert the following octal numbers to their binary equivalents:
(a) $(723)_8$ (b) $(0.301)_8$ (c) $(723.301)_8$

Solution

Each digit of the octal number is represented by an equivalent 3-bit binary code.

(a) $(723)_8 = (111010011)_2$

(b) $(0.301)_8 = (0.011000001)_2$

(c) $(723.301)_8 = (111010011.011000001)_2$

Example 2.6 Convert the following binary numbers to their octal equivalents:
(a) 10010111 (b) 0.0110101 (c) 10110111.0110101

Solution

(a) $10010111 = 10\,010\,111$
$= 010\,010\,111$
$= (227)_8$

(b) $0.0110101 = 0.011\,010\,1$
$= 0.011\,010\,100$
$= (0.324)_8$

(c) 10110111. 0110101

Integer part $= 10110111$
$= 10\,110\,111$
$= 010\,110\,111$
$= (267)_8$

Fractional part $= 0.\ 0110101$
$= 0.\ 011\,010\,1$
$= 0.\ 011\,010\,100$
$= (0.324)_8$

$10110111.\ 0110101 = (267)_8 + (0.324)_8$
$= (267.324)_8$

2.3.7 Hexadecimal to Decimal Conversion

The hexadecimal number system is also a positional number system, where each hexadecimal digit carries a certain weight based on its position relative to the hexadecimal point. The hexadecimal number can be converted to its decimal equivalent by summing the weights of various positions. The generalized method for hexadecimal to decimal conversion is explained below.

Consider the hexadecimal number: $H_4H_3H_2H_1H_0 \cdot H_{-1}H_{-2}H_{-3}$

The first leftmost digit from the hexadecimal point is denoted by H_0. Subscript 0 defines the position of the digit. The value of H_0 in decimal is $H_0 \times 16^0$. Similarly the value of H_1 is $H_1 \times 16^1$, and so on. The first digit right to the hexadecimal point is denoted by H_{-1}. The value of H_{-1} is $H_{-1} \times 16^{-1}$. The decimal number of the hexadecimal number is the sum of all these digit values.

$$\text{Decimal} = H_4 \times 16^4 + H_3 \times 16^3 + H_2 \times 16^2 + H_1 \times 16^1 + H_0 \times 16^0 + H_{-1} \times 16^{-1} + H_{-2} \times 16^{-2} + H_{-3} \times 16^{-3}$$

Examples

(i) $(A23)_{16} = 10 \times 16^2 + 2 \times 16^1 + 3 \times 16^0$
$= 2560 + 32 + 3 = (2595)_{10}$

(ii) $(0.351)_{16} = 3 \times 16^{-1} + 5 \times 16^{-2} + 1 \times 16^{-3}$
$= 0.1875 + 0.01953 + 0.0002441$
$= (0.20727)_{10}$

(iii) $(A23.351)_{16} = 10 \times 16^2 + 2 \times 16^1 + 3 \times 16^0 + 3 \times 16^{-1} + 5 \times 16^{-2} + 1 \times 16^{-3}$
$= 2560 + 32 + 3 + 0.1875 + 0.01953 + 0.0002441$
$= (2595.20727)_{10}$

2.3.8 Decimal to Hexadecimal Conversion

There are different methods to convert the integer and decimal parts of a decimal number into its hexadecimal equivalent.

(a) Integer part

The integer part of a decimal number can be converted to hexadecimal by using the repeated division method. The integer part of a decimal number is divided by 16 and the remainder is found after each division until the quotient becomes zero. The hexadecimal number is obtained by writing the first remainder as the LSD and the last remainder as the MSD.

For example, $(293)_{10} = (?)_{16}$

	Quotient	*Remainder*	
$293 \div 16$	18	5	♦ LSD
$18 \div 16$	1	2	
$1 \div 16$	0	1	♦ MSD

Ans: $(293)_{10} = (125)_{16}$

(b) Fractional part

The fractional part of a decimal number can be converted to hexadecimal by using the repeated multiplication method. The fractional part is multiplied by 16, the integer part is found after each multiplication operation until the fractional part becomes zero. The fractional part of the hexadecimal result is obtained by writing the first integer as the MSD and the last integer as the LSD.

For example, $(0.210)_{10} = (?)_{16}$

Multiplication	*Integer part*		
	Decimal	*Hex*	
$0.210 \times 16 = 3.36$	3	3	♦ MSD
$0.36 \times 16 = 5.76$	5	5	
$0.76 \times 16 = 12.16$	12	C	
$0.16 \times 16 = 2.56$	2	2	
$0.56 \times 16 = 8.96$	8	8	
$0.96 \times 16 = 15.36$	15	F	♦ LSD

Ans: $(0.210)_{10} = (0.35C28\ F)_{16}$

Now let us consider a decimal number having both integer and fractional parts. The integer and fractional parts are treated separately for the conversion. The hexadecimal representation of such a decimal number is the sum of hexadecimal representation of the integer part and the fractional part.

For example, $(125.201)_{10} = (?)_{16}$

Conversion of integer part

	Quotient	*Integer part*		
		Decimal	*Hex*	
$125 \div 16$	7	13	D	♦ LSD
$7 \div 16$	0	7	7	♦ MSD

$(125)_{10} = (7D)_{16}$

Conversion of fractional part

	Integer		
	Decimal	*Hex*	
$0.201 \times 16 = 3.216$	3	3	♦ MSD
$0.216 \times 16 = 3.456$	3	3	
$0.456 \times 16 = 7.296$	7	7	
$0.296 \times 16 = 4.736$	4	4	
$0.736 \times 16 = 11.776$	11	B	♦ LSD

$(0.201)_{10} = (0.3374B)_{16}$

Ans: $(125.201)_{10} = (7D)_{16} + (0.3374B)_{16} = (7D.3374B)_{16}$

2.3.9 Hexadecimal to Binary Conversion

A hexadecimal number is also known as a 4-bit binary number, because each digit of a hexadecimal number can be represented by a 4-bit binary number, as given in Table 2.3.

Table 2.3 Hexadecimal digit and its binary equivalent

Hexadecimal	*Binary*	*Hexadecimal*	*Binary*
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

The conversion of each hexadecimal digit to its 4-bit binary equivalent performs the conversion from hexadecimal to binary.

(i) $(47A)_{16} = (?)_2$

$4 = 0100, \quad 7 = 0111, \quad A = 1010$

Ans: $(47A)_{16} = (010001111010)_2$

(ii) $(0.264)_{16} = (?)_2$

$2 = 0010, \quad 6 = 0110, \quad 5 = 0101$

Ans: $(0.265)_{16} = (0.001001100101)_2$

(iii) $(F23.6A1)_{16} = (?)_2$

$F = 1111, \quad 2 = 0010, \quad 3 = 0011, \quad 6 = 0110, \quad A = 1010, \quad 1 = 0001$

Ans: $(F23.6A1)_{16} = (111100100011.011010100001)_2$

2.3.10 Binary to Hexadecimal Conversion

There are different methods to convert the integer and decimal parts of a binary number into its hexadecimal equivalent.

(a) Integer part

The integer part of a binary number can be converted into its hexadecimal equivalent by making groups of four bits starting from LSB and moving towards MSB and then replacing each group of four bits by its equivalent hexadecimal representation. If the number of bits is less than four in the last group, add zeros to the left side.

For example, $(100011110101)_2 = 1000\ 1111\ 0101$
$= (8F5)_{16}$

(b) Fractional part

The fractional part of a binary number can be converted into its hexadecimal equivalent by making groups of four bits starting from MSB and moving towards LSB and then replacing each group of four bits by its equivalent hexadecimal representation. If the number of bits is less than four in the last group, add zeros to the right side.

For example, $(0.010111101)_2 = 0.0101\ 1110\ 1000$
$= (0.5E8)$

Now let us consider a binary number having both integer and fractional parts.

For example, $1010010111.0110101 = (?)_{16}$

Integer part $= 1010010111$

$= 10\ 1001\ 0111$

$= 0010\ 1001\ 0111$ (by adding two 0s to the left)

$= (297)_{16}$

Fractional part $= 0.0110101$

$= 0.0110\ 101$

$= 0.0110\ 1010$ (by adding one 0 to the right)

$= (0.6A)_{16}$

$1010010111.0110101 = (297)_{16} + (0.6A)_{16} = (297.6A)_{16}$

Example 2.7 Convert the following decimal numbers to their hexadecimal equivalents:

(a) $(214)_{10}$ (b) $(0.356)_{10}$ (c) $(214.356)_{10}$

Solution

(a) $(214)_{10}$

Quotient	*Remainder*			
	Decimal	Hex	
$214 \div 16$	13	6	6	♦ LSD
$13 \div 16$	0	13	D	♦ MSD

Ans: $(214)_{10} = (D6)_{16}$

(b) $(0.356)_{10}$

Multiplication	*Decimal*	*Hex*	
$0.356 \times 16 = 5.696$	5	5	♦ MSD
$0.696 \times 16 = 11.136$	11	B	
$0.136 \times 16 = 2.176$	2	2	
$0.176 \times 16 = 2.816$	2	2	♦ LSD

Ans: $(0.356)_{10} = (0.5B22)_{16}$

(c) $(214.356)_{10}$

$(214)_{10} = (D6)_{16}$

and $(0.356)_{10} = (0.5B22)_{16}$

Hence, $(214.356)_{10} = (D6.5B22)_{16}$

Example 2.8 Convert the following hexadecimal numbers to their decimal equivalents:

(a) $(356)_{16}$ (b) $(0.21)_{16}$ (c) $(2AF.31)_{16}$

Solution

(a) $(356)_{16} = 3 \times 16^2 + 5 \times 16^1 + 6 \times 16^0$

$= 3 \times 256 + 5 \times 16 + 6 \times 1$

$= 768 + 80 + 6$

$= (854)_{10}$

(b) $(0.21)_{16} = 2 \times 16^{-1} + 1 \times 16^{-2}$

$= \frac{2}{16} + \frac{1}{256}$

$= 0.125 + 0.0039$

$= (0.1289)_{10}$

(c) $(2AF.31)_{16} = 2 \times 16^2 + 10 \times 16^1 + 15 \times 16^0 + 3 \times 16^{-1} + 1 \times 16^{-2}$

$= 2 \times 256 + 10 \times 16 + 15 + 316 + 1256$

$= 512 + 160 + 15 + 0.1875 + 0.0339$

$= (687.19)_{10}$

Example 2.9 Convert the following binary numbers to their hexadecimal equivalents:

(a) 1010110110111 (b) 0.010011011

(c) 101011011001.1010100

Solution

(a) $(1010110110111)_2 = 1\ 0101\ 1011\ 0111$

$= 0001\ 0101\ 1011\ 0111$ (by adding three 0s to the left)

$= (15B7)_{16}$

(b) $(0.010011011)_2 = 0.0100\ 1101\ 1$

$= 0.0100\ 1101\ 1000$ (by adding three 0s to the right)

$= (0.4D8)_{16}$

(c) $(101011011001.1010111)_2 = 1010\ 1101\ 1001.1010\ 111$

$= 1010\ 1101\ 1001.1010\ 1110$

$= (AD9.AE)_{16}$

Example 2.10 Convert the following hexadecimal numbers into their binary equivalents:

(a) $(A23.4E)_{16}$ (b) $(F23)_{16}$ (c) $(0.45B)_{16}$

Solution

Each digit of a hexadecimal number is represented by its equivalent 4-bit binary code.

(a) $(A23.4E)_{16} = (101000100011.01001110)_2$

(b) $(F23)_{16} = (111100100011)_2$

2.4 SIGNED BINARY NUMBER

The numbers without positive or negative signs are known as *unsigned numbers*. The unsigned numbers are always considered as positive numbers. In the signed number system, the number may be positive or negative. In the decimal number system, the positive sign (+) is used before the number to define a positive number and the negative sign (–) is used before the number to define a negative number. In case of signed binary number system, the most significant bit represents the sign of the number. When the MSB is 1, the number is negative; and when it is 0, the number is positive.

There are three methods used to represent sign binary numbers.

- Sign–magnitude representation,
- 1's complement representation, and
- 2's complement representation.

2.4.1 Sign–Magnitude Representation

In sign magnitude representation of a binary number, the MSB represents the sign and the remaining bits represent its magnitude. When the MSB is 1, the sign is negative; and when it is 0, the sign is positive.

Consider the binary number: $(B_7B_6B_5B_4B_3B_2B_1B_0)_2$

where B_7 represents the sign and B_6 to B_0 represent the magnitude of the binary number.

For example: 00001000 +8
10001000 –8
01111111 +127
11111111 –127

The unsigned 8-bit binary representation can represent the maximum decimal equivalent $2^8 - 1 = 255$; and the sign-magnitude 8-bit binary representation can represent the maximum positive decimal equivalent 127 and the maximum negative decimal equivalent is –127. In generalized form, the n-bit unsigned binary number represents the maximum positive number $(2^n - 1)$, and the n-bit binary sign-magnitude binary number system represents the maximum positive number $(2^{n-1} - 1)$ and the maximum negative number $(2^{n-1} - 1)$.

2.4.2 1's Complement Representation

The 1's complement of a binary number is obtained when each bit of a binary number is subtracted from 1. The 1's complement of 0 is $1 - 0 = 1$ and the 1's complement of 1 is $1 - 1 = 0$. The 1's complement of a binary number is obtained by replacing 0 by 1 and 1 by 0. In this representation, MSB is 0 for positive and 1 for negative binary numbers.

For example, representation of +8 is 00001000, and the 1's complement of +8 is 11110111

The 1's complement of a positive binary number is negative and vice-versa. $(+8)_{10}$ is represented by $(00001000)_2$, whereas $(11110111)_2$ represents $(-8)_{10}$ in 1's complement representation of binary number.

An unsigned 8-bit binary represents the maximum decimal equivalent as 255. But the 1's complement 8-bit binary number represents the maximum positive number as 127 and the maximum negative number as –127. In generalized form, the n-bit unsigned binary number represents the maximum positive number as $(2^n - 1)$, and the n-bit binary number in 1's complement form represents the maximum positive number as $(2^{n-1} - 1)$ and the maximum negative number as $(2^{n-1} - 1)$. The following steps are used to find the 1's complement representation of a negative number.

Step 1: Write the positive number in binary form.

Step 2: Find the 1's complement of the binary number.

2.4.3 2's Complement Representation

The 2's complement of a binary number is obtained by adding 1 to the 1's complement of a binary number. In this representation, the most significant bit is 0 for positive number and 1 for negative number.

For example: 1's complement of (+ 8) = 1 1 1 1 0 1 1 1
2's complement of (+ 8) = 1 1 1 1 0 1 1 1
\+ 1
1 1 1 1 1 0 0 0

The 2's complement of a positive number is negative and vice–versa. $(+8)_{10}$ is represented by $(00001000)_2$, whereas $(11111000)_2$ represents $(-8)_{10}$ in 2's complement representation.

Unsigned 8-bit binary number represents the maximum decimal number as 255; the 2's complement eight-bit binary number represents the maximum positive number as 127 and the maximum negative number as –128. In generalized form, the n-bit unsigned binary number represents the maximum positive number as $(2^n - 1)$, and the n-bit binary number in 2's complement form represents the maximum positive number as $(2^{n-1} - 1)$ and the maximum negative number as (2^{n-1}).

Steps to represent the negative number in 2's complement representation are:

Step 1: Write the positive sign number.

Step 2: Find the 1's complement of the number by replacing 0 by 1 and 1 by 0.

Step 3: Find the 2's complement of the number by adding 1 to 1's complement of the number.

Example 2.11 Find the sign-magnitude representation of numbers using 8 bits:
(a) + 27 (b) –27 (c) + 101 (d) –106

Solution

(a) $(27)_{10}$ =

	Quotient	*Remainder*	
27 ÷ 2	13	1	♦ LSB
13 ÷ 2			

$6 \div 2$	3	0	
$3 \div 2$	1	1	
$1 \div 2$	0	1	♦ MSB

Ans: $(27)_{10} = (11011)_2$
$= (0011011)_2$. (Magnitude is represented by 7 bits)
The number is positive and the most significant bit is 0.
Hence, the sign-magnitude representation of $(27)_{10} = 00011011$

(b) The binary equivalent of $(27)_{10} = (0011011)_2$
The number is negative and the most significant bit is 1.
Hence, sign-magnitude representation of $(-27)_{10} = 10011011$

(c) $(101)_{10} =$

	Quotient	*Remainder*	
$101 \div 2$	50	1	♦ LSB
$50 \div 2$	25	0	
$25 \div 2$	12	1	
$12 \div 2$	6	0	
$6 \div 2$	3	0	
$3 \div 2$	1	1	
$1 \div 2$	0	1	♦ MSB

Hence, $(101)_{10} = (1100101)_2$
The number is positive and the most significant bit is 0.
Hence, the sign-magnitude representation of $(+101)_{10} = 01100101$

(d) $(106)_{10} =$

	Quotient	*Remainder*	
$106 \div 2$	53	0	♦ LSB
$53 \div 2$	26	1	
$26 \div 2$	13	0	
$13 \div 2$	6	1	
$6 \div 2$	3	0	
$3 \div 2$	1	1	
$1 \div 2$	0	1	♦ MSB

Hence, $(106)_{10} = (1101010)_2$
The number is negative and the most significant bit is 1.
Hence, the sign-magnitude representation of $(-106)_{10} = 11101010$

Example 2.12 Find decimal equivalents of the following binary numbers:
(a) $(10100001)_2$ (b) $(00010011)_2$ (c) $(10110011)_2$
Assume the given numbers in sign-magnitude form.

Solution

(a) The most significant bit is 1; the sign of the number is negative. The remaining seven bits represent the magnitude of the number.

$(0100001)_2 = 0\times2^6 + 1\times2^5 + 0\times2^4 + 0\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0$
$= 32 + 1$
$= 33$
Hence, the decimal equivalent of $(10100001)_2 = (-33)_{10}$

(b) The most significant bit is 0; the sign of the number is positive.
$(0010011)_2 = 0\times2^6 + 0\times2^5 + 1\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$
$= 16 + 2 + 1$
$= 19$
Hence, the decimal equivalent of $(00010011)_2 = (+19)_{10}$

(c) The most significant bit is 1; the sign of the number is negative.
$(0110011)_2 = 0\times2^6 + 1\times2^5 + 1\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$
$= 32 + 16 + 2 + 1$
$= 51$
Hence, the decimal equivalent of $(10110011)_2 = (-51)_{10}$

Example 2.13 Represent the following decimal numbers in 1's complement representation using eight bits.
(a) -67 (b) 102 (c) -88

Solution

(a) *Step 1:* Find the binary equivalent of the number.
$(67)_{10} =$

	Quotient	*Remainder*	
$67 \div 2$	33	1	LSB
$33 \div 2$	16	1	
$16 \div 2$	8	0	
$8 \div 2$	4	0	
$4 \div 2$	2	0	
$2 \div 2$	1	0	
$1 \div 2$	0	1	MSB

Hence, $(67)_{10} = (1000011)_2$
Step 2: Write the positive number using 8-bits
$(+67)_{10} = (01000011)_2$
Step 3: Find the 1's complement by replacing 0 by 1 and 1 by 0
1's complement of $(+67)_{10} = (-67)_{10} = (10111100)_2$

(b) *Step 1:* Find the binary equivalent of the number.
$(102)_{10} =$

	Quotient	*Remainder*	
$102 \div 2$	51	0	♦ LSB
$51 \div 2$	25	1	
$25 \div 2$	12	1	

$12 \div 2$	6	0
$6 \div 2$	3	0
$3 \div 2$	1	1
$1 \div 2$	0	1 ◆ MSB

Hence, $(102)_{10} = (1100110)_2$

Step 2: Write the positive number using 8-bits.

$(+102)_{10} = (01100110)_2$

The 1's complement representation of the positive number is same as the sign-magnitude representation of the positive number.

(c) *Step 1:* Find the binary equivalent of the number.

$(88)_{10} =$

	Quotient	*Remainder*	
$88 \div 2$	44	0	◆ LSB
$44 \div 2$	22	0	
$22 \div 2$	11	0	
$11 \div 2$	5	1	
$5 \div 2$	2	1	
$2 \div 2$	1	0	
$1 \div 2$	0	1	◆ MSB

Hence, $(88)_{10} = (1011000)_2$

Step 2: Write the positive number using 8-bits.

$(+88)_{10} = (01011000)_2$

Step 3: Find the 1's complement by replacing 0 by 1 and 1 by 0

1's complement of $(+88)_{10} = (-88)_{10} = (10100111)_2$

Example 2.14 Find decimal equivalent of the following binary numbers.

(a) $(10100111)_2$ (b) $(01010011)_2$ (c) $(10111011)_2$

Assume the given numbers in 1's complement representation.

Solution

(a) *Step 1:* Check the sign of the given number.

The most significant bit of the given number is 1; the sign of the number is negative.

Step 2: Find the 1's complement of the number.

1's complement of $(10100111)_2 = 01011000$

Step 3: Find the decimal equivalent.

$$1011000 = 1\times2^6 + 0\times2^5 + 1\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 0\times2^0$$
$$= 64 + 16 + 8$$
$$= 88$$

Hence, the decimal equivalent of $(10100111)_2 = (-88)_{10}$

(b) *Step 1:* Check the sign of the given number.

The most significant bit of the given number is 0; the sign of the number is positive.

Step 2: Since the number is positive, go to step 3.

Step 3: Find the decimal equivalent.

$$1010011 = 1\times2^6 + 0\times2^5 + 1\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$$
$$= 64 + 16 + 2 + 1$$
$$= 83$$

Hence, the decimal equivalent of $(01010011)_2 = (+83)_{10}$

(c) *Step 1:* Check the sign of the given number.

The most significant bit of the given number is 1; the sign of the number is negative.

Step 2: Find the 1's complement of the number.

1's complement of $(10111011)_2 = 01000100$

Step 3: Find the decimal equivalent.

$$01000100 = 1\times2^6 + 0\times2^5 + 0\times2^4 + 0\times2^3 + 1\times2^2 + 0\times2^1 + 0\times2^0$$
$$= 64 + 4$$
$$= 68$$

Hence, the decimal equivalent of $(10111011)_2 = (-68)_{10}$

Example 2.15 Represent the following decimal numbers in 2's complement representation using 8-bits.

(a) -44 (b) 64 (c) -89

Solution

(a) *Step 1:* Find the binary equivalent of the number.

$(44)_{10} =$

	Quotient	*Remainder*	
$44 \div 2$	22	0	◆ LSB
$22 \div 2$	11	0	
$11 \div 2$	5	1	
$5 \div 2$	2	1	
$2 \div 2$	1	0	
$1 \div 2$	0	1	◆ MSB

Hence, $(44)_{10} = (101100)_2$

$= (00101100)_2$ (adding two 0s to the left)

Step 2: Write the positive number using 8-bits.

$(+44)_{10} = (00101100)_2$

Step 3: Find the 1's complement by replacing 0 by 1 and 1 by 0.

1's complement of $(+44)_{10} = (-44)_{10} = (11010011)_2$

Step 4: Find the 2's complement by adding 1 to the 1's complement.

2's complement of $(+44)_{10} = (-44)_{10} =$ 1 1 0 1 0 0 1 1

$+$ 1

1 1 0 1 0 1 0 0

2's complement

(b) *Step 1:* Find the binary equivalent of the number.

	Quotient	*Remainder*	
$64 \div 2$	32	0	♦ LSB
$32 \div 2$	16	0	
$16 \div 2$	8	0	
$8 \div 2$	4	0	
$4 \div 2$	2	0	
$2 \div 2$	1	0	
$1 \div 2$	0	1	♦ MSB

Hence, $(64)_{10} = (1000000)_2$

Step 2: Write the positive number using 8-bits.

$(+64)_{10} = (01000000)_2$

The 2's complement representation of a positive number is same as the sign-magnitude representation of a positive number.

Hence, the 2's complement representation of $(+64)_{10} = 01000000$

(c) *Step 1:* Find the binary equivalent of the number.

	Quotient	*Remainder*	
$89 \div 2$	44	1	♦ LSB
$44 \div 2$	22	0	
$22 \div 2$	11	0	
$11 \div 2$	5	1	
$5 \div 2$	2	1	
$2 \div 2$	1	0	
$1 \div 2$	0	1	♦ MSB

Hence, $(89)_{10} = (1011001)_2$

$= (01011001)_2$ (adding 0 to the left)

Step 2: Write the positive number using 8-bits.

$(+89)_{10} = (01011001)_2$

Step 3: Find the 1's complement by replacing 0 by 1 and 1 by 0.

1's complement of $(+89)_{10} = (-89)_{10} = (10100110)_2$

Step 4: Find the 2's complement by adding 1 to 1's complement.

2's complement of $(+89)_{10} = (-89)_{10} =$ 1 0 1 0 0 1 1 0

$+$ 1

1 0 1 0 0 1 1 1

Hence, the 2's complement representation of $(-89)_{10} = 10100111$

Example 2.16 Find decimal equivalent of the following binary numbers

(a) $(10011001)_2$ (b) $(01100111)_2$ (c) $(10101011)_2$

Assume the given numbers in 2's complement representation.

Solution

(a) *Step 1:* Check the sign of the given number.

The most significant bit of the given number is 1; the sign of the number is negative.

Step 2: Find the 1's complement of the number.

1's complement of $(10011001)_2 = 01100110$

Step 3: Find the 2's complement of the number.

0 1 1 0 0 1 1 0

$+$ 1

0 1 1 0 0 1 1 1

Step 4: Find the decimal equivalent.

$1100111 = 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$= 64 + 32 + 4 + 2 + 1$

$= 103$

Hence, the decimal equivalent of $(10011001)_2 = (-103)_{10}$

(b) Given number is $(01100111)_2$

Step 1: Check the sign of the given number.

The most significant bit of the given number is 0, the sign of the number is positive.

Step 2: Since the number is positive, go to step 3.

Step 3: Since the number is positive, go to step 4.

Step 4: Find the decimal equivalent.

$1100111 = 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$= 64 + 32 + 4 + 2 + 1$

$= 103$

Hence, the decimal equivalent of $(01100111)_2 = (+103)_{10}$

(c) *Step 1:* Check the sign of the given number.

The most significant bit of the given number is 1; the sign of the number is negative.

Step 2: Find the 1's complement of the number.

1's complement of $(10101011)_2 = 01010100$

Step 3: Find the 2's complement of the number.

0 1 0 1 0 1 0 0

$+$ 1

0 1 0 1 0 1 0 1

Step 4: Find the decimal equivalent.

$1010101 = 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= 64 + 16 + 4 + 1$

$= 85$

Hence, the decimal equivalent of $(10101011)_2 = (-85)_{10}$

2.5 FLOATING POINT REPRESENTATION OF NUMBER

In the decimal number system, very large and very small numbers are expressed in scientific notation as: 0.423×10^{23} and 0.450×10^{-23}. This scientific notation consists of two parts: the first part represents a signed, fixed point number called as *mantissa* and the second part designates the position of the decimal point and is called as *exponent*.

For example, the decimal number 51729.739 is represented in scientific notation as $+5.1729738 \times 10^{+04}$, where

Mantissa: +5.1729739

Exponent: +04

Here, the mantissa is considered to be a fixed point fraction. This representation is equivalent to the number expressed as a fraction 10 times by an exponent.

Similarly binary numbers can be expressed by the scientific notation known as the floating point representation of number. The floating point representation of a number consists of two parts: the first part represents a signed, fixed point number called the mantissa (m) and the second part designates the position of the binary point and is called the exponent (e). The fixed point mantissa may be a fraction or an integer part of the number.

The floating point numbers representated in binary by the expression

$$(-1)^s \times (\text{Significant}) \times 2^{(exponent + bias)} \quad (2.6)$$

where 's' is the sign bit defined as $s = 0$ for positive numbers and $s = 1$ for negative numbers, *significant* defines the magnitude of mantissa and *bias* is the value that is used to control the range of the number. These mantissa and bias exponent are represented by the number of bits in binary. The number of bits is used to determine the accuracy of the system. IEEE has defined two floating point standards called as *single precision* and *double precision*.

In single precision, the word size of the floating point number is 32-bits and 127 is the bias. The format of single precision floating point number is shown in Fig. 2.1. Out of these 32-bits, MSB (D_{31}) is used for sign, LSB 23-bits (D_{22} to D_0) are used for significant and 8-bits (D_{30} to D_{23}) are used for bias exponent.

D_{31}	D_{30} - D_{23}	D_{22} - D_0
S	Bias exp.	Significant

Fig. 2.1 Format of single precision floating point number

Let us consider the example of a decimal number 44.25. Its binary representation is

$$(44.25)_{10} = 101100.01$$

$$= 1.0110001 \times 2^5$$

The mantissa is converted to 23-bits significant value by putting zeros to the right side of the number as

Significant = 01100010000000000000000

The bias exponent is the sum of exponent value (5) and bias (127). It is given as

Bias exponent = $5 + 127 = (132)_{10}$

Using the conversion techniques, its binary representation is

Bias exponent = 10000100

The number is positive, hence the sign bit is 0, and the single precision floating point representation of the given number is:

0	10000100	01100010000000000000000

In double precision, the word size of the floating point number is 64-bits and 256 is the bias. The format of double precision floating point number is shown in Fig. 2.2. Out of these 64-bits, MSB (D_{63}) is used for sign, LSB 52-bits (D_{51} to D_0) are used for significant and 11-bits (D_{62} to D_{52}) are used for bias exponent.

D_{63}	D_{62} - D_{52}	D_{51} - D_0
S	Bias exp.	Significant

Fig. 2.2 Format of double precision floating point number

Let us consider the example of a decimal number 144.125. Its binary representation is

$$(144.125)_{10} = 10010000.001$$

$$= 1.0010000001 \times 2^7$$

The mantissa is converted to 52-bits significant value by putting zeros to the right side of the number as

Significant = 0010000001000000000000000000000000000000000000000000

The bias exponent is the sum of exponent value 7 and bias 256. It is given as

Bias exponent = $5 + 256 = (261)_{10}$

Using the conversion techniques, its 11-bits binary representation is

Bias exponent = 00100000111

The number is negaitive, hence the sign bit is 1, and the double precision floating point representation of the given number is:

1	00100000111	0010000001000000000000000000000000000000000000000000

2.6 BINARY ARITHMETIC

The computer is a digital system that supports various arithmetic operations. It performs addition, subtraction, multiplication, and division over the binary data. To learn the basic circuits of a digital system, it is necessary to study binary

arithmetic operations such as binary addition, binary subtraction, binary multiplication, and binary division.

2.6.1 Binary Addition

The addition of two binary numbers is performed exactly in the same manner as the addition of decimal numbers. Only four cases occur in the addition of two binary bits.

$0+0=0$, $0+1=1$, $1+0=1$, and $1+1=10$, i.e. the sum = 0 and carry = 1

The rules of binary addition are given in Table 2.4.

Table 2.4 Sum of Binary bits

A	*B*	*Sum*	*Carry*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

When the binary numbers are more than one bit, the addition takes place bit by bit, which starts from left side. The carry of the previous sum is added to the next bit addition.

For example,
```
  10110111   ♦ A
+ 01110101   ♦ B
  ---------
   1110111   ♦ Carry
 ---------
 100101100
 ---------
```

2.6.2 Binary Subtraction

The subtraction of two binary numbers is performed exactly in the same manner as the subtraction of decimal numbers. Only four cases occur in the subtraction of two binary bits.

$0-0=0, 0-1=-1, 1-0=1$, and $1-1=0$

In $0-1=-1$, the result is negative. It indicates that the second number is greater than the first one. Similar to decimal number system, a borrow is generated.

The rules of binary subtraction are given in Table 2.5.

Table 2.5 Binary subtraction

A	*B*	*Difference*	*Borrow*
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

When the binary numbers are more than one bit, the subtraction takes place bit by bit, which starts from the left side. The borrow of the previous subtraction is subtracted from the next bit subtraction.

For example,
```
  10100111   ♦ A
- 01110101   ♦ B
  ---------
    111      ♦ Borrow
  ---------
  00110010
  ---------
```

2.6.3 Binary Multiplication

The binary multiplication is similar to the decimal multiplication. If the multiplier bit is 1, then the partial product is same as the multiplicand. If the multiplier bit is 0, then the partial product is 0.

Consider the example: 1010×1001

```
    1 0 1 0   ♦ multiplicand
  × 1 0 0 1   ♦ multiplier
  ---------
    1 0 1 0
  0 0 0 0 -
0 0 0 0 - -
1 0 1 0 - - -
---------------
1 0 1 1 0 1 0
---------------
```

2.6.4 Binary Division

The binary division is same as the decimal division. Binary division has two results, i.e. quotient and remainder. Let us consider the example: 10011011 is divided by 100.

```
            1 0 0 1 1 0
        ---------------
1 0 0 | 1 0 0 1 1 0 1 1
        1 0 0
      -----------------
        0 0 0 1 1 0
            1 0 0
          -------
          0 1 0 1
              1 0 0
            -------
            0 0 1 1
            -------
```

∴ The quotient is 100011 and the remainder is 11.

Example 2.17 Find the addition of the following binary numbers.

(a) 10110101 and 10100011

(b) 10110001 and 00110111

(ii) Addition of positive and negative number:

$8 + (-9) = -1$

Step 1: Find the number of bits required to represent the number.

$2^{n-1} - 1 > \max(8, 9, 1)$

$2^{n-1} - 1 > 9$ and $n = 5$

Step 2: 1's complement representation of (+8) = 01000

1's complement representation of (–9) = 1's complement of (+9)

= 1's complement of (01001)

= 10110

Step 3: Binary addition:

```
  01000   ♦ A
+ 10110   ♦ B
-------
  11110
```

The most significant bit is 1, so the answer is negative, and it is in 1's complement form.

Result = –(00001) = –1

(iii) Addition of two negative numbers:

(–8) + (–9) = –17

Step 1: Find the number of bits required to represent the number.

$2^{n-1} - 1 > \max(8, 9, 17)$

$2^{n-1} - 1 > 17$ and $n = 6$

Step 2: 1's complement representation of (–8) = 1's complement of (+8)

= 1's complement of (001000)

= 110111

1's complement representation of (–9) = 1's complement of (+9)

= 1's complement of (001001)

= 110110

Step 3: Binary addition:

```
  110111   ♦ A
+ 110110   ♦ B
  1 11     ♦ Carry
--------
1101101
  └──→ 1   ♦ Carry is added
--------
  101110
```

The most significant bit is 1, so the answer is negative, and it is in 1's complement form.

∴ Result = –(010001) = –17

Subtraction

The subtraction of a binary number *B* from another binary number *A* is equivalent to the addition of 1's complement of *B* to *A* i.e. $(A - B) = A +$ 1's complement of *B*. The algorithm for binary subtraction using 1's complements is:

1. Find the number of bits required to represent the sign number. The number of bits required to represent the sign number is *n*, such that $2^{n-1} - 1$ is greater than or equal to the maximum of the magnitude of *A*; *B*; $A - B$.
2. Find the 1's complement of the subtrahend.
3. Add *A* and the 1's complement of *B*.
4. Check the carry, if the carry is generated, add the carry in LSB position and take the 1's complement of the result.

If the MSB of result is 1, then the answer is negative; it is in 1's complement form. If the MSB is 0, then the answer is positive and in true form.

Consider the following examples.

(i) 8 – 9 = –1

8 – 9 = 8 + 1's complement of 9

Step 1: Find the number of bits required to represent the number.

$2^{n-1} - 1 > \max(8, 9, 1)$

$2^{n-1} - 1 > 9$ and $n = 5$

Step 2: 1's complement of (+9) = 1's complement of (01001)

= 10110

Step 3: Binary addition:

```
  01000   ♦ A
+ 10110   ♦ 1's complement of B
-------
  11110
```

The most significant bit is 1, the answer is negative, and it is in 1's complement form.

∴ Result = –(00001) = –1

(ii) 9 – 8 = 1

9 – 8 = 9 + 1's complement of 8

Step 1: Find the number of bits required to represent the number.

$2^{n-1} - 1 > \max(8, 9, 1)$

$2^{n-1} - 1 > 9$ and $n = 5$

Step 2: 1's complement representation of (+9) = 01001

1's complement of (+8) = 1's complement of (01000)

= 10111

Step 3: Binary addition:

```
  01001   ♦ A
+ 10111   ♦ 1's complement of B
  1111    ♦ Carry
 100000
    └──→1 ♦ Carry is added
 -------
  00001
```

The most significant bit is 0, the answer is positive, and it is in true form.

∴ Result = (00001) = 1

2.7.2 2's Complement Arithmetic

Addition

The addition of sign-binary numbers takes place using 2's complement. The algorithm for sign binary number addition using 2's complement is given below. Consider *A* and *B* are two sign numbers.

1. Find the number of bits required to represent the sign number.
   The number of bits required to represent the sign number is n, such that $2^{n-1}-1$ is greater than or equal to the maximum of the magnitude of A; B; $A+B$
2. Find the 1's complement of a number by replacing 0 by 1 and 1 by 0.
3. Find the 2's complement of the same number by adding 1 to the 1's complement.
4. Find the binary addition.
5. Discard the carry, if it is generated.

If MSB of the result is 1, then the answer is negative; it is in 2's complement form. If the MSB is 0, then the answer is positive; it is in true form.

Consider the following examples.

(i) Addition of two positive numbers:

$8 + 9 = 17$

Step 1: Find the number of bits required to represent the number.

$2^{n-1} - 1 > \max(8, 9, 17)$

$2^{n-1} - 1 > 17$ and $n = 6$

Step 2: 1's complement representation of (+8) = 001000

1's complement representation of (+9) = 001001

Step 3: 2's complement representation of (+8) = 001000

2's complement representation of (+9) = 001001

Step 4: Binary addition:

```
  001000  ♦ A
+ 001001  ♦ B
    1     ♦ Carry
 -------
  010001
 -------
```

The most significant bit is 0, so the sign of the number is positive.

Result = +(010001) = +17

(ii) Addition of positive and negative numbers:

$8 + (-9) = -1$

Step 1: Find the number of bits required to represent the number

$2^{n-1} - 1 > \max(8, 9, 1)$

$2^{n-1} - 1 > 9$ and $n = 5$

Step 2: 1's complement representation of (+8) = 01000

1's complement representation of (–9) = 1's complement of (+9)

= 1's complement of (01001)

= 10110

Step 3: 2's complement representation of (+8) = 01000

2's complement representation of (–9) = 10110 + 1

= 10111

Step 4: Binary addition:

```
  01000  ♦ A
+ 10111  ♦ B
 -------
  11111
 -------
```

The most significant bit is 1, the answer is negative, and it is in 2's complement form.

Result = –2's complement of (11111)

= –1's complement of (11111) + 1

= –(00000) + 1

= –1

(iii) Addition of two negative numbers:

$(-8) + (-9) = -17$

Step 1: Find the number of bits required to represent the number.

$2^{n-1} - 1 > \max(8, 9, 17)$

$2^{n-1} - 1 > 17$ and $n = 6$

Step 2: 1's complement representation of (–8) = 1's complement of (+8)

= 1's complement of (001000)

= 110111

1's complement representation of (–9) = 1's complement of (+9)

= 1's complement of (001001)

= 110110

Step 3: 2's complement representation of (–8) = 1's complement of (+8) + 1

= 1's complement of (001000) + 1

= 110111 + 1

= 111000

2's complement representation of (–9) = 1's complement of (+9),
= 1's complement of (001001) + 1
= 110110 + 1
= 110111

Step 4: Binary addition:

```
  111000   ♦ A
+ 110111   ♦ B
--------
 X101111   ♦ Discard the carry
```

The most significant bit is 1, the answer is negative; and it is in 2's complement form.

Result = –2's complement of (101111)
= –1's complement (101111) + 1
= –(010000) + 1
= –(010001)
= –17

Subtraction

The subtraction of a binary number B from another binary number A is equivalent to the addition of the 2's complement of B with A, i.e. $(A-B) = A + 2$'s complement of B. The algorithm for binary subtraction using 2's complement is as follows:

1. Find the number of bits required to represent the sign number.
   The number of bits required to represent the sign-binary number is n, such that $2^{n-1} - 1$ is greater than or equal to the maximum of the magnitude A; B; $A - B$.
2. Find the 1's complement of the subtrahend by replacing 0 by 1 and 1 by
3. Find the 2's complement of the subtrahend by adding 1 to the 1's complement
4. Add these numbers using binary addition.
5. Discard the carry, if it is generated.

If MSB of the result is 1, then the answer is negative; it is in 2's complement form. If the MSB is 0, then the answer is positive; it is in true form.

Consider the following examples.

(i) 8 – 9 = –1

8 – 9 = 8 + 2's complement of 9

Step 1: Find the number of bits required to represent the number.

$2^{n-1} - 1 > \max(8, 9, 1)$

$2^{n-1} - 1 > 9$ and $n = 5$

Step 2: 1's complement of (+9) = 1's complement of (01001)
= 10110

Step 3: 2's complement of (+9) = 1's complement of (+9) + 1
= 1's complement of (01001) + 1
= 10110 + 1

Step 4: Binary addition:

```
  01000   ♦ A
+ 10111   ♦ 2's complement of B
--------
  11111
```

The most significant bit is 1, the answer is negative, and it is in 2's complement form.

Result = –2's complement of (11111)
= –1's complement of (11111) + 1
= –(00000) + 1
= –(00001)
= –1

(ii) 9 – 8 = 1

9 – 8 = 9 + 1's complement of 8

Step 1: Find the number of bits required to represent the number.

$2^{n-1} - 1 > \max(8, 9, 1)$

$2^{n-1} - 1 > 9$ and $n = 5$

Step 2: 1's complement of (+8) = 1's complement of (01000)
= 10111

Step 3: 2's complement of (+8) = 1's complement of (+8) + 1
= 1's complement of (01000) + 1
= 10111 + 1
= 11000

Step 4: Binary addition:

```
  01001   ♦ A
+ 11000   ♦ 2's complement of B
      1   ♦ Carry
--------
 X00001   ♦ Discard the carry
```

The MSB is 0, the answer is positive, and it is in true form.

∴ Result = (00001) = 1

Example 2.21 Perform the following operations by using 1's complement method.

(a) 42 – 22 (b) 20 – 42 (c) –42 – 20

Solution

(a) **42 – 22 = 20**

42 – 22 = 42 + 1's complement of 22

Step 1: Find the number of bits required to represent the number:

$2^{n-1} - 1 > \max(42, 20, 22)$

$2^{n-1} - 1 > 42$ and $n = 7$

Step 2: 1's complement of (+22) = 1's complement of (0010110)
= 1101001

Step 3: Binary addition:

```
  0 1 0 1 0 1 0   ♦ A
+ 1 1 0 1 0 0 1   ♦ 1's complement of B
-----------------
  1 0 0 1 0 0 1 1
  └─────────→ 1
-----------------
  0 0 1 0 1 0 0
-----------------
```

The MSB is 0, the answer is positive, and it is in true form.

∴ Result = +(0010100) = +20

(b) **20 – 42 = – 22**

20 – 42 = 20 + 1's complement of 42

Step 1: Find the number of bits required to represent the number:

$2^{n-1} - 1 > \max(20, 42, 22)$

$2^{n-1} - 1 > 42$ and $n = 7$

Step 2: 1's complement of (+42) = 1's complement of (0101010)
= 1010101

Step 3: Binary addition:

```
  0 0 1 0 1 0 0   ♦ A
+ 1 0 1 0 1 0 1   ♦ 1's complement of B
-----------------
  1 1 0 1 0 0 1
-----------------
```

The MSB is 1, the answer is negative, and it is in complement form

∴ Result = –1's complement of (1101001)
= –(0010110)
= –22

(c) **–42 – 20 = –62**

Step 1: Find the number of bits required to represent the number.

$2^{n-1} - 1 > \max(42, 20, 62)$

$2^{n-1} - 1 > 62$ and $n = 7$

Step 2: 1's complement representation of (–42) = 1's complement of (+4

= 1's complement of (0101010)

= 1010101

1's complement representation of (–20) = 1's complement of (+2

= 1's complement of (0010100)

= 1101011

Step 3: Binary addition:

```
  1 0 1 0 1 0 1   ♦ A
+ 1 1 0 1 0 1 1   ♦ B
  1 1 1 1 1 1     ♦ Carry
-----------------
  1 1 0 0 0 0 0 0
  └─────────→ 1   ♦ Carry is added
-----------------
  1 0 0 0 0 0 1
-----------------
```

The MSB is 1, the answer is negative, and it is in 1's complement form.

∴ Result = –1's complement of (1000001)
= –(0111110)
= –62

Examples 2.22 Perform the following operations by using 2's complement method.

(a) 46 – 23 (b) 23 – 46 (c) – 46 – 25

Solution

(a) **46 – 23 = 23**

46 – 23 = 46 + 1's complement of 23

Step 1: Find the number of bits required to represent the number.

$2^{n-1} - 1 > \max(46, 23, 23)$

$2^{n-1} - 1 > 46$ and $n = 7$

Step 2: 1's complement of (+23) = 1's complement of (0010111)
= 1101000

Step 3: 2's complement of (+23) = 1's complement of (+23) + 1
= 1's complement of (0010111) + 1
= 1101000 + 1
= 1101001

Step 4: Binary addition:

```
  0 1 0 1 1 1 0   ♦ A
+ 1 1 0 1 0 0 1   ♦ 2's complement of B
  1   1           ♦ Carry
-----------------
✱0 0 1 0 1 1 1   ♦ Discard the carry
-----------------
```

The MSB is 0, the answer is positive, and it is in true form.

∴ Result = (0010111)
= +23

(b) **23 – 46 = –23**

23 – 46 = 23 + 2's complement of 46

Step 1: Find the number of bits required to represent the number.

$2^{n-1} - 1 > \max(23, 46, 23)$

Step 2: 1's complement of (+46) = 1's complement of (0101110)
= 1010001

Step 3: 2's complement of (+46) = 1's complement of (+46) + 1
= 1's complement of (0101110) + 1
= 1010001 + 1
= 1010010

Step 4: Binary addition:

```
  0 0 1 0 1 1 1   ♦ A
+ 1 0 1 0 0 1 0   ♦ 2's complement of B
-----------------
  1 1 0 1 0 0 1
-----------------
```

The MSB is 1, so the answer is negative and it is in 2's complement form.

∴ Result = –2's complement of (1101001)
= –1's complement of (1101001) + 1
= –(0010110) + 1
= –(0010111)
= –23

(c) **–46 – 25 = –71**
(–46) + (–25) = –71

Step 1: Find the number of bits required to represent the number.
$2^{n-1} - 1 > \max(46, 25, 71)$
$2^{n-1} - 1 > 71$ and $n = 8$

Step 2: 1's complement representation of (–46) = 1's complement of (+46)
= 1's complement of (00101110)
= 11010001

1's complement representation of (–25) = 1's complement of (+25)
= 1's complement of (00011001)
= 11100110

Step 3: 2's complement representation of (–46) = 1's complement of (–46) + 1
= 1's complement of (00101110) + 1
= 11010001 + 1
= 11010010

2's complement representation of (–25) = 1's complement of (+25) + 1
= 1's complement of (00011001) + 1
= 11100110 + 1
= 11100111

Step 4: Binary addition:

```
  1 1 0 1 0 0 1 0   ♦ A
+ 1 1 1 0 0 1 1 1   ♦ B
  1     1 1         ♦ Carry
-----------------
* 1 0 1 1 1 0 0 1   ♦ Discard the carry
```

The MSB is 1, the answer is negative, and it is in 2's complement form.

Result = –2's complement of (10111001)
= –1's complement of (10111001) + 1
= –(01000110) + 1
= –(01000111) = –71

2.8 ARITHMETIC OVERFLOW

When we add two positive or negative binary numbers and the result exceeds the original length of the number, it is known as *overflow result*. Overflow causes a sign change. Let us consider the examples of the sum of two positive binary numbers as Case 1 and the sum of two negative binary numbers as Case 2.

Case 1: Sum of two positive numbers

Consider the sum of +125 and +75

```
  +125        0 1 1 1 1 1 0 1
  +  75       0 1 0 0 1 0 1 1
              1 1 1 1 1 1 1
-------     -----------------
  +200       1 1 0 0 1 0 0 0
-------     -----------------
```

As the decimal sum of +125 and +75 is +200; the length of the number is 8-bit, the result is 9-bit and an overflow occurs. This overflow changes the sign of the result and the answer is wrong. Result shows that the sum of two positive numbers is negative, which is wrong.

Case 2: Sum of two negative numbers

Consider the sum of –61 and –43

```
  –61         1 0 1 1 1 1 0 1
  –43         1 0 1 0 1 0 1 1
                1 1 1 1 1 1
-------     -----------------
  –104       1 0 1 1 0 1 0 0 0
-------     -----------------
```

As the decimal sum of –61 and –43 is –104, the length of the number is 8-bit, the result is 9-bit and an overflow occurs.

2.9 CODES

Codes are the representation of information in a particular format. The informatio may include numbers, alphabets, and symbols, which man and machine co recognize. All the computer systems understand only the machine code; the ba property of machine code is binary in nature. Different codes are used to store a transmit the data efficiently.

2.9.1 Classification of Codes

Computers recognize binary data. The digital data as an input to the machine conta alphabets, numbers, punctuations, and special characters. These numbers, alphabe punctuations and control characters should be represented by unique combinatio of 1 and 0. These combinations of 1 and 0 are referred as the code for informatio

When the data is transmitted over a long distance, it is transmitted in terms *code-words*. During the transmission process, errors may be introduced. To dete and correct the errors, special codes are used in digital communication.

The commonly used binary codes are classified as:

1. Weighted codes
2. Self-complementary codes
3. Unit distance codes
4. Alphanumerical codes
5. Cyclic codes
6. Error detecting and correcting codes

Weighted codes

In weighted codes, the weight of a digit or a bit depends on its position. For exampl in decimal code 589, the weight of 5 is 500, weight of 8 is 80, and weight of 9 is 9 Similarly in binary code, the value of 1 depends on its position. Binary, BCD, 8-4-2- and 2-4-2-1 are the examples of weighted codes which will be discussed in th next section.

Self-complementary codes

Excess-3 code is the example of self-complementary code. In this code, the 1' complement of the excess-3 code is the excess-3 code for the 9's complement o the corresponding decimal number. For example, the excess-3 code for 2 is 0101 the 1's complement of 0101 is 1010. It is the excess-3 code of 7 and 7 is the 9' complement of 2. Excess-3 code will be discussed in detail in the next section.

Unit distance codes

The name itself indicates that there is unit distance between two consecutive code That is, the bit patterns for two consecutive numbers differ in only one bit position Gray code is an example of unit distance code. Gray code will be discussed in detai in the next section.

Alphanumerical codes

The binary codes of alphabets, numbers, and special symbols are known as alphanumerical codes. ASCII (American Standard Code for Information Interchange) and EBCDIC (Extended Binary Coded Decimal Interchange Code) are commonly used alphanumerical codes. ASCII and EBCDIC codes will be discussed in detail in the next section.

Error detecting and correcting codes

When information in digital form is transmitted to a long distance, errors may get introduced and 1 becomes 0 and 0 becomes 1. Special codes are used to detect and correct such errors. Parity and Hamming codes are commonly used for error detection and correction.

2.9.2 Binary Coded Decimal Code (BCD CODE)

It is a 4-bit binary coded decimal number. Each digit of the decimal number is represented by four-bits, the digit 0 in BCD is represented as 0000 and the digit 9 is represented as 1001. The BCD codes of 0 to 9 are given in Table 2.6.

Table 2.6 BCD codes

Decimal Digit	*BCD Code*			
	B_3	B_2	B_1	B_0
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

The codes, 1010, 1011, 1100, 1101, 1110, and 1111 are invalid. The weights of B_3, B_2, B_1, and B_0 are 8, 4, 2, and 1. Hence, BCD code is also known as 8-4-2-1 code.

An N digit decimal number is represented by $4 \times N$ bits in BCD code.The BCD code of $(15)_{10}$ is 00010101, and the binary code of $(15)_{10}$ is $(1111)_2$. The BCD code of $(15)_{10}$ is eight bits and the binary code of $(15)_{10}$ is four bits; it shows that the BCD code is not efficient as compared to binary. The BCD code requires more space and time to transmit the information. The arithmetic of BCD code is also complex. A few examples are given below.

(i) Decimal 2 6
BCD eqivalent 0010 0110

(ii) Decimal 8 6 7

BCD arithmetic

BCD Addition In BCD addition, each digit of a decimal number is converted into its 4-bit binary equivalent and the addition of two BCD codes is carried out using the rules of binary addition.

After the addition of two BCD codes, the result may be an invalid or a valid BCD. If the result is invalid, it is converted into a valid BCD by adding $(0110)_2$ i.e. $(6)_{10}$ for the 4-bit addition. If carry is generated after the addition of $(0110)_2$, it is added to the next bit.

Algorithm for BCD addition:

1. Convert the decimal numbers into their equivalent BCD codes.
2. Add the BCD numbers using the rules of binary addition.
3. Check the result. If it is valid (less than or equal to 9), no correction is needed and if the result is invalid (greater than 9), go to the next step, otherwise stop.
4. Add 0110 to the 4-bit sum to get the correct result.

(i) Addition of 5 and 3 in BCD

```
   5        0101   ♦ BCD code of 5
  +3        0011   ♦ BCD code of 3
  ---       ----
   8         111   ♦ Carry
            1000   ♦ BCD code of 8
```

The addition of two BCD numbers, 0101 and 0011, is carried out using binary addition and the sum, 1000, is obtained which is the BCD code for 8.

(ii) Addition of 8 and 5 in BCD

```
   8        1000   ♦ BCD code of 8
  +5        0101   ♦ BCD code of 5
  ---       ----
  13        1101   ♦ Invalid BCD code
```

$(1101)_2$ is the binary equivalent of 13 in decimal. It is an invalid BCD code. It can be corrected by the addition of (0110) to the invalid BCD result.

```
   1101   ♦ Invalid BCD
  +0110
  ----
      1   ♦ Carry
  ----
  00010011   ♦ BCD code of 13
```

(iii) Addition of 7 and 9 in BCD

```
   7        0111   ♦ BCD code of 7
  +9        1001   ♦ BCD code of 9
  ---       ----
  16         111   ♦ Carry
            ----
           10000   ♦ Invalid BCD result
```


To correct the result, add 0110

```
   10000
  + 0110
  ------
  00010110   ♦ BCD code of 16
```

Example 2.23 Perform the addition of the following numbers in BCD (8-4-2-1):

(i) 8 + 7 (ii) 3 + 4 (iii) 8 + 9

Solution

(i)
```
   8        1000
  +7       +0111
  ---      ----
  15       1111   ♦ Invalid BCD code
          +0110   Add 6
          ------
          00010101   ♦ BCD code of 15
```

(ii)
```
   3        0011
  +4       +0100
  ---      ----
   7       0111
```

(iii)
```
   8        1000
  +9       +1001
  ---      ----
  17       10001   ♦ Invalid BCD code
          +0110   Add 6
          ------
          00010111   ♦ BCD code of 17
```

Example 2.24 Perform the addition of the following numbers in BCD (8-4-2-1):

(i) 15 + 24 (ii) 19 + 22 (iii) 91 + 81

Solution

(i)
```
   15       0001   0101   ♦ BCD code of 15
  +14        001   0100   ♦ BCD code of 14
  ---        ---   ----
  29           1      1   ♦ Carry
              ----   ----
              0010   1001   ♦ BCD code of 29
```

(ii)
```
   19       0001   1001   ♦ BCD code of 19
  +22      +0010   0010   ♦ BCD code of 22
  ---      ----   ----
  41        0011   1011   ♦ Invalid BCD code
           +      0110   ♦ Add 6
            ---   ----
            111     11   ♦ Carry
           ----   ----
           0100   0001   ♦ BCD code of 41
```

(iii)

91	1001	0001	♦ BCD code of 91
+ 81	+1000	0001	♦ BCD code of 81
172		1	♦ Carry
	10001	0010	♦ Second digit is invalid BCD
	+0110	0000	♦ Add 6 to 2 second digit
	00010111	0010	♦ BCD code of 172

Example 2.25 Perform the addition of the following numbers in (8-4-2-1)

(i) 191 + 171 (ii) 917 + 215

Solution

(i)

191	0001	1001	0001 ♦ BCD code of
+ 171	+0001	0111	0001 ♦ BCD code of
362	11	111	1 ♦ Carry
	0011	0000	0010

Since carry is propagated from the second digit, the second digit is invalid BCD. Add 6 to the second digit to correct the result.

	0011	0000	0010
	+	0110	♦ Add 6
	0011	0110	0010 ♦ BCD code of

(ii)

917	1001	0001	0111	♦ BCD code of
+215	+0010	0001	0101	♦ BCD code of
1132		1	111	♦ Carry
	1011	0010	1100	♦ BCD code

The first and third digits are invalid BCD. Add 6 to both the digits.

	1011	0010	1100	
	+0110		0110	♦ Add 6
	11		1	♦ Carry
1	0001	0010	10010	♦ Propagate the
0001	0001	0011	0010	♦ BCD code of

BCD Subtraction The subtraction of numbers is nothing but an addition of second negative number with the first number, i.e.,

$$A - B = A + (-B)$$

In other words, the subtraction of numbers is the addition of a signed num The addition of signed BCD numbers can be performed using 9's or 10's comple The negative BCD number can be expressed by the 9's or 10's complement o BCD number. The subtraction of two BCD numbers, B from A, is equivalent t addition of 9's or 10's complement of B with the BCD number A.

$$A - B = A + C$$

where C is 9's or 10's complement of B. There are two methods of BCD subtra (i) using 9's complement and (ii) using 10's complement.

(i) BCD subtraction using 9's complement Subtraction of the second number from the first number is the addition of 9's complement of the second number with the first number. The 9's complement of a decimal number is obtained by subtracting each digit of the decimal number from 9. The 9's complements of the decimal digits are given in Table 2.7.

Table 2.7 9's complement of decimal digits

Decimal Digit	*9's complement*	*Decimal Digit*	*9's complement*
0	9	5	4
1	8	6	3
2	7	7	2
3	6	8	1
4	5	9	0

Algorithm for BCD subtraction using 9's complement:

(i) Find the 9's complement of the subtractor.
(ii) Perform BCD addition of the first number with the 9's complement of the second number.
(iii) If carry is generated, then the result is positive. Add the carry to the result to get the correct result. If carry is not generated, then the result is negative and it is in 9's complement form.

Let us look at a few examples.

(a) 8 – 3

8 – 3 = 8 + (9's complement of 3)
= 8 + 6

1000	♦ BCD code of 8
+0110	♦ BCD code of 6
1110	♦ Invalid BCD
+ 110	♦ Add 6
11	
10100	
└→1	♦ Add the carry to LSB
0101	♦ BCD code of 5

(b) 4 – 1

4 – 1 = 4 + (9's complement of 1)
= 4 + 8

0100	♦ BCD code of 4
+1000	♦ BCD code of 8
1100	♦ Invalid BCD code
+110	♦ Add 6
1	
10010	
└→1	♦ Add the carry to LSB
0011	♦ BCD code of 3

(c) 3 – 8

3 – 8 = 3 + (9's complement of 8)
= 3 + 1

0011	♦ BCD code of 3
+0001	♦ BCD code of 1
11	♦ Carry
0100	♦ It is 9's complement of 5.

Carry is not generated, the result is negative.

∴ Result = –5

(d) 1 – 4

1 – 4 = 1 + (9's complement of 4)
= 1 + 5

0001	♦ BCD code of 1
+0101	♦ BCD code of 5
1	♦ Carry
0110	♦ It is 9's complement of 3.

Carry is not generated, the result is negative.

∴ Result = –3

(ii) BCD subtraction using 10's complement Subtraction of the second number from the first number is the addition of 10's complement of the second number with the first number. The 10's complement of a decimal number is obtained by adding 1 with 9's complement of the decimal number. The 9's and 10's complements of the decimal digits are given in Table 2.8.

Table 2.8 9's and 10's complements of decimal digits

Decimal digit	*9's complement*	*10's complement*	*Decimal digit*	*9's complement*	*10's complement*
0	9	10	5	4	5
1	8	9	6	3	4
2	7	8	7	2	3
3	6	7	8	1	2
4	5	6	9	0	1

Algorithm for BCD subtraction using 10's complement:

1. Find the 9's complement of the subtractor.
2. Find the 10's complement [i.e., 9's complement + 1].
3. Perform BCD addition of the first number and 10's complement of the second number.
4. If carry is not generated, then the result is negative, and it is in 10's complement form. If carry is generated, then the result is positive, discard the carry to get the correct result.

Let us look at a few examples.

(a) (8 – 3) = 8 + (10's complement of 3)
= 8 + (9's complement of 3 + 1) = 8 + 7

```
  1000   ♦ BCD code of 8
+ 0111   ♦ BCD code of 7
  1111   ♦ Invalid BCD code
+  110   ♦ Add 6
   11    ♦ Carry
 ~0101   ♦ Discard the carry
  0101   ♦ BCD code of 5
```

(b) (4 – 1) = 4 + (10's complement of 1)
= 4 + (9's complement of 1 + 1) = 4 + 9

```
  0100   ♦ BCD code of 4
+ 1001   ♦ BCD code of 9
  1101   ♦ Invalid BCD code
+  110   ♦ Add 6
   1     ♦ Carry
 ~0011   ♦ Discard the carry
  0011   ♦ BCD code of 3
```


(c) (3 – 8) = 3 + (10's complement of 8)
= (3 + 9's complement of 8 + 1) = 3 + 2

```
  0011   ♦ BCD code of 3
+ 0010   ♦ BCD code of 2
  0101   ♦ 10's complement of 5
```

Carry is not generated, hence the result is negative.
∴ Result = –5

(d) (1 – 4) = (1 + 10's complement of 4)
= 1 + (9's complement of 4 + 1) = 1 + 6

```
  0001   ♦ BCD code of 1
+ 0110   ♦ BCD code of 6
  0111   ♦ 10's complement of 3
```

Carry is not generated, the result is negative.
∴ Result = –3

Example 2.26 Perform the following subtractions of BCD numbers using 9's complement.

(a) 8 – 4 (b) 4 – 8 (c) 68 – 24 (d) 24 – 68

Solution

(a) (8 – 4) = 8 + (9's complement of 4)
= 8 + 5

```
  1000   ♦ BCD code of 8
+ 0101   ♦ BCD code of 5
  1101   ♦ Invalid BCD code
+ 0110   ♦ Add 6
   1     ♦ Carry
 10011
  └──→1  ♦ Add the carry
   11    ♦ Carry
  0100   ♦ BCD code of 4
```

(b) (4 – 8) = 4 + (9's complement of 8)
= 4 + 1

```
  0100   ♦ BCD code of 4
+ 0001   ♦ BCD code of 1
  0101   ♦ 9's complement of 4
```

Carry is not generated, the result is negative.
∴ Result = –4

(c) (68 – 24) = 68 + (9's complement of 24)
= 68 + 75

0 1 1 0	1 0 0 0	♦ BCD code of 68
+0 1 1 1	0 1 0 1	♦ BCD code of 75
1 1		♦ Carry
1 1 0 1	1 1 0 1	♦ Both the digits are invalid BCD code
+0 1 1 0	0 1 1 0	♦ Add 6 to each digit
1	1	♦ Carry
1 0 0 1 1	1 0 0 1 1	
⟶1	⟶1	♦ Add the carry
1 1	1 1	♦ Carry
0 1 0 0	0 1 0 0	♦ BCD code of 44

(d) (24 – 68) = 24 + (9's complement of 68)
= 24 + 31

0 0 1 0	0 1 0 0	♦ BCD code of 24
+0 0 1 1	0 0 0 1	♦ BCD code of 31
0 1 0 1	0 1 0 1	♦ 9's complement of 44

Carry is not generated, the result is negative.
∴ Result = –44

Example 2.27 Perform the following subtractions of BCD numbers using 10's complement.
(a) 8 – 4 (b) 4 – 8 (c) 68 – 24 (d) 24 – 68

Solution

(a) (8 – 4) = 8 + (10's complement of 4)
= 8 + (9's complement of 4 + 1)
= 8 + 6

	1 0 0 0	♦ BCD code of 8
+	0 1 1 0	♦ BCD code of 6
	1 1 1 0	♦ Invalid BCD
+	1 1 0	♦ Add 6
	1 1	♦ Carry
	✱0 1 0 0	♦ Discard the carry to get the correct result
	0 1 0 0	♦ BCD code of 4

(b) (4 – 8) = 4 + (10's complement of 8)
= 4 + (9's complement of 8 + 1)
= 4 + 2

	0 1 0 0	♦ BCD code of 4
+	0 0 1 0	♦ BCD code of 2
	0 1 1 0	♦ It is 10's complement of 4

Carry is not generated, the result is negative.
∴ Result = –4

(c) (68 – 24) = 68 + (10's complement of 24)
= 68 + (9's complement of 24 + 1)
= 68 + 76

	0 1 1 0	1 0 0 0	♦ BCD code of 68
+	0 1 1 1	0 1 1 0	♦ BCD code of 76
	1 1		♦ Carry
	1 1 0 1	1 1 1 0	♦ Invalid BCD code
+		0 1 1 0	♦ Add 6
		1 1	♦ Carry
	1 1 0 1	1 0 1 0 0	♦ Propagate carry
	1 ⟵		♦ Propagate carry
	1 1 1 0	0 1 0 0	♦ Invalid second digit
+	0 1 1 0		♦ Add 6
	1 1		♦ Carry
	✱0 1 0 0	0 1 0 0	♦ Discard the carry
	0 1 0 0	0 1 0 0	♦ BCD code of 44

(d) (24 – 68) = 24 + (10's complement of 68)
= 24 + (9's complement of 68 + 1)
= 24 + 32

0 0 1 0	0 1 0 0	♦ BCD code of 24
+0 0 1 1	0 0 1 0	♦ BCD code of 32
0 1 0 1	0 1 1 0	♦ 10's complement of 44

Carry is not generated, the result is negative, and it is in 10's complement form.

9's complement of 56 = 99 – 56 = 43

10's complement of 56 = 43 + 1 = 44

∴ Result = – 44

2.9.3 2-4-2-1 Code

This is a numeric code, where each digit of a decimal number is represented using four bits. It is a weighted code, the weight of binary symbol 1 depends on its position. The weights of b_3, b_2, b_1, and b_0 are 2, 4, 2, and 1, respectively. Thus, this code is referred to as 2-4-2-1 code.

Decimal digits and their 2-4-2-1 codes are given in Table 2.9.

Table 2.9 Decimal digits and their 2-4-2-1 codes

Decimal digit	*2-4-2-1 code*			
	b_3	b_2	b_1	b_0
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	1	0	1	1
6	1	1	0	0
7	1	1	0	1
8	1	1	1	0
9	1	1	1	1

2.9.4 4-Bit BCD Codes

There are various 4-bit codes of decimal digits. These codes are numeric codes, in which each digit of a decimal number is represented by four bits, as given in Table 2.10. These codes are weighted codes, the weight of binary symbol 1 depends on its position.

Table 2.10 Decimal digits and its 4-bit BCD codes

Decimal digit	*7421*	*6311*	*5421*	*5311*	*5211*	*4221*	*3321*
0	0000	0000	0000	0000	0000	0000	0000
1	0001	0001	0001	0001	0001	0001	0001
2	0010	0011	0010	0011	0011	0010	0010
3	0011	0100	0011	0111	0101	0011	0011
4	0100	0101	0100	0101	0111	1000	0101
5	0101	0111	1000	1000	1000	0111	1010
6	0110	1000	1001	1001	1001	1100	1100
7	0111	1001	1010	1011	1011	1101	1101
8	1000	1011	1011	1100	1101	1110	1110
9	1001	1100	1100	1101	1111	1111	1111

2.9.5 5-bit BCD Codes

Similar to the 4-bit BCD code, each digit of decimal numbers can also be represented by five bits, and hence these codes are known as 5-bit BCD codes. 63210 is an example of 5-bit code, it is used for storing the data on magnetic tape. It is a weighted code (except for decimal digit 0). Every code group has two 1s. This special characteristic of the code is useful for error detection.

Two-out-of-five code is also an example of 5-bit code. It is used in communications. It is a non-weighted code consisting of two 1s in every code group. Johnson code and 51111 codes are a few more examples of 5-bit codes. Decimal digits and their 5-bit codes are given in Table 2.11.

Table 2.11 Decimal digits and their 5-bit BCD codes

Decimal digit	*63210*	*Two-out-of-five*	*Johnson code*	*51111*
0	00110	00011	00000	00000
1	00011	00101	00001	00001
2	00101	00110	00011	00011
3	01001	01001	00111	00111
4	01010	01010	01111	01111
5	01100	01100	11111	10000
6	10001	10001	11110	11000
7	10010	10010	11100	11100
8	10100	10100	11000	11110
9	11000	11000	10000	11111

2.9.6 Biquinary Code

It is a weighted 7-bit BCD code. Each digit of a decimal number is represented by seven binary digits. These seven bits are divided into two subgroups. Group 1 has 2 bits (i.e., bit number B_6, B_5) and Group 2 has 5 bits (i.e., bit numbers B_4, B_3, B_2, B_1, B_0). Each subgroup consists of only one 1. This code is also known as 5043210 code. Decimal digits and their biquinary codes are given in Table 2.12.

Table 2.12 Biquinary code

Decimal digit	*Biquinary code*						
	5	*0*	*4*	*3*	*2*	*1*	*0*
	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	1	0	0	0	0	1
1	0	1	0	0	0	1	0
2	0	1	0	0	1	0	0
3	0	1	0	1	0	0	0
4	0	1	1	0	0	0	0
5	1	0	0	0	0	0	1
6	1	0	0	0	0	1	0
7	1	0	0	0	1	0	0
8	1	0	0	1	0	0	0
9	1	0	1	0	0	0	0

2.9.7 Excess-3 Code

It is a modified BCD code. It is obtained by adding 3 to each digit of the decimal number and then, it is represented by a 4-bit binary code.

For example, let us find out the excess-3 code of 9.

1001 ♦ BCD code of 9
0011 ♦ Add 3
11 ♦ Carry
1100 ♦ Excess-3 code of 9

or, the excess-3 code of $9 = (9 + 3)_{10} = (12)_{10} = (1100)_2$

The excess-3 codes of single decimal digits and their BCD codes are given in Table 2.13.

Table 2.13

Decimal digit	BCD Code				Excess-3 Code			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

In BCD code, the 9's complement is obtained by subtracting each digit from 9, or by adding 1010 with its complement. Whereas in excess-3 code, the 9's complement is obtained by the 1's complement of excess-3 code and hence, excess-3 code is called as *self-complementing code*.

In excess-3 code, the N-digit decimal number is represented by $(4 \times N)$ bits. For example, excess-3 code of $(15)_{10}$ is 01001000 (number of bits $= 4 \times 2 = 8$). The binary code of $(15)_{10}$ is $(1111)_2$. The excess-3 code of $(15)_{10}$ needs eight bits and the binary code of $(15)_{10}$ needs four bits only. This shows that the excess-3 code is not efficient as compared to binary. It requires more space and time to transmit the information. The arithmetic of excess-3 code is also complex.

Example 2.28 Find the BCD code and the 9's complement for the following decimal numbers.

(a) 5 (b) 27 (c) 156

Solution

(a) BCD code of 5 is (0101)

9's complement of 5 = (1's complement of 5) + 1010

```
  1010   ♦ 1's complement of BCD 5
 +1010
    1
 ----
 ✗0100   ♦ Discard the carry
  0100   ♦ 9's complement of BCD 5
 ----
```

(b) BCD code of 27 is (00100111)

9's complement of 27 = (1's complement of 27) + 1010

= (11011000) + 1010


```
  1101    1000   ♦ 1's complement of BCD 27
 +1010    1010
 ----    ----
 ✗0111   ✗0010   ♦ Discard the carry
  0111    0010   ♦ 9's complement of BCD 27
 ----    ----
```

(c) BCD code of 156 is (000101010110)

9's complement of 156 = 1's complement of 156 + 1010

= (000101010110) + 1010

```
  1110    1010    1001   ♦ 1's complement of BCD 156
 +1010    1010    1010
 ----    ----    ----
 ✗0100   ✗0100   ✗0011   ♦ Discard the carry
  1000    0100    0011   ♦ 9's complement of BCD 156
 ----    ----    ----
```

Example 2.29 Find the excess-3 code and its 9's complement for the following decimal numbers:

(a) 4 (b) 56 (c) 812

Solution

(a) $(4)_{10}$

```
  0100   ♦ BCD code of 4
+ 0011   ♦ Add 3
 ----
  0111   ♦ Excess-3 code of 4
 ----
```

9's complement of 4 = 1's complement of (0111)

= 1000

1000 is the excess-3 code of 5.

(b) $(56)_{10}$

```
  0101    0110   ♦ BCD code of 56
  0011    0011   ♦ Add 3 to each digit
  111.      11   ♦ Carry
 ----    ----
  1000    1001   ♦ Excess-3 code of (56)
 ----    ----
```

1000 is the excess-3 code of 5 and 1001 is the excess-3 code of 6.

9's complement of 56 = 1's complement of (10001001)

= 01110110

(c) $(812)_{10}$

```
BCD code of 812      → 1000    0001    0010
Add 3 to each digit  → 0011    0011    0011
                               11       1
                       ------------------------
Excess-3 code of 812 → 1011    0100    0101
                       ------------------------
```

9's complement of 812 = 1's complement of (1011 0100 0101)

= 0100 1011 1010

0100 is the excess-3 code of 1, 1011 is the excess-3 code of 8, and 1010 is the excess-3 code of 7.

2.9.8 Gray Code

Gray code is a 4-bit numeric code. Decimal numbers, 0 to 15, are represented by 4-bit binary codes. This code is also known as *unit distance code* because two consecutive codes differ in only one bit position.

Gray code is an unweighted code, the bit positions in the code group do not have any specific weight assigned to them. Because of this, the Gray code is not suitable for arithmetic operations, but it finds application in analog-to-digital converters. The Gray code representation for the decimal numbers, 0 to 15, is given in Table 2.14.

Table 2.14 Gray code representation of decimal numbers

Decimal digit	Gray code				Decimal digit	Gray code			
	G_3	G_2	G_1	G_0		G_3	G_2	G_1	G_0
0	0	0	0	0	8	1	1	0	0
1	0	0	0	1	9	1	1	0	1
2	0	0	1	1	10	1	1	1	1
3	0	0	1	0	11	1	1	1	0
4	0	1	1	0	12	1	0	1	0
5	0	1	1	1	13	1	0	1	1
6	0	1	0	1	14	1	0	0	1
7	0	1	0	0	15	1	0	0	0

If we examine the Gray code for each decimal number, one can notice that only one bit of the Gray code changes for two consecutive digits. For example, if the decimal digit changes from 6 to 7, then the Gray code changes from 0101 to 0100; there is change in leftmost bit only. This is the principle characteristic of the Gray code.

Binary-to-Gray code conversion

It is difficult to remember the Gray code of each decimal number as compared to binary code. Hence, write the binary code of the decimal number and then convert it into Gray code by using binary-to-Gray code conversion technique. The generalized method of conversion for N-bits binary to Gray is as follows:

$$G_{N-1} = B_{N-1} \quad (2.8)$$

$$G_N - 2 = B_N - 1 \oplus B_N - 2 \quad (2.9)$$

$$G_N - 3 = B_N - 2 \oplus B_N - 1 \quad (2.10)$$

$$\vdots$$

$$G_2 = B_3 \oplus B_2 \quad (2.11)$$

$$G_1 = B_2 \oplus B_1 \quad (2.12)$$

$$G_0 = B_1 \oplus B_0 \quad (2.13)$$

For a 4-bit binary ($N=4$) code, the Gray code is obtained by using the following expressions:

$$G_3 = B_3 \quad (2.14)$$

$$G_2 = B_3 \oplus B_2 \quad (2.15)$$

$$G_1 = B_2 \oplus B_1 \quad (2.16)$$

$$G_0 = B_1 \oplus B_0 \quad (2.17)$$

Example 2.30 Find the Gray codes for the following decimal numbers:
(a) 15 (b) 14 (c) 8

Solution

(a) $(15)_{10} = (1111)_2$

$B_3 = 1, B_2 = 1, B_1 = 1, B_0 = 1$

$G_3 = B_3 = 1$

$G_2 = B_3 \oplus B_2 = 1 \oplus 1 = 0$

$G_1 = B_2 \oplus B_1 = 1 \oplus 1 = 0$

$G_0 = B_1 \oplus B_0 = 1 \oplus 1 = 0$

Gray code of 15 = 1000

(b) $(14)_{10} = (1110)_2$

$B_3 = 1, B_2 = 1, B_1 = 1, B_0 = 0$

$G_3 = B3 = 1$

$G_2 = B_3 \oplus B_2 = 1 \oplus 1 = 0$

$G_1 = B_2 \oplus B_1 = 1 \oplus 1 = 0$

$G_0 = B_1 \oplus B_0 = 1 \oplus 0 = 1$

Gray code of 14 = 1001

(c) $(8)_{10} = (1000)_2$

$B_3 = 1, B_2 = 0, B_1 = 0, B_0 = 0$

$G_3 = B_3 = 1$

$G_2 = B_3 \oplus B_2 = 1 \oplus 0 = 1$

$G_1 = B_2 \oplus B_1 = 0 \oplus 0 = 0$

$G_0 = B_1 \oplus B_0 = 0 \oplus 0 = 0$

Gray code of 8 = 1100

Example 2.31 Find the Gray codes for the following binary numbers.
(a) 11001100 (b) 01011110

Solution

(a) 11001100

$B_7 = 1, B_6 = 1, B_5 = 0, B_4 = 0,$

$B_3 = 1, B_2 = 1, B_1 = 0, B_0 = 0$

$G_7 = B_7 = 1$

$G_6 = B_7 \oplus B_6 = 1 \oplus 1 = 0$

$G_5 = B_6 \oplus B_5 = 1 \oplus 0 = 1$

$G_4 = B_5 \oplus B_4 = 0 \oplus 0 = 0$

$G_3 = B_4 \oplus B_3 = 0 \oplus 1 = 1$

$G_2 = B_3 \oplus B_2 = 1 \oplus 1 = 0$

$G_1 = B_2 \oplus B_1 = 1 \oplus 0 = 1$

$G_0 = B_1 \oplus B_0 = 0 \oplus 0 = 0$

The gray code is 10101010.

(b) 01011110

$B_7 = 0, B_6 = 1, B_5 = 0, B_4 = 1,$

$B_3 = 1, B_2 = 1, B_1 = 1, B_0 = 0$

$G_7 = B_7 = 0$

$G_6 = B_7 \oplus B_6 = 0 \oplus 1 = 1$

$G_5 = B_6 \oplus B_5 = 1 \oplus 0 = 1$

$G_4 = B_5 \oplus B_4 = 0 \oplus 1 = 1$

$G_3 = B_4 \oplus B_3 = 1 \oplus 1 = 0$

$G_2 = B_3 \oplus B_2 = 1 \oplus 1 = 0$

$G_1 = B_2 \oplus B_1 = 1 \oplus 1 = 0$

$G_0 = B_1 \oplus B_0 = 1 \oplus 0 = 1$

The gray code is 01110001.

Example 2.32 Find the Gray codes for: (a) the BCD code (b) the 4-bit binary code.

Solution :

(a) The BCD is a 4-bit binary coded decimal number. The 4-bit binary represents each digit of the decimal number. The BCD codes and Gray codes of the decimal numbers are given in Table 2.15.

Table 2.15 BCD code and its Gray code

Decimal digit	BCD dode				Gray code			
	B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1

(b) The 4-bit binary code represents the decimal numbers, 0 to 15. The 4-bit binary codes and Gray codes of the decimal numbers are given in Table 2.16.

Table 2.16 4-bit binary code and its Gray code

Decimal digit	BCD dode				Gray code			
	B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Gray-to-Binary code conversion

In a telephone communication system, the message may be transmitted in the form of Gray code. But at the receiving end, it becomes difficult to decode the message. Hence, it is necessary to convert the Gray code into its equivalent binary by using Gray-to-Binary code converter.

The generalized method of conversion for N bits Gray-to-Binary code is as follows

$$B_{N-1} = G_{N-1} \quad (2.18)$$

$$B_N - 2 = B_N - 1 \oplus G_N - 2 \quad (2.19)$$

$$B_N - 3 = B_N - 2 \oplus G_N - 3 \quad (2.20)$$

$$\vdots$$

$$B_2 = B_3 \oplus G_2 \quad (2.21)$$

$$B_1 = B_2 \oplus G_1 \quad (2.22)$$

$$B_0 = B_1 \oplus G_0 \quad (2.23)$$

For a 4-bit Gray code ($N=4$), the Binary code is obtained by using the following expressions:

$$B_3 = G_3 \quad (2.24)$$

$$B_2 = B_3 \oplus G_2 \quad (2.25)$$

$$B_1 = B_2 \oplus G_1 \quad (2.26)$$

$$B_0 = B_1 \oplus G_0 \quad (2.27)$$

Example 2.33 Find the Binary codes for the following Gray codes:
(a) 1000 (b) 1010 (c) 1111

Solution

(a) Given Gray code is 1000.

$G_3 = 1, G_2 = 0, G_1 = 0, G_0 = 0$

$B_3 = G_3 = 1$

$B_2 = B_3 \oplus G_2 = 1 \oplus 0 = 1$

$B_1 = B_2 \oplus G_1 = 1 \oplus 0 = 1$

$B_0 = B_1 \oplus G_0 = 1 \oplus 0 = 1$

The Binary code is 1111.

(b) Given Gray code is 1010.

$G_3 = 1, G_2 = 0, G_1 = 1, G_0 = 0$

$B_3 = G_3 = 1$

$B_2 = B_3 \oplus G_2 = 1 \oplus 0 = 1$

$B_1 = B_2 \oplus G_1 = 1 \oplus 1 = 0$

$B_0 = B_1 \oplus G_0 = 0 \oplus 0 = 0$

The Binary code is 1100.

(c) Given Gray code is 1111.

$G_3 = 1, G_2 = 1, G_1 = 1, G_0 = 1$

$B_3 = G_3 = 1$

$B_2 = B_3 \oplus G_2 = 1 \oplus 1 = 0$

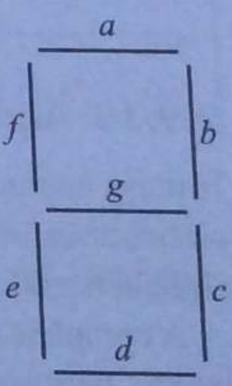
$B_1 = B_2 \oplus G_1 = 0 \oplus 1 = 1$

$B_0 = B_1 \oplus G_0 = 1 \oplus 1 = 0$

The Binary code is 1010.

2.9.9 Seven-Segment Code

In most of the applications, seven segments are used to display the result. Binary or decimal digits cannot be dispalyed directly over the seven segments. The seven-segment display includes seven segments, which are refered as a, b, c, d, e, f and g. The segment will be ON or OFF according to the input data and the type of segments. The seven segments are of two types: (a) *common anode type* and (b) *common cathode type*. The seven segments are shown in the adjacent figure.



Common anode type

In a common anode type seven segment, the anode of light emmiting diodes (segment) are connected together to V_{cc}. When the input is 0, the coresponding segment is ON; and when the input is 1, the corresponding segment is OFF. The seven-segment code for common anode type is given in Table 2.17.

Table 2.17 Seven-segment code of decimal digits and hexadecimal equivalent

Decimal digit	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*Hex code*
0	0	0	0	0	0	0	1	1	03
1	1	0	0	1	1	1	1	1	9F
2	0	0	1	0	0	1	0	1	25
3	0	0	0	0	1	1	0	1	0D
4	1	0	0	1	1	0	0	1	99
5	0	1	0	0	1	0	0	1	49
6	0	1	0	0	0	0	0	1	41
7	0	0	0	1	1	1	1	1	1F
8	0	0	0	0	0	0	0	1	01
9	0	0	0	0	1	0	0	1	09

Common cathode

In a common cathode type seven segment, the cathode of light emmiting diodes (segment) are connected together to ground. When the input is 1, the corresponding segment is ON; and when the input is 0, the coresponding segment is OFF. The seven-segment code for common cathode type is given in Table 2.18.

Table 2.18 Seven-segment code of decimal digits and hexadecimal equivalent

Decimal digit	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*Hex code*
0	1	1	1	1	1	1	0	0	FC
1	0	1	1	0	0	0	0	0	60
2	1	1	0	1	1	0	1	0	DA
3	1	1	1	1	0	0	1	0	F2
4	0	1	1	0	0	1	1	0	66
5	1	0	1	1	0	1	1	0	B6
6	1	0	1	1	1	1	1	0	BE
7	1	1	1	0	0	0	0	0	E0
8	1	1	1	1	1	1	1	0	FE
9	1	1	1	1	0	1	1	0	F6

2.9.10 Alphanumeric Codes

In many applications, a computer system has to handle data consisting of numericals, alphabets, and special symbols. Therefore, it is necessary to have binary codes for alphabets and special symbols too. These codes are called *alphanumeric codes*.

A complete alphanumeric code would include 26 lower case letters, 26 uppercase letters, 10 numeric digits, 7 punctuation marks, and 20 to 40 other characters, such

as +, /, #, %, *, etc. In other words, alphanumeric codes are the binary representation of all the characters and functions that are found on a standard typewriter. ASCII code and EBCDIC are the examples of *alphanumeric codes*.

ASCII code

American Standard Code for Information Interchange (ASCII) is the most widely used alphanumeric code. It is used in most of the microcomputers and minicomputers and in many mainframes. It is a 7-bit code and so, it has $2^7 = 128$ possible code groups. All the keys of a keyboard are represented by 7 bit binary code. This code is used to exchange the information between input/output device and computers, and stored into the memory. The ASCII code and its octal and hexadecimal equivalents are given in Table 2.19.

Table 2.19 ASCII code and its octal and hexadecimal equivalents

Character	*7-Bit ASCII*	*Octal*	*Hex*	*Character*	*7-Bit ASCII*	*Octal*	*Hex*
A	100 0001	101	41	Y	101 1001	131	59
B	100 0010	102	42	Z	101 1010	132	5A
C	100 0011	103	43	0	101 0000	160	30
D	100 0100	104	44	1	101 0001	161	31
E	100 0101	105	45	2	101 0010	162	32
F	100 0110	106	46	3	101 0011	163	33
G	100 0111	107	47	4	101 0100	164	34
H	100 1000	110	48	5	101 0101	165	35
I	100 1001	111	49	6	101 0110	166	36
J	100 1010	112	4A	7	101 0111	167	37
K	100 1011	113	4B	8	101 1000	170	38
L	100 1100	114	4C	9	101 1001	171	39
M	100 1101	115	4D	BLANK	101 0000	040	20
N	100 1110	116	4E		101 1110	056	2E
O	100 1111	117	4F	(	101 1000	050	28
P	100 0000	120	50	+	101 1011	053	2B
Q	100 0001	121	51	$	101 0100	044	24
R	100 0010	122	52	*	101 1010	052	2A
S	100 0011	123	53	)	101 1001	051	29
T	100 0100	124	54	–	101 1101	055	2D
U	100 0101	125	55	/	101 1111	057	2F
V	100 0110	126	56	,	101 1100	054	2C
W	100 0111	127	57	=	101 1101	075	3D
X	100 1000	130	58	<RETURN>	101 1101	015	0D
				<LINEFEED>	101 1010	012	0A

EBCDIC

Extended Binary Coded Decimal Interchange Code (EBCDIC) is an alphanumeric code. This is an 8-bit code and so, it has $2^8 = 256$ possible code groups. The EBCDIC is given in Table 2.20.

Table 2.20 EBCDIC

Character	8-Bit EBCDIC	Character	8-Bit EBCDIC	Character	8-Bit EBCDIC
A	11000001	P	11010111	5	11110101
B	11000010	Q	11011000	6	11110110
C	11000011	R	11011001	7	11110111
D	11000100	S	11100010	8	11111000
E	11000101	T	11100011	9	11111001
F	11000110	U	11100100	BLANK	01000000
G	11000111	V	11100101	?	01001011
H	11001000	W	11100110	(	01001101
I	11001001	X	11100111	+	01001110
J	11010001	Y	11101000	$	01011011
K	11010010	Z	11101001	*	01011100
L	11010011	0	11110000	)	01011101
M	11010100	1	11110001	–	01100000
N	11010101	2	11110010	/	01100001
O	11010110	3	11110011	,	01101011
		4	11110100	=	01111110

2.9.11 Error-detecting Codes

In digital communication, the digital data is sent over telephone lines using different binary codes. During the transmission, because of noise signal, 0 may become 1 or 1 may become 0, and wrong information may be received at the destination. This problem of communication is overcome by using error-detecting codes. The simple error detecting codes are: (i) parity codes and (ii) block parity codes.

Parity codes

Usually ASCII code is used for sending digital data over telephone lines. The 1-bit or more than 1-bit errors may occur in transmitted data. To detect these errors, a parity bit is usually transmitted along with the data bits. At the receiving end, parity will be checked. The format of parity code word is $P\ M_{N-1}\ M_{N-2}\ M_{N-3} \ldots M_3\ M_2\ M_1\ M_0$ where P is a parity bit and M_{N-1} to M_0 are N message bits. There are two types of parity: (i) even parity and (ii) odd parity.

For an even parity code, the total number of 1s in the parity code word is even. For an odd parity code, the total number of 1s in the parity code word is odd. A 4-bit message with even parity and odd parity is given in Table 2.21.

The single parity bit code can detect single bit errors. If the error is more than 1-bi t, it cannot be detected. For example, assume the even parity code word sent by the transmitter is 10111 and the code word received by the receiver is 10011, the parity of received code word is odd, it shows that a 1-bit error is introduced over the channel. But if the receiving code word is 10001, then the parity of received code

word is even, and it shows that there is no error introduced over the channel, actually two bits are changed over the path.

Table 2.21 4-bit message with even parity and odd parity

4-bit message				Even parity code word					Odd parity code word				
M_3	M_2	M_1	M_0	P	M_3	M_2	M_1	M_0	P	M_3	M_2	M_1	M_0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	1	1	0	0	0	1	0	0	0	0	1
0	0	1	0	1	0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	1	1	1	0	0	1	1
0	1	0	0	1	0	1	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0	1	1	0	1	0	1
0	1	1	0	0	0	1	1	0	1	0	1	1	0
0	1	1	1	1	0	1	1	1	0	0	1	1	1
1	0	0	0	1	1	0	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0	1	1	1	0	0	1
1	0	1	0	0	1	0	1	0	1	1	0	1	0
1	0	1	1	1	1	0	1	1	0	1	0	1	1
1	1	0	0	0	1	1	0	0	1	1	1	0	0
1	1	0	1	1	1	1	0	1	0	1	1	0	1
1	1	1	0	1	1	1	1	0	0	1	1	1	0
1	1	1	1	0	1	1	1	1	1	1	1	1	1

The 1-bit parity code word can detect 1-bit errors, but it cannot detect the location of the error or correct the error. The circuit that generates the parity bit at the transmitting end is called a *parity generator* and the circuit that checks the parity at the receiving end is called a *parity checker*. The designs of parity generator and parity checker will be discussed in Chapter 4.

Block parity codes

Parity code is suitable to detect only 1-bit errors. Whenever the error is two or more than two bits, the parity of the message may be unchanged and the parity checker cannot detect the error. This limitation of parity code is overcome up to a certain extent in block parity codes. In this technique, the transmitter transmits the information as a block, each block consists of several binary words as shown in Fig. 2.3. Parity bits can be assigned to both rows and columns. Let us consider the block consisting of four words, each word consists of eight bits and even parity bits are assigned to the rows and columns.

	1011	0001	0
	1011	0110	1 ↑ Parity column
	1000	1000	0
	0110	1010	0
Parity row →	1110	0101	1

Fig. 2.3 Block code

The receiver circuit receives the block of data and checks the parity of each row and column. If there is no change in the parity of rows and columns, then the block is error-free. If there is a change in the parity of one row and one column, then there is a 1-bit error, and the position of error is determined by row and column. Let us consider Fig. 2.4.

1 0 1 1 0 0 0 1	0
1 0 **0** 1 0 1 1 0	1 ← Parity error
1 0 0 0 1 0 0 0	0
0 1 1 0 1 0 1 0	0
1 1 **1** 0 0 1 0 1	1
↑ Parity error	

Fig. 2.4 Block code with 1-bit error

There is a parity error in row number 2 and column number 3. The position of error is shown in Fig. 2.4 by dark shadow. This can be corrected by complementing the error bit. If there is a change of parity of more than one rows or columns, then more than 1-bit error has occurred, as shown in Fig. 2.5.

1011 0001	0
1000 0110	1
1000 1000	0
0110 1010	0
1110 0101	1
↑↑ Parity error	

Fig 2.5 Block code with more than 1-bit error

Here, the parity error is in column numbers 3 and 4. This error cannot be corrected because there is no information about parity error in rows, and hence the position of error bits cannot be located.

2.9.12 Error-correcting Codes

The error-detecting codes cannot define the position of an error and cannot correct it. Therefore, there is need for a code which can detect as well as correct the errors.

The error detection and correction capability of a code depends on the number of extra bits that are added in a message to get the code words, which increases the distance between adjacent code words. The minimum distance of coding scheme is the shortest distance between any two distinct codes. For a minimum distance d_{min}, the number of errors that can be detected is $(d_{min} - 1)$ and the number of errors that can be corrected is $(d_{min} - 1)/2$.

In error-correcting codes, the receiver circuit checks the error. If an error is present, then the receiver is capable to detect its position, which can be corrected. Linear block code, hamming code, and Binary cyclic code are the examples of error-correcting code.

Linear block code

The size of a linear code is n bits, where $n = k + r$, k is the number of message bits and r is the number of check bits. The block code is designated as $(n \times k)$ code. The receiver checks whether the error is present or not with the help of check bits.

Encoding of linear block codes Linear block codes are represented by a matrix of size $(k \times n)$ where k is the size of the message and n is the size of the linear block code $(n = k + r)$. The block code C is obtained by the multiplication of message bits and the generation matrix.

$$C = M \times G \tag{2.28}$$

While doing the matrix multiplication, MOD 2 operations are used. The generation matrix is constructed as

$$G = [I_k : A]_{k \times n} \tag{2.29}$$

where I_k is the identity matrix of order k and A is the arbitrary matrix of order $(k \times r)$.

The arbitrary matrix is selected such that the resulting code should be capable to detect and correct errors at the receiving end.

$$C = M \times G$$
$$[C_1 C_2 \ldots C_n] = [M_1 M_2 \ldots M_k] \times [I_k : A] \tag{2.30}$$

For example, consider the matrix

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

and message bits are [0 1 1].

The linear block code $C = [M] \times [G]$

$$= [M] \times [I_k : A]$$

$$= [0\ 1\ 1] \times \begin{bmatrix} 100 : 100 \\ 010 : 101 \\ 001 : 111 \end{bmatrix}$$

$$= [0 \times 1 \oplus 1 \times 0 \oplus 1 \times 0, 0 \times 0 \oplus 1 \times 1 \oplus 1 \times 0, 0 \times 0 \oplus 1 \times 0 \oplus 1 \times 1,$$
$$0 \times 1 \oplus 1 \times 1 \oplus 1 \times 1, 0 \times 0 \oplus 1 \times 0 \oplus 1 \times 1, 0 \times 0 \oplus 1 \times 1 \oplus 1 \times 1]$$
$$= [0 \oplus 0 \oplus 0, 0 \oplus 1 \oplus 0, 0 \oplus 0 \oplus 1, 0 \oplus 1 \oplus 1, 0 \oplus 0 \oplus 1, 0 \oplus 1 \oplus 1]$$
$$= [0\ 1\ 1\ 0\ 1\ 0]$$

Decoding of linear block codes The receiver circuit receives the linear block code of n bits; it does not know the transmitted message. The receiver checks whether or not an error is present in the receiving message with the help of an arbitrary matrix. The receiver circuit calculates the parity check matrix,

$$H = [A^T : I_r] \tag{2.31}$$

where I_r is the identity matrix of order r and A^T is the transpose of matrix A.

The parity check matrix, H, and the transpose of the received message are multiplied and it is decided whether the receiving message is correct or not. If the multiplication $(H \times R^T)$ is zero, then the receiving message is correct, otherwise there is an error in the receiving code.

Error correction The error correction capability of a code depends on the minimum distance. For a (6, 3) block code, minimum distance is 3, it can detect 2-bit errors and correct 1-bit errors. The correction procedure is as follows:

1. Find the syndrome matrix. It is a row matrix and is calculated as

$$S = RH^T \tag{2.32}$$

where R is the received code of order $(1 \times n)$, H^T is the transpose of parity check matrix H of order $(n \times r)$

2. Match the row matrix S with the row of H^T. The number of a row where a match occurs gives the error position.

For example, if matrix S matches with the third row of matrix H^T, then the error is at the third position of the received message.

For example, let us assume that the message received by the receiver $R = 011000$ and the matrix

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

The parity check matrix H is calculated as follows:

$$H = [A^T : I_r] = \begin{bmatrix} 111 : 100 \\ 001 : 010 \\ 011 : 001 \end{bmatrix} \qquad R = [011000]$$

$$R^T = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad H \times R^T = \begin{bmatrix} 111 : 100 \\ 001 : 010 \\ 011 : 001 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$= [0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0, 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0, 0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0]$$
$$= [010]$$

$(H \times R^T)$ is not equal to zero, which shows that an error is present in the received message. To correct the error, find the syndrome matrix. It is a row matrix and calculated as

$$S = R \times H^T = [0\,1\,1\,0\,0\,0] \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= [0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0, 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0, 0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0]$$
$$= [0\,1\,0]$$

Syndrome matrix S matches with the 5th row of H^T which shows that the error is at the 5th position in the received message R. The correct result is obtained by taking the complement of the 5th bit of the received message R.

Correct received message = 0 1 1 0 1 0

Example 2.34 Find all the code vectors for the given generator matrix of a (6,3) block code.

$$G = \begin{bmatrix} 100 & : & 011 \\ 010 & : & 101 \\ 001 & : & 110 \end{bmatrix}$$

Solution

The linear block code $C = [M] \times [G]$

$$M = [0\,0\,0],\ C = [0\,0\,0] \times \begin{bmatrix} 100 & : & 011 \\ 010 & : & 101 \\ 001 & : & 110 \end{bmatrix}$$

$$= [0 \times 1 \oplus 0 \times 0 \oplus 0 \times 0, 0 \times 0 \oplus 0 \times 1 \oplus 0 \times 0, 0 \times 0 \oplus 0 \times 0 \oplus 0 \times 1,$$
$$0 \times 0 \oplus 0 \times 1 \oplus 0 \times 1, 0 \times 1 \oplus 0 \times 0 \oplus 0 \times 1, 0 \times 1 \oplus 0 \times 1 \oplus 0 \times 0]$$
$$= [0 \oplus 0 \oplus 0, 0 \oplus 0 \oplus 0, 0 \oplus 0 \oplus 0, 0 \oplus 0 \oplus 0, 0 \oplus 0 \oplus 0, 0 \oplus 0 \oplus 0]$$
$$= [0\,0\,0\,0\,0\,0]$$

$$M = [0\,0\,1],\ C = [0\,0\,1] \times \begin{bmatrix} 100 & : & 011 \\ 010 & : & 101 \\ 001 & : & 110 \end{bmatrix} = [0\,0\,1\,1\,1\,0]$$

$$M = [0\,1\,0],\ C = [0\,1\,0] \times \begin{bmatrix} 100 & : & 011 \\ 010 & : & 101 \\ 001 & : & 110 \end{bmatrix} = [0\,1\,0\,1\,0\,1]$$

$$M = [0\ 1\ 1],\ C = [0\ 1\ 1] \times \begin{bmatrix} 100 & : & 011 \\ 010 & : & 101 \\ 001 & : & 110 \end{bmatrix} = [0\ 1\ 1\ 0\ 1\ 1]$$

$$M = [1\ 0\ 0],\ C = [1\ 0\ 0] \times \begin{bmatrix} 100 & : & 011 \\ 010 & : & 101 \\ 001 & : & 110 \end{bmatrix} = [1\ 0\ 0\ 0\ 1\ 1]$$

$$M = [1\ 0\ 1],\ C = [1\ 0\ 1] \times \begin{bmatrix} 100 & : & 011 \\ 010 & : & 101 \\ 001 & : & 110 \end{bmatrix} = [1\ 0\ 1\ 1\ 0\ 1]$$

$$M = [1\ 1\ 0],\ C = [1\ 1\ 0] \times \begin{bmatrix} 100 & : & 011 \\ 010 & : & 101 \\ 001 & : & 110 \end{bmatrix} = [1\ 1\ 0\ 1\ 1\ 0]$$

$$M = [1\ 1\ 1],\ C = [1\ 1\ 1] \times \begin{bmatrix} 100 & : & 011 \\ 010 & : & 101 \\ 001 & : & 110 \end{bmatrix} = [1\ 1\ 1\ 0\ 0\ 0]$$

Hamming code

Hamming code detects the error as well as locates the position of the error, so that the error can be corrected. It is constituted by adding parity bits in an n-bit message. The length of Hamming code is $(n + k)$, where k is the number of parity bits added in a message. The number of parity bits depends upon the number of message bits. For a n-bit message, the number of parity bits is k. The value of k must be chosen such that

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

For a 4-bit message, the number of parity bits is 3.

The location of parity bits in the Hamming code corresponds to an ascending power of two ($2^0, 2^1, 2^2, 2^3, \ldots$). The location of parity bit and message bit is given as

Bit number	B_7	B_6	B_5	B_4	B_3	B_2	B_1
Binary equivalent	111	110	101	100	011	010	001
Parity bits	-	-	-	P_3	-	P_2	P_1
Message bit	M_4	M_3	M_2	-	M_1	-	-
Hamming code word	M_4	M_3	M_2	P_3	M_1	P_2	P_1

The parity bits P_3, P_2 and P_1 depend upon M_4, M_3, M_2, M_1 message bits. The procedure to assign the value to the parity bits is as follows:

- Find the binary location of the parity bits.
- Check the position of 1 in the binary equivalent.
- Check the bit's location, which has a 1 position defined by the binary equivalent of parity bit, and assign the value

For example, assign the value to P_1:

- Binary location of the parity *b*it P_1 is 001.
- Position of 1 in the binary equivalent is b_0.
- Position of 1 at b_0 is found in bits M_4, M_2, M_1, P_1.

For even parity, Hamming code assigns the value to P_1 such that the parity of M_4, M_2, M_1, P_1 is even.

For odd parity, Hamming code assigns the value of P_1 such that the parity of M_4, M_2, M_1, P_1 is odd.

Assign the value to P_2:

- Binary location of the parity bit P_2 is 010.
- Position of 1 in the binary equivalent is b_1.
- Position of 1 at b_1 is found in bits M_4, M_3, M_1, P_2.

For even parity, Hamming code assigns the value to P_2 such that the parity of M_4, M_3, M_1, P_2 is even.

For odd parity, Hamming code assigns the value of P_2 such that the parity of M_4, M_3, M_1, P_2 is odd.

Assign the value to P_3:

- Binary location of the parity bit P_3 is 100.
- Position of 1 in the binary equivalent is b_2.
- Position of 1 at b_2 is found in bits M_4, M_3, M_2, P_3.

For even parity, Hamming code assigns the value to P_3 such that the parity of M_4, M_3, M_2, P_3 is even.

For odd parity, Hamming code assigns the value of P_3 such that the parity of M_4, M_3, M_2, P_3 is odd.

At the receiving end, the Hamming code bits are separated as message bits and parity bits. The parity of each group is checked and it assigns the value as 0 if the parity is correct, otherwise it assigns the value as 1. If the assigned values of all the groups are zero, then there is no error, otherwise there is an error. The position of an error in the receiving message is defined by the digital equivalent of (C_3, C_2, C_1), where C_1 is assigned the value of group one (M_4, M_2, M_1, P_1), C_2 is assigned the value of group two (M_4, M_3, M_1, P_2), and C_3 is assigned the value of group three (M_4, M_3, M_2, P_3).

Example 2.35 Determine the single error-detecting code for the message code 1011 for even parity.

Solution

Given message code is 1011.

(i) Find the number of parity bits (k) required

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

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

$$k = 3$$

(ii) Position of parity bit

Bit number	B_7	B_6	B_5	B_4	B_3	B_2	B_1
Binary equivalent	111	110	101	100	011	010	001
Parity bits	-	-	-	P_3	-	P_2	P_1
Message bit	M_4	M_3	M_2	-	M_1	-	-
Hamming code word	M_4	M_3	M_2	P_3	M_1	P_2	P_1

where $M_4 = 1, M_3 = 0, M_2 = 1, M_1 = 1$

(iii) Assign the value to the parity bits such that the parity of a code is even. Assign the value to P_1 such that the parity of (M_4, M_2, M_1, P_1) is even, i.e. parity of $(111\ P_1)$ is even, $P_1 = 1$. Assign the value to P_2 such that the parity of (M_4, M_3, M_1, P_2) is even, i.e. parity of $(101\ P_2)$ is even, $P_2 = 0$. Assign the value to P_3 such that the parity of (M_4, M_3, M_2, P_3) is even, i.e. parity of $(101\ P_3)$ is even, $P_3 = 0$.

The Hamming code for the message (1011) is 1010101.

Example 2.36 A transmitter uses a single error-correcting code for the message using even parity. The message received at the receiving end is 1110101. Check and correct the error.

Solution

The message at the receiving end is 1110101.

(i) The format of Hamming code is $M_4\ M_3\ M_2\ P_3\ M_1\ P_2\ P_1$

where $M_4 = 1,\ M_3 = 1,\ M_2 = 1,\ P_3 = 0,\ M_1 = 1,\ P_2 = 0,\ P_1 = 1$

(ii) Check the parity of the group and assign the value to C_3, C_2, and C_1.

Parity of (M_4, M_2, M_1, P_1) = Parity of (1 1 1 1) = even, hence $C_1 = 0$.

Parity of (M_4, M_3, M_1, P_2) = Parity of (1 1 1 0) ≠ even, hence $C_2 = 1$.

Parity of (M_4, M_3, M_2, P_3) = Parity of (1 1 1 0) ≠ even, hence $C_3 = 1$.

$C_3\ C_2\ C_1 = (1\ 1\ 0) \neq 0$ and therefore, an error is present at the position of the decimal equivalent of (1 1 0) = 6

∴ Correct message is 1010101.

Example 2.37 Determine the single error-correcting code for the message code (11010). Assume the parity code is odd.

Solution

Given message code is 11010.

(i) Find the number of parity bits (k) required

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

$2^k \geq 5 + k + 1$

$k = 4$

The length of the message = $n + k = 5 + 4 = 9$

(ii) Position of parity bit

Bit number	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1
Binary equivalent	1001	1000	0111	0110	0101	0100	0011	0010	0001
Parity bits	-	P_4	-	-	-	P_3	-	P_2	P_1
Message bit	M_5	-	M_4	M_3	M_2	-	M_1	-	-
Hamming code word	M_5	P_4	M_4	M_3	M_2	P_3	M_1	P_2	P_1

where $M_5 = 1, M_4 = 1, M_3 = 0, M_2 = 1, M_1 = 0$

(iii) Assign the value to the parity bits such that the parity of the code is odd. Assign the value to P_1 such that the parity of $(M_5, M_4, M_2, M_1, P_1)$ is odd, i.e. parity of $(1110\ P_1)$ is odd, $P_1 = 0$. Assign the value to P_2 such that the parity of (M_4, M_3, M_1, P_2) is odd, i.e. parity of $(100\ P_2)$ is odd, $P_2 = 0$. Assign the value to P_3 such that the parity of (M_4, M_3, M_2, P_3) is odd, i.e. parity of $(101\ P_3)$ is odd, $P_3 = 1$. Assign the value to P_4 such that the parity of (M_5, P_4) is odd, i.e. parity of $(1\ P_4)$ is odd, $P_4 = 0$.

The Hamming code for the message (11010) is 101011000.

SOLVED EXAMPLES

Example 2.37: Perform the following:

(a) $(5531)_8 - (3261)_8 + (100)_{10}$

(b) Convert $(19.75)_{10}$ into binary.

(c) Convert $(F4D2)_{16}$ into decimal.

(d) Determine the base x: $(211)_x = (152)_8$

(e) Encode the binary word 1010 into 7-bit even Hamming code.

Solution

(a) $(5531)_8 - (3261)_8 + (100)_{10}$

Convert octal to decimal:

$(5531)_8 = (2950)_{10}$ and $(3261)_8 = (1713)_{10}$ (refer octal to decimal conversion)

$(5531)_8 - (3261)_8 + (100)_{10} = (2950)_{10} - (1713)_{10} + (100)_{10} = (1337)_{10}$

(b) Convert to binary: $(19.75)_{10}$

The integer part of $(19.75)_{10}$ is 19 and its fractional part is 0.75.

Conversion of the integer part of decimal number:

	Quotient	*Remainder*	
$19 \div 2$	9	1	♦ LSB
$9 \div 2$	4	1	
$4 \div 2$	2	0	
$2 \div 2$	1	0	
$1 \div 2$	0	1	♦ MSB

∴ $(19)_{10} = (10011)_2$

Conversion of the fractional part of decimal number:

Multiplication	*Integer part*	
$0.75 \times 2 = 1.5$	1	♦ MSB
$0.5 \times 2 = 1.00$	1	♦ LSB

$\therefore \quad (0.75)_{10} = (0.11)_2$

$\therefore \quad (19.75)_{10} = (10011)_2 + (0.11)_2 = (10011.11)_2$

(c) Convert to decimal: $(F4D2)_{16}$

$(F4D2)_{16} = 2 \times 16^0 + 13 \times 16^1 + 4 \times 16^2 + 15 \times 16^3 = (62674)_{10}$

(d) Determine the base x: $(211)_x = (152)_8$

$$2 \times x^2 + x + 1 = 1 \times 8^2 + 5 \times 8^1 + 2 \times 8^0$$

$$\Rightarrow \quad 2 \times x^2 + x + 1 = 106$$

$$\Rightarrow \quad x = 7$$

Example 2.38 Convert the following numbers into decimal:

(i) 100100111000.0111(BCD) (ii) $(11001101.111)_2$

(iii) $(CF.5)_{16}$ (iv) $(234)_5$

Solution

(i) 100100111000.0111(BCD) = 1001 0011 1000. 0111

$= (938.7)_{10}$

(ii) $(11001101.111)_2 = 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 0 \times 2^5 + 1 \times 2^6 + 1 \times 2^7 + 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$

$= 1 + 4 + 8 + 64 + 128 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8}$

$= (205.875)_{10}$

(iii) $(CF.5)_{16} = 15 \times 16^0 + 12 \times 16^1 + 5 \times 16^{-1}$

$= (207.3125)_{10}$

(iv) $(234)_5 = 4 \times 5^0 + 3 \times 5^1 + 2 \times 5^2$

$= (69)_{10}$

Example 2.39 Write the sign magnitude 1's complement and 2's complement forms for the following decimal numbers:

(i) +9.5 (ii) –17.0

Solution

(i) $+9.5 = (+9.5)_{10} = (1001.1)_2$

Sign magnitude representation of $(+9.5)_{10} = 01001.1$

1's complement representation of $(+9.5)_{10} = 01001.1$

2's complement representation of $(+9.5)_{10} = 01001.1$

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

Sign magnitude representation of (–17) = 110001

1's complement representation of $(-17)_{10}$ = 1's complement of (+17)

= 1's complement of (010001)

= 101110

2's complement representation of $(-17)_{10}$ = 2's complement of (+17)

= 1's complement of (010001) + 1

= 101111

Example 2.40 Perform the following subtractions using 2's complement method:

(i) (0011.1001) – (0001.1110) (ii) $(3F)_{16} - (5C)_{16}$

Solution

(i) 0011.1001 – 0001.1110

1's complement representation of number = 1's complement of (0001.1110)

= 1110.0001

2's complement = 1's complement + 1

= 1's complement of (0001.1110) + 1

= 1110.0001 + 1.0

= 1111.0001

Binary addition :

```
  0011.1001   ♦ A
+ 1111.0001   ♦ 2's complement of B
-----------
 *0010.1010   ♦ Discard the carry
```

The most significant bit is 0, so the answer is positive, and it is in true form.

∴ Result = +(0010.1010)

Example 2.41 Encode the binary word into a 7-bit even Hamming code: 1010.

Solution

Given message code is 1010.

(i) Find the number of parity bit (k) required

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

$2^k \geq 4 + k + 1$ (since n is length of message code)

$k = 3$

The length of the message = $n + k = 4 + 3 = 7$

(ii) Position of parity bit

Bit number	B_7	B_6	B_5	B_4	B_3	B_2	B_1
Binary equivalent	111	110	101	100	011	010	001
Parity bits	-	-	-	P_3	-	P_2	P_1
Message bit	M_4	M_3	M_2	-	M_1	-	-
Hamming code word	M_4	M_3	M_2	P_3	M_1	P_2	P_1

where $M_4 = 1, M_3 = 0, M_2 = 1, M_1 = 0$

(iii) Assign the value to the parity bits such that parity of a code is even.

Assign the value to P_1 such that the parity of (M_4, M_2, M_1, P_1) is even, i.e. the parity of $(110\ P_1)$ is even, $P_1 = 0$. Assign the value to P_2 such that the parity of (M_4, M_3, M_1, P_2) is even, i.e. the parity of $(100\ P_2)$ is even,

$P_2 = 1$. Assign the value to P_3 such that the parity of (M_4, M_3, M_2, P_3) is even, i.e. the parity of (101 P_3) is even, $P_3 = 0$.

The Hamming code for the message (1010) is 1010010.

The most significant bit is 0, the answer is positive, and it is in true form.

∴ Result = +(0010.1010)

SUMMARY

- On the basis of radix and the number of symbols used, number systems are classified as,
  1. Decimal number system
  2. Binary number system
  3. Octal number system
  4. Hexadecimal number system
- Computer systems recognize only binary data. The information given by the user may be in the form of decimal, hexadecimal or octal number, therefore an understanding of the system operation requires the ability to convert the numbers from one number system to another.
- There are three methods used to represent sign-binary numbers:
  1. Sign-magnitude representation
  2. 1's complement representation
  3. 2's complement representation
- In sign-magnitude representation, the most significant bit represents the sign and the remaining bits represent its magnitude. When the most significant bit is 1, the sign is negative and when the most significant bit is 0, the sign is positive.
- The 1's complement of a binary number is obtained by subtracting each bit of the number from 1. The 1's complement of 0 is $1 - 0 = 1$ and the 1's complement of 1 is $1 - 1 = 0$.
- The 2's complement of a binary number is obtained by adding 1 to the 1's complement of the binary number.
- The rules of binary addition are given in the following table.

A	*B*	*Sum*	*Carry*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- The rules of binary subtraction are given in the following table.

A	*B*	*Difference*	*Borrow*
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

- Subtraction of a binary number B from another binary number A is equivalent to the addition of 1's complement of B to A, i.e. $(A - B) = A$ + 1's complement of B.

- Subtraction of a binary number B from another binary number A is equivalent to the addition of 2's complement of B and A, i.e. $(A - B) = A$ + 2's complement of B.
- The commonly used binary codes are classified as
  1. Weighted codes
  2. Self complementary codes
  3. Unit distance codes
  4. Alphanumerical codes
  5. Cyclic codes
  6. Error detecting and correcting codes
- There are two methods of BCD subtraction:
  1. BCD subtraction using 9's complement
  2. BCD subtraction using 10's complement
- The 9's complement of a decimal number is obtained by subtracting each digit of the decimal number from 9.
- The 10's complement of a decimal number is obtained by adding 1 with the 9's complement of the decimal number.
- The Gray code representation for the decimals 0 to 15 is given in the following table.

Decimal digit	*Gray code*				*Decimal digit*	*Gray code*			
	G_3	G_2	G_1	G_0		G_3	G_2	G_1	G_0
0	0	0	0	0	8	1	1	0	0
1	0	0	0	1	9	1	1	0	1
2	0	0	1	1	10	1	1	1	1
3	0	0	1	0	11	1	1	1	0
4	0	1	1	0	12	1	0	1	0
5	0	1	1	1	13	1	0	1	1
6	0	1	0	1	14	1	0	0	1
7	0	1	0	0	15	1	0	0	0

KEY TERMS AND DEFINITIONS

Number systems A number is a collection of symbols. Each symbol's value is a function of the type of system and its position. All the number systems are positional because the value of a symbol depends upon its position.

Signed binary number The numbers with a positive or negative sign are known as signed binary numbers.

1's complement representation The 1's complement of a binary number is obtained when each bit of a binary number is subtracted from 1.

2's complement representation The 2's complement of a binary number is obtained by adding 1 to the 1's complement of a binary number.

Codes It is the representation of information in a particular format. The information may include numbers, alphabets, and symbols, which men and machine can recognize. All the computer systems understand only the machine code, the basic property of machine code is binary in nature.

Binary coded decimal code (BCD code) It is the 4-bit binary coded decimal number. Each digit of a decimal number is represented by 4-bits, the digit 0 in BCD is represented as 0000 and the digit 9 is represented as 1001.

2-4-2-1 code This is a numeric code; each digit of the decimal number is represented using four bits. It is a weighted code; the weight of binary symbol 1 depends on its position.

Four bit BCD codes There are various 4-bit codes of decimal digits. These codes are numeric codes, in which each digit of the decimal number is represented by four bits.

Five bit BCD codes Similar to the 4-bit BCD code, each digit of decimal numbers can also be represented by 5-bits, such codes are known as 5-bit BCD codes. 63210 is an example of 5-bit code, it is used for storing the data on magnetic tapes. It is a weighted code (except for decimal digit 0). Every code group has two 1s. This special characteristic of the code is useful for error detection.

Biquinary code It is a weighted 7-bit BCD code. Each digit of the decimal number is represented by 7 binary digits. These 7 bits are divided into two subgroups. Group 1 has 2 bits (i.e., bit number B_6, B_5) and Group 2 has 5 bits (i.e., bit number B_4, B_3, B_2, B_1, B_0). Each subgroup consists of only one 1. This code is also known as 5043210 code.

Excess-3 code It is modified BCD code. This code is obtained by adding 3 to each BCD code or by adding 3 to each digit of the decimal number.

Gray code Gray code is a 4-bit numeric code; decimal numbers 0 to 15 are represented by 4-bit binary code. This code is also known as unit distance code because two consecutive codes differ in only 1 bit position. The Gray code is an unweighted code, the bit positions in the code group do not have any specific weight assigned to them.

Error-detecting codes In digital communication, the digital data is sent over the telephone lines using different binary codes. During the transmission, because of noise signal, 0 may become 1 or 1 may become 0 and an incorrect information may be received at the destination. This problem of communication is overcome by using error-detecting codes. The simple error-detecting codes are: (i) parity codes and (ii) block parity codes.

EXERCISES

Review Questions

1. What is the radix called in case of decimal, binary, octal, and hexadecimal number systems?
2. What is the advantage of octal and hexadecimal numbers over binaries?
3. What do you mean by sign-magnitude representation?
4. Explain the 1's complement and 2's complement representation of sign-numbers.
5. Explain the rules of 1's complement addition and subtraction with suitable example.
6. Explain the rules of 2's complement addition and subtraction with suitable example.

7. Explain the role of codes.
8. What is BCD code? What are the rules for BCD addition? Explain with suitable example.
9. What is excess-3 code? Explain the rules of addition of two excess-3 code numbers.
10. Find out the BCD, excess-3, and Gray code for the decimal numbers 0 to 9.
11. What do you mean by self-complementary code? What are the two self-complementary codes?
12. Write the Gray code for a 4-bit binary number.
13. What is Gray code? Give the advantages of Gray code over binary code.
14. Explain even and odd parity codes.
15. Write short notes on the following codes:
    (i) Excess-3 code (ii) BCD code
    (iii) Gray code (iv) Unit distance code
    (v) 5-bit code (vi) Biquinary code
16. What is a linear block code? Derive the matrix equation relating to k-message bits, n-coded message bits, and the generator matrix for a linear block code.
17. Explain with examples, how Hamming code is useful for detecting and correcting errors in digital data transmission.
18. Design a BCD to Excess-3 code converter using truth table, K-maps, and logic circuits.
19. Design the following code converters:
    (i) Binary to BCD (ii) BCD to Binary
    (iii) Binary to Excess-3 (iv) Excess-3 to Binary
    (v) BCD to Excess-3 (vi) Excess-3 to BCD
    (vii) Binary to Gray (viii) Gray to Binary

Problems

1. Perform BCD addition of the following numbers:
   (i) $24 + 36$ (ii) $08 + 10$
   (iii) $106 + 305$
2. Perform the following additions of BCD numbers:
   (i) $801 + 205$ (ii) $99 + 10$
3. Perform the following subtractions of BCD numbers using 9's complement:
   (i) $9 - 2$ (ii) $2 - 9$
   (iii) $33 - 77$ (iv) $77 - 33$
4. Perform the following subtractions of BCD numbers using 10's complement:
   (i) $7 - 3$ (ii) $3 - 7$
   (iii) $36 - 69$ (iv) $69 - 36$
5. Perform the following subtractions using (i) 1's complement method, (ii) 2's complement method:
   (i) $(11010)_2 - (10000)_2$ (ii) $(1000100)_2 - (1010100)_2$
6. Perform the following operation:
   $(738)_8 - (123)_{16} + (100)_{10}$
7. Convert the following decimal numbers to binary:
   (i) $(120625)_{10}$ (ii) $(120625)_{10}$
8. Encode the following binary words into 7-bit even parity Hamming code:
   (i) 1000 (ii) 1011

9. Perform the following:
   (i) Convert to binary : $(19.75)_{10}$ (ii) Convert to decimal : $(F4D2)_{16}$
10. Perform additions of the following BCD numbers:
   (i) 16 + 17 (ii) 20 + 30
11. Perform subtractions of the following BCD numbers:
   (i) 16 – 17 (ii) 27 – 33
12. Perform the following subtractions using 1's complement:
   (i) 23 – 44 (ii) 35 – 17
13. Perform the following subtractions using 2's complement:
   (i) 18 – 35 (ii) 42 – 27
14. Convert the following numbers from the given base to other three bases indicated:
   (i) Decimal 52.45 to binary, octal, and hexadecimal
   (ii) Hexadecimal 2ACS to binary, octal, and decimal
15. Perform the following subtraction using 1's complement and 2's complement method: 45 + (–17). Find the result in sign-magnitude.
16. Determine the value of base if
   (i) $(211)_x = (152)_8$ (ii) $(193)_x = (623)_8$
17. Using 2's complement method, perform: $(156)_{10} - (99)_{10}$
18. Carry out the following conversions:
   (i) Decimal 268.75 to binary, octal, and Hex.

Multiple Choice Questions

Select the correct option:

1. The binary equivalent of 57 is .....
   (a) 111001 (b) 01010111 (c) 101111
2. The BCD equivalent of 57 is .....
   (a) 111001 (b) 01010111 (c) 101111
3. The decimal equivalent of 10111.1100 is .....
   (a) 22.3 (b) 23.75 (c) 25
4. 0111 is an excess-3 of .....
   (a) 7 (b) 10 (c) 4
5. The code used to reduce the error due to the ambiguity in reading a binary encoder is .....
   (a) Octal code (b) Excess-3 code
   (c) Gray code (d) BCD code
6. In the 8-4-2-1 BCD code, the decimal number 125 is written as .....
   (a) 1111101 (b) 000100100101
   (c) 7D (d) none of these

Answers

1. (a) 2. (b) 3. (b) 4. (c)
5. (c) 6. (b)

Chapter 3

BOOLEAN ALGEBRA AND LOGIC GATES

Chapter Outline

- Boolean algebra and basic theorems
- Overview of the logic circuits
- Representation of logic functions on K-map
- Simplification of sum of products form and product of sums form
- Minimization of Boolean equation using Quine-McCluskey method

3.1 INTRODUCTION

Digital systems support only binary data. In binary data, HIGH is represented as logic 1 and LOW as logic 0 in positive logic; and in negative logic, HIGH is represented as logic 0 and LOW is represented as logic 1.

In most of the cases, the digital system is defined by Boolean expression or truth table or state diagram. To implement a Boolean expression with basic gates, a large number of gates and a large number of inputs of gates are needed. The requirements and inputs of the gates can be reduced by simplifying the Boolean expression. For simplification, the following three methods are used:

1. Boolean method,
2. K-map method, and
3. Quine-McCluskey method.

3.2 BOOLEAN ALGEBRA

In the modern mathematics of set theory and logic, George Boole introduced a new concept in 1854, which is known as Boolean algebra. Claude E. Shannan applied this theory to switching circuits in 1938, which has changed the entire science of electronics.

Boolean algebra defines different types of logical operations. These logical operations are realized by electronic circuitry; such electronic circuits are known as GATE. The gate has one or more inputs and one output. Both the input and output are in digital form (i.e., either logic 1 or logic 0).

In Boolean algebra, every variable is called as a Boolean variable. It takes any one of the two different values, which are associated with two events. These two

events can be recognized as TRUE or FALSE, YES or NO, ON or OFF. When these logics associate with a binary variable, then the logic is said to be binary logic. In binary logic, the events TRUE/YES/ON represent logic 1 and FALSE/NO/OFF represent logic 0.

3.2.1 Principle of Logic Circuits

Boolean algebra defines different logical operations, these operations can be realized by electronic circuits, which are known as logical circuits. The logical circuit has one or more inputs and one output. Both the input and output are in digital form, either logic 1 or logic 0. Switching circuit is one of the best examples of a logic circuit. The swiching circuit consists of a voltage source, switches, and a lamp. The lamp and switches represent Boolean variables. ON and OFF states of a lamp are referred to as logic 1 and logic 0, whereas close and open states of a switch are designated as logic 1 and logic 0 respectively.

Figure 3.1 shows a switching circuit. The switches *A* and *B* are connected in parallel and the lamp in series with the source. The switches *A* and *B* represent the input variables and the lamp is the output variable. The lamp will be ON (logic 1), when the switches A and B are closed or any one of the switch is closed; otherwise the lamp will be OFF (logic 0). The operation of the circuit is summarized in Table 3.1.

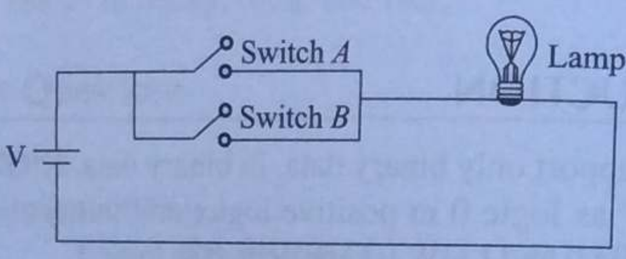


Fig. 3.1 Switching circuit

Table 3.1(a) Operation of circuit

Switch A	*Switch B*	*LAMP*
OPEN	OPEN	OFF
OPEN	CLOSE	ON
CLOSE	OPEN	ON
CLOSE	CLOSE	ON

Table 3.1(b) Truth table

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

Open = logic 0, Close = logic 1, ON = logic 1, OFF = logic 0.

3.2.2 Boolean Constants, Variables and Functions

Boolean algebra differs in a major way from ordinary algebra. Variables are represented in Boolean algebra by letters such as *A*, *B*, *C*, *X*, *Y*, *Z*. The Boolean variables are often used to represent the voltage level present on a wire or at the input/output terminals of a circuit. The Boolean value of a variable is either logic 0 or logic 1. These 0 and 1 are known as Boolean constants.

Boolean algebra is relatively easy to work with as compared to ordinary algebra because of only two values. In Boolean algebra, there are no fractions, decimals, negative numbers, square roots, cube roots, etc. In fact, in Boolean algebra, there are only three basic functions, that is (i) OR, (ii) AND, and (iii) NOT.

These basic functions are called logic operations. The circuits, which perform these logic operations, are known as gates. The gates can be constructed from diodes, transistors, and resistors connected in such a way that the circuit output is the result of basic logic operations performed on the inputs. Let us describe and analyse these basic logic operations.

OR Operation

The switching circuit shown in Fig. 3.2 has two switches *A* and *B*, which are connected in parallel and the lamp in series with the source. The lamp is either ON or OFF depending on the state of switch *A* and switch *B*.

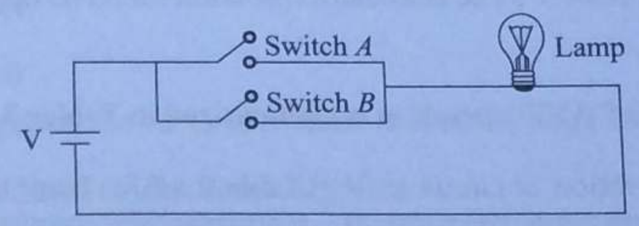


Fig. 3.2 Switching circuit for OR logic

There are four possible states of the switches:

(i) Both *A* and *B* are open.
(ii) *A* is closed and *B* is open.
(iii) *A* is open and *B* is closed.
(iv) Both *A* and *B* are closed.

When we think logically, keeping basic ideas of electrical circuits in mind, one can conclude that out of four possible combinations, the lamp will be ON for the case (ii), (iii), and (iv) and it will be OFF for case (i). In other words, the lamp is ON if either switch *A* or switch *B* is closed or both are closed. Significance of this logical statement lies in the words 'either … or' which simply represents 'OR' operation with the symbol '+'. The Boolean expression for this switching circuit is

$$Y = A \text{ OR } B$$
$$= A + B \qquad (3.1)$$

The operation of the switching circuit shown in Fig. 3.2 is summarized in Tables 3.2 (a) and (b).

Table 3.2(a) Operation of circuit

Switch A	*Switch B*	*LAMP*
OPEN	OPEN	OFF
OPEN	CLOSE	ON
CLOSE	OPEN	ON
CLOSE	CLOSE	ON

Table 3.2(b) Truth table

A	*B*	*Y = A + B*
0	0	0
0	1	1
1	0	1
1	1	1

Open = 0, Closed = 1, ON = 1, OFF = 0.

AND Operation

In the switching circuit shown in Fig. 3.2, the switches were arranged in parallel, whereas in the switching circuit of Fig. 3.3, they are arranged in series.

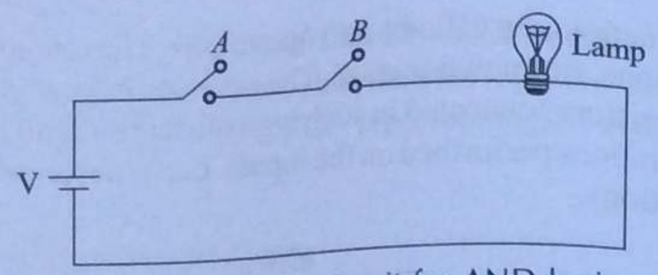


Fig. 3.3 Switching circuit for AND logic

In this circuit diagram, the lamp will be ON if and only if both the switches are closed, otherwise the lamp will be OFF. This logic is known as AND logic and it is denoted by a symbol '•'. The Boolean expression for AND operation is given as

$$Y = A \cdot B \tag{3.2}$$

The behaviour of AND operation is summarized in Tables 3.3 (a) and 3.3 (b).

Table 3.3(a) Operation of circuit

Switch A	*Switch B*	*LAMP*
OPEN	OPEN	OFF
OPEN	CLOSE	OFF
CLOSE	OPEN	OFF
CLOSE	CLOSE	ON

Table 3.3(b) Truth table

A	*B*	*Y = A • B*
0	0	0
0	1	0
1	0	0
1	1	1

Open = 0, Closed = 1, ON = 1, OFF = 0.

NOT Operation

The switching circuit shown in Fig. 3.4 has one switch and a lamp. Whenever the switch is open, the lamp is ON; and when the switch is closed, the lamp is OFF; because the current always passes through a low resistance path. This operation is known as NOT operation. As per standard notations, output Y is not equal to input A. This means that Y is equal to the complement of A. This can be expressed as

$$Y = \overline{A} \tag{3.3}$$

The operation of the circuit shown in Fig. 3.4 is summarized in Tables 3.4(a) and 3.4(b).

Table 3.4(a) Operation of circuit

Switch A	*Lamp*
OPEN	ON
CLOSE	OFF

Table 3.4(b) Truth table

A	*Y = $\overline{A}$*
0	1
1	0

Open = 0, Closed = 1, ON = 1, OFF = 0

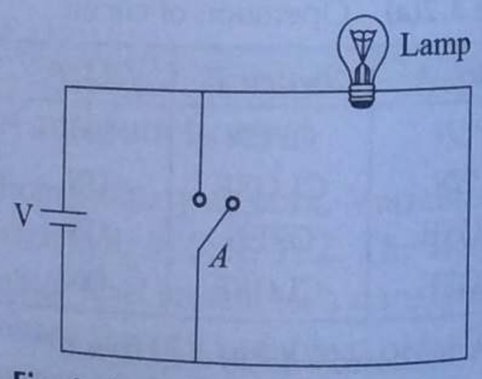


Fig. 3.4 Switching circuit for NOT logic

3.2.3 Basic Laws of Boolean Algebra

The basic laws of Boolean algebra are given as:

(i) $A + 0 = A$ (3.4)

(ii) $A + 1 = 1$ (3.5)

(iii) $A + A = A$ (3.6)

(iv) $A + \overline{A} = 1$ (3.7)

(v) $A \cdot 0 = 0$ (3.8)

(vi) $A \cdot 1 = A$ (3.9)

(vii) $A \cdot A = A$ (3.10)

(viii) $A \cdot \overline{A} = 0$ (3.11)

In Boolean algebra, the value of a variable is either logic 1 or logic 0.

Proofs of Basic Laws of Boolean Algebra

(i) $A + 0 = ?$
$0 + 0 = 0$
$1 + 0 = 1$
Hence, $A + 0 = A$

(ii) $A + 1 = ?$
$0 + 1 = 1$
$1 + 1 = 1$
Hence, $A + 1 = 1$

(iii) $A + A = ?$
$0 + 0 = 0$
$1 + 1 = 1$
Hence, $A + A = A$

(iv) $A + \overline{A} = ?$
$0 + 1 = 1$
$1 + 0 = 1$
Hence, $A + \overline{A} = 1$

(v) $A \cdot 0 = ?$
$0 \cdot 0 = 0$
$1 \cdot 0 = 0$
Hence, $A \cdot 0 = 0$

(vi) $A \cdot 1 = ?$
$0 \cdot 1 = 0$
$1 \cdot 1 = 1$
Hence, $A \cdot 1 = A$

(vii) $A \cdot A = ?$
$0 \cdot 0 = 0$
$1 \cdot 1 = 1$
Hence, $A \cdot A = A$

(viii) $A \cdot \overline{A} = ?$
$0 \cdot 1 = 0$
$1 \cdot 0 = 0$
Hence, $A \cdot \overline{A} = 0$

3.2.4 Boolean Theorems

Boolean algebra can be used to analyse a logic circuit and express its operations mathematically. It plays an important role in digital system design.

Boolean theorems help us minimize the Boolean equations. The basic Boolean theorems are listed below. They are also known as single variable Boolean theorems or basic Boolean laws.

Law 1: $A + 0 = A$
Law 2: $A + 1 = 1$
Law 3: $A + A = A$
Law 4: $A + \overline{A} = 1$
Law 5: $A \cdot 0 = 0$
Law 6: $A \cdot 1 = A$
Law 7: $A \cdot A = A$
Law 8: $A \cdot \overline{A} = 0$

Boolean multivariable theorems involve more than one variable. Boolean algebra supports the basic laws of ordinary algebra such as: (i) Commutative law, (ii) Associative law, and (iii) Distributive law.

Commutative Law

Commutative law states that the order of the variable in OR and AND operation is not important.

Law 9 : $A + B = B + A$ (3.12)

Law 10 : $A \cdot B = B \cdot A$ (3.13)

Associative Law

Associative law states that the grouping of variables in AND or OR expression does not affect the result.

Law 11: $A + (B + C) = (A + B) + C$ (3.14)

Law 12 : $A \cdot (B \cdot C) = (A \cdot B) \cdot C$ (3.15)

Distributive Law

Distributive law states that multiplying term-by-term in just the same way as in ordinary algebra can expand an expression.

Law 13: $A \cdot (B + C) = A \cdot B + A \cdot C$ (3.16)

Law 14: $(A + B) \cdot (C + D) = AC + AD + BC + BD$ (3.17)

Law 15: $AB + AC = A \cdot (B + C)$ (3.18)

Example 3.1 Prove the following Boolean theorems:

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

(b) $A + AB = A$

(c) $A(A + B) = A$

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

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

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

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

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

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

Solution

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

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

Hence proved.

Note that

$0 \cdot 0 = 0$ $\quad 0 + 0 = 0$

$0 \cdot 1 = 0$ $\quad 0 + 1 = 1$

$1 \cdot 0 = 0$ $\quad 1 + 0 = 1$

$1 \cdot 1 = 1$ $\quad 1 + 1 = 1$

(b) $A + AB = A$

L.H.S. $= A + AB$

$= A(1 + B)$

$= A$ $\quad (\because 1 + B = 1)$

$=$ R.H.S.

(c) $A(A + B) = A$

L.H.S. $= A(A + B)$

$= AA + AB$

$= A + AB$ $\quad (\because A\,A = A)$

$= A(1 + B)$

$= A$ $\quad (\because 1 + B = 1)$

$=$ R.H.S.

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

R.H.S. $= (A + B) = (A + B) \cdot 1$ $\quad (\because A + \overline{A} = 1)$

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

$= AA + A\overline{A} + AB + \overline{A}\,B$ $\quad (\because A\overline{A} = 0$ and $AA = A)$

$= A + 0 + AB + \overline{A}\,B$

$= A(1 + B) + \overline{A}\,B$ $\quad (\because 1 + B = 1)$

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

$=$ L.H.S.

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

L.H.S. $= A \cdot (\overline{A} + B) = A\overline{A} + AB$

$= 0 + AB$

$= AB =$ R.H.S.

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

L.H.S. $= AB + A\overline{B} = A(B + \overline{B})$

$= A \cdot 1$ $\quad (\because B + \overline{B} = 1)$

$= A$

$=$ R.H.S.

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

L.H.S. $= (A + B)(A + \overline{B}) = AA + A\overline{B} + AB + B\overline{B}$

$= A + A\overline{B} + AB + 0$

$= A(1 + \overline{B}) + AB$ $\quad (\because 1 + \overline{B} = 1)$

$= A + AB$

$= A(1 + B)$ $\quad (\because 1 + B = 1)$

$= A$

$=$ R.H.S.

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

R.H.S. $= (A + C)(\overline{A} + B) = A\overline{A} + \overline{A}C + AB + BC$

$= 0 + \overline{A}C + AB + BC$

$= \overline{A}C + AB + BC \cdot 1$

$$= \overline{A}C + AB + BC(A + \overline{A})$$
$$= \overline{A}C + AB + ABC + \overline{A}BC$$
$$= \overline{A}C(1 + B) + AB(1 + C)$$
$$= \overline{A}C + AB$$
$$= \text{L.H.S.}$$

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

$$\text{L.H.S.} = AB + \overline{A}C + BC$$
$$= AB + \overline{A}C + BC(A + \overline{A})$$
$$= AB + \overline{A}C + ABC + \overline{A}BC$$
$$= AB(1 + C) + \overline{A}C(1 + B)$$
$$= AB + \overline{A}C$$
$$= \text{R.H.S.}$$

Example 3.2 Simplify the following expression:

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

Solution

$$Y = A\overline{B}D + A\,\overline{B}\,\overline{D}$$
$$Y(A, B, C, D) = A\overline{B}(D + \overline{D})$$
$$= A\overline{B} \cdot 1$$
$$= A\overline{B}$$

Example 3.3 Simplify the following Boolean equations

(i) $Y(A, B, C) = ABC + A\,\overline{B} + AB\,\overline{C}$

(ii) $Y(A, B, C, D) = ACD + \overline{A}\,BCD$

Solution

(i) $Y(A, B, C) = ABC + \overline{A}B + AB\,\overline{C}$
$$= AB(C + \overline{C}) + \overline{A}B$$
$$= AB \cdot 1 + \overline{A}B$$
$$= AB + \overline{A}B$$
$$= B(A + \overline{A})$$
$$Y = B$$

(ii) $Y(A, B, C, D) = ACD + \overline{A}BCD$
$$= CD(A + \overline{A}B)$$

We have seen that $A + \overline{A}B = A + B$

$$Y(A, B, C, D) = CD(A + B)$$
$$= ACD + BCD$$

Example 3.4 Simplify the following Boolean equations.

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

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

Solution

(i) Given Boolean expression is

$$Y(A, B, C) = A\,\overline{C} + AB\,\overline{C}$$
$$= A\,\overline{C}\,[1 + B]$$
$$= A\,\overline{C} \qquad (\text{since } 1 + B = 1)$$

(ii) Given Boolean expression is

$$Y(A, B, C, D) = \overline{A}\,\overline{B}C\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$
$$= \overline{A}\,\overline{B}\,\overline{D}\,[C + \overline{C}]$$
$$= \overline{A}\,\overline{B}\,\overline{D} \qquad (\text{since } C + \overline{C} = 1)$$

3.3 OVERVIEW OF LOGIC CIRCUIT

Logic Gates

Logic gate is a digital circuit. It has one or more inputs and only one output. The input(s) and the output of a gate is logic 1 or logic 0. Here the word 'logic' is used because the values of 1 and 0 are not fixed. The gates, which perform logical OR, AND and NOT operations, are known as basic gates. Any Boolean expression can be realized using these gates.

Truth Table

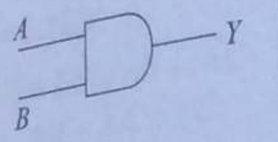
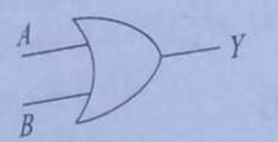
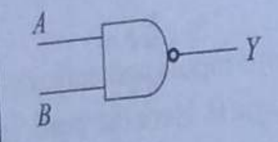
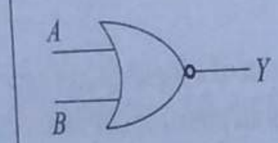
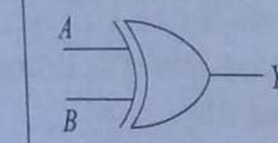
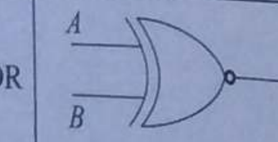
A truth table is a means to describe how the output of a logic circuit depends on the logic level present at the input(s). The table lists all possible combinations of logic levels present at the input(s) along with the corresponding output. There are four entries for the two-inputs truth table, eight entries for a three-inputs truth table, and so on. The number of input combinations will be equal to 2^N for an N-input truth table. The list of all possible input combinations follows the binary counting sequence and so it is an easy task to prepare a truth table.

Types of Gates

Listed below are the different types of gates:

1. AND gate
2. OR gate
3. NOT gate
4. NAND gate
5. NOR gate
6. EX-OR gate
7. EX-NOR gate

Table 3.5 Summary of logic gates

Gate	Symbol	Boolean expression	Truth Table
AND		$Y = A \cdot B$	A B Y: 0 0 0; 0 1 0; 1 0 0; 1 1 1
OR		$Y = A + B$	A B Y: 0 0 0; 0 1 1; 1 0 1; 1 1 1
NOT		$Y = \overline{A}$	A Y: 0 1; 1 0
NAND		$Y = \overline{A \cdot B}$	A B Y: 0 0 1; 0 1 1; 1 0 1; 1 1 0
NOR		$Y = \overline{A + B}$	A B Y: 0 0 1; 0 1 0; 1 0 0; 1 1 0
EX-OR		$Y = A \oplus B$ $= \overline{A}B + A\overline{B}$	A B Y: 0 0 0; 0 1 1; 1 0 1; 1 1 0
EX-NOR		$Y = A \odot B$ $= \overline{A}\,\overline{B} + AB$	A B Y: 0 0 1; 0 1 0; 1 0 0; 1 1 1

Boolean Expression for Logic Circuits

Any logic circuit, no matter how complex it is, can be described using Boolean expressions. The gates are the basic building blocks of the digital systems.

Examples

1. Consider the logic circuit shown in Fig. 3.5.
   The circuit consists of two AND gates, one OR gate, and the inputs to the OR gate are the outputs of AND gate$_1$ and AND gate$_2$. Since the inputs of AND gate$_1$ are A and B, the output will be $A \cdot B$; and since the inputs of AND gate$_2$ are C and D, the output will be $C \cdot D$.
   The output of OR gate is
   $$Y(A, B, C, D) = AB + CD$$

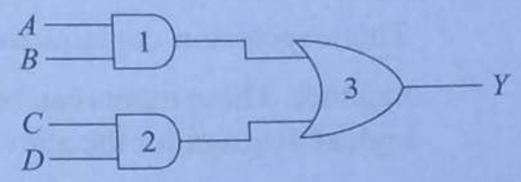


Fig. 3.5 Logic circuit

2. Consider the logic circuit shown in Fig. 3.6.
   The circuit consists of two OR gates, one AND gate, and the inputs to AND gate are the output of OR gates. Since A and B are the inputs to OR gate$_1$, the output will be $A + B$; and since C and D are the inputs to OR gate$_2$, the output will be $C + D$. The output of AND gate is $Y(A, B, C, D) = (A + B) \cdot (C + D)$.

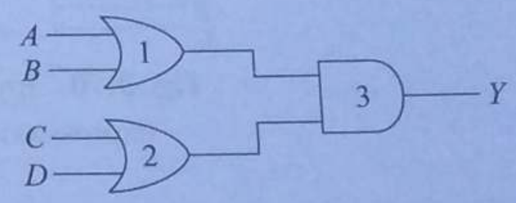


Fig. 3.6 Logic circuit

3. Consider the logic circuit shown in Fig. 3.7.
   The circuit consists of an AND and a NOT gate. Whenever a NOT gate is present in a logic circuit, its output expression is simply equal to the input expression with a bar over it.
   The output of AND gate is $A \cdot B$, and the output of the given logic diagram is $Y = \overline{A \cdot B}$.

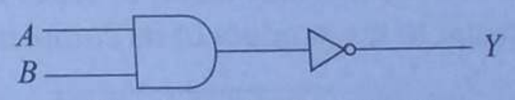


Fig. 3.7 Logic circuit

Implementing Circuits from Boolean Expressions

The operation of logic circuits is defined by Boolean expressions. A logic circuit diagram can be implemented directly from the expression.

Examples

1. Implement the Boolean expression $Y = A + B + C$.
   This equation can be implemented using a 3-input OR gate, as shown in Fig. 3.8.

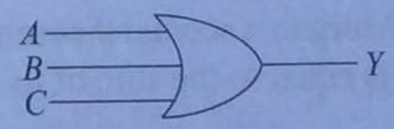


Fig. 3.8 Implementation of the Boolean expression $Y = A + B + C$

2. Implement the Boolean expression $Y = \overline{A} \cdot B$

This equation can be implemented using a 2-input AND gate with an inverter on one of the inputs as shown in Fig. 3.9.

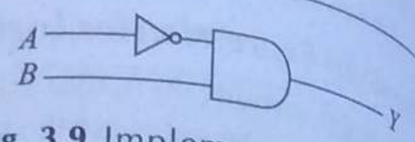


Fig. 3.9 Implementation of the Boolean expression $Y = \overline{A} \cdot B$

3. Implement the Boolean expression $Y = AB + A\overline{C} + \overline{A}BC$

This expression contains three terms (AB, $A\overline{C}$, $\overline{A}BC$) which are ORed together. These inputs can be named as $Y_1 = AB$, $Y_2 = A\overline{C}$, $Y_3 = \overline{A}BC$. The logical diagram for the above expression is shown in Fig. 3.10.

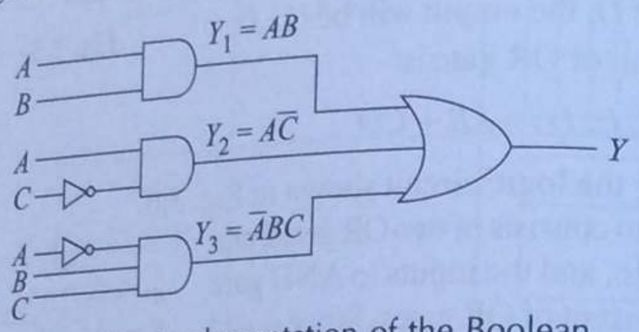


Fig. 3.10 Implementation of the Boolean expression $Y = AB + A\overline{C} + \overline{A}BC$

3.4 DeMORGAN'S THEOREMS

DeMorgan contributed the two most important theorems of Boolean algebra. These theorems play an important role in the simplification of Boolean expressions. DeMorgan's first theorem states that the complement of sum of digital signal is equal to the product of its complements.

$$\text{i.e. } \overline{A+B+C+\ldots+N} = \overline{A} \cdot \overline{B} \cdot \overline{C} \ \ldots \ \overline{N} \tag{3.19}$$

In other words, when the ORing of the variables is inverted, it is same as inverting each variable individually and then ANDing these inverted variables.

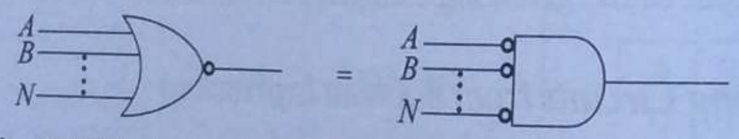


Fig. 3.11(a) Implementation of DeMorgan's first theorem

The small circle at the inputs of AND gate indicates that the input signal is inverted first and then given to the AND gate.

DeMorgan's second theorem states that the complement of a product of digital signal is equal to the sum of its complement.

$$\overline{A \cdot B \cdot C \ldots N} = \overline{A} + \overline{B} + \overline{C} + \ldots + \overline{N} \tag{3.20}$$

In other words, when the product of variables is inverted, it is same as inverting each variable individually and then ORing them.



Fig. 3.11(b) Implementation of DeMorgan's second theorem

Example 3.5 Simplify the expression

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

Solution

Given Boolean expression is

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

Using DeMorgan's theorem,

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

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

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

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

Example 3.6 Simplify the Boolean expression:

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

Solution

Given Boolean expression is

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

Using DeMorgan's theorem,

$$Y(A, B) = (\overline{A} \cdot \overline{B})(\overline{\overline{A}} \cdot \overline{\overline{B}})$$

$$= \overline{A} \cdot \overline{B} \cdot A \cdot B \quad (\text{since } \overline{A} \cdot A = 0)$$

$$= 0$$

3.5 STANDARD REPRESENTATION FOR LOGICAL FUNCTIONS

Boolean expressions are also known as logic expressions. Logical functions are expressed in terms of logical variables. The value assumed by logical variables is in binary form. The logical functions can be represented in two forms, viz.

(i) Sum of products form
(ii) Product of sums form

3.5.1 Sum of Products (SOP)

In sum of products form, the Boolean expressions are defined by the sum of product terms. This form consists of two or more AND terms that are ORed together. Each AND term consists of one or more variables appearing in either complemented or uncomplemented form.

For example,

$$Y = AB + AB\overline{C} + B \tag{3.21}$$

where Y is OR of two AND terms and input B. The first AND term contains two variables (i.e. AB); the second AND term contains three variables (i.e. $AB\overline{C}$); and the third AND term contains only one variable (i.e. B).

3.5.2 Product of Sums (POS)

In product of sums, the Boolean expressions are defined by the product of sum terms. This form consists of two or more than two OR terms, which are ANDed together. Each OR term consists of one or more variables in complemented or uncomplemented form. For example,

$$Y = (A + B)(A + B + \overline{C})(\overline{B}) \tag{3.22}$$

where Y is AND of three OR terms. The first OR term contains two variables (i.e. $A + B$); the second OR term contains three variables (i.e. $A + B + \overline{C}$); and the third OR term contains only one variable (i.e. $\overline{B}$).

3.6 MINTERM AND MAXTERM

Equation (3.21) is in the form of sum of products, and the output is a function of three variables. But the first term has two variables and the third term has only one variable.

If each term in the sum of products form contains all the variables (literals), then the expression is known as *standard sum of products form* or *canonical sum of products form*; and each individual term in standard sum of products form is called as *minterm*.

The SOP form can be converted to standard SOP form by ANDing the terms in the expression with the terms formed by ORing the variables which are not present in that term and their complements. For example,

$$Y(A, B, C) = AB + AB\overline{C} + \overline{B} \tag{3.23}$$

The first term of Eq. (3.23) has two variables and the third term has one variable only. In the first term, variable C is missing; in the third term, variables A and C are missing and hence to get the standard SOP form, multiply the first term by $(C + \overline{C})$ and the third term by $(A + \overline{A})(C + \overline{C})$

$$\begin{aligned} Y(A, B, C) &= AB(C + \overline{C}) + AB\overline{C} + \overline{B}(A + \overline{A})(C + \overline{C}) \\ &= ABC + AB\overline{C} + AB\overline{C} + A\overline{B}C + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}\,\overline{B}\,\overline{C} \\ &= ABC + AB\overline{C} + A\overline{B}C + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}\,\overline{B}\,\overline{C} \end{aligned} \tag{3.24}$$

The total minterms are six, which are:

$$ABC(111) = m_7 \qquad AB\overline{C}(110) = m_6 \qquad A\overline{B}C(101) = m_5$$

$$A\overline{B}\,\overline{C}(100) = m_4 \qquad \overline{A}\,\overline{B}C(001) = m_1 \qquad \overline{A}\,\overline{B}\,\overline{C}(000) = m_0$$

Equation (3.24) can be written as a sum of minterms, as follows:

$$Y(A, B, C) = \Sigma m(0, 1, 4, 5, 6, 7)$$

If each term of the product of sums contains all the variables, then the expression is known as *standard product of sums form* or *canonical product of sums form*, and each individual term in the standard product of sums form is called as *maxterm*.

The POS form can be converted to standard POS by ORing the terms in the expression with the terms formed by ANDing the variables which are not present in that term and their complements. For example,

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

The first term of the above equation has two variables, variable C is missing and hence to get the standard POS, the term (C) in the first term must be ORed.

$$\begin{aligned} Y(A, B, C) &= (A + \overline{B} + C\overline{C}) \cdot (A + B + \overline{C}) \\ &= (A + \overline{B} + C)(A + \overline{B} + \overline{C})(A + B + \overline{C}) \end{aligned} \tag{3.25}$$

The total maxterms are three, which are:

$$(A + \overline{B} + C)(010) = M_2$$

$$(A + \overline{B} + \overline{C})(011) = M_3$$

$$(A + B + \overline{C})(001) = M_1$$

Equation (3.25) can be written as a product of maxterms, as follows:

$$Y(A, B, C) = \Pi M(1, 2, 3)$$

For SOP, the uncomplemented variable corresponds to logic 1 and the complemented variable corresponds to logic 0. For POS, the uncomplemented variable corresponds to logic 0 and the complemented variable corresponds to logic 1. The minterm and maxterm for three variables are given in Table 3.6.

Table 3.6 Minterm and maxterm for three variables

Variables			*Minterm*	*Maxterm*
A	*B*	*C*		
0	0	0	$\overline{A}\,\overline{B}\,\overline{C} = m_0$	$A + B + C = M_0$
0	0	1	$\overline{A}\,\overline{B}C = m_1$	$A + B + \overline{C} = M_1$
0	1	0	$\overline{A}B\overline{C} = m_2$	$A + \overline{B} + C = M_2$
0	1	1	$\overline{A}BC = m_3$	$A + \overline{B} + \overline{C} = M_3$
1	0	0	$A\overline{B}\,\overline{C} = m_4$	$\overline{A} + B + C = M_4$
1	0	1	$A\overline{B}C = m_5$	$\overline{A} + B + \overline{C} = M_5$
1	1	0	$AB\overline{C} = m_6$	$\overline{A} + \overline{B} + C = M_6$
1	1	1	$ABC = m_7$	$\overline{A} + \overline{B} + \overline{C} = M_7$

Example 3.7 Find the minterms for the function

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

Solution

Given equation is

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

This equation can be converted into standard SOP form by multiplying the first term by $(C + \overline{C})$ and the second term by $(A + \overline{A})(B + \overline{B})$.

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

$A\overline{B}C(101) = m_5$ $\quad A\overline{B}\,\overline{C}(100) = m_4$ $\quad AB\overline{C}(110) = m_6$

$A\overline{B}\overline{C}(100) = m_4$ $\quad \overline{A}B\overline{C}(010) = m_2$ $\quad \overline{A}\,\overline{B}\,\overline{C}(000) = m_0$

In terms of minterms, the expression will be

$$Y(A, B, C) = \Sigma m\,(0, 2, 4, 5, 6)$$

Example 3.8 Convert the following Boolean function into standard SOP and express it in terms of minterms.

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

Solution

Given equation is

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

This equation can be converted into standard SOP by multiplying the first term by $(C + \overline{C})$, the second term by $(B + \overline{B})$, and the third term by $(A + \overline{A})$.

$$Y(A, B, C) = AB(C + \overline{C}) + A\overline{C}(B + \overline{B}) + BC(A + \overline{A})$$
$$= ABC + AB\overline{C} + AB\overline{C} + A\overline{B}\,\overline{C} + ABC + \overline{A}BC$$
$$= ABC + AB\overline{C} + A\overline{B}\,\overline{C} + \overline{A}BC$$

Each term of the standard SOP can be represented by a minterm.

$ABC(111) = m_7$ $\quad AB\overline{C}(110) = m_6$

$A\overline{B}\overline{C}(100) = m_4$ $\quad \overline{A}B\overline{C}(010) = m_2$

The expression in terms of minterms can be written as

$$Y(A, B, C) = \Sigma m\,(2, 4, 6, 7)$$

Example 3.9 Convert the following Boolean function into standard POS and express it in terms of maxterms.

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

Solution

Given equation is

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

This equation can be converted into standard POS by ORing the first term by $(C\overline{C})$, second term by $(A\overline{A})$, and third term by $(B\overline{B})$.

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

Each term of the standard POS is called as maxterm.

$(A + B + C)(000) = M_0$ $\quad (A + B + \overline{C})(001) = M_1$

$(\overline{A} + B + \overline{C})(101) = M_5$ $\quad (A + \overline{B} + C)(010) = M_2$

The expression in terms of maxterms can be written as

$$Y(A, B, C) = \Pi M(0, 1, 2, 5)$$

3.7 SIMPLIFICATION OF BOOLEAN EXPRESSION

The design of a digital system is nothing but to determine the Boolean expression from the given information and to implement the Boolean expressions by using suitable digital components [i.e. gates and flip-flops]. To reduce the requirements of hardware, it is necessary to simplify the Boolean expressions. Boolean expressions can be simplified using

(i) Algebraic simplification,
(ii) Karnaugh map method, or
(iii) Quine–McCluskey method.

3.7.1 Algebraic Method

Algebraic method is one of the simplest methods to simplify the Boolean expression. In this method, Boolean theorems are used to simplify the expression. The drawback of this method is that there is no way to distinguish whether the simplified expression is in its simplest form or it can be simplified further. For example,

$$Y(A, B, C) = AB + A\overline{B} \cdot (\overline{A} + \overline{C})$$
$$= AB + A\overline{B}\,\overline{A} + A\overline{B}\,\overline{C} \qquad (\text{since } A \cdot \overline{A} = 0)$$
$$= AB + A\overline{B}\,\overline{C}$$

Example 3.10 Simplify the logic circuit shown in Fig. 3.12.

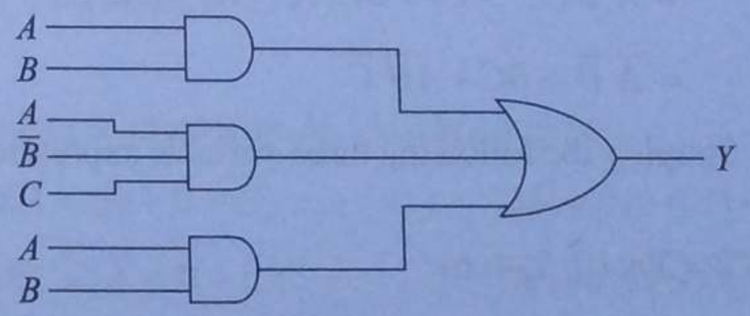


Fig. 3.12 Logic circuit

Solution

The expression for the given logic circuit is

$$Y = Y_1 + Y_2 + Y_3$$

$$Y_1 = AB,\ Y_2 = A\overline{B}C,\ Y_3 = AB$$

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

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

Example 3.11 Simplify the expression: $Y(A, B, D) = (\overline{A} + B)(A + B + D)\overline{D}$

Solution

Given Boolean expression is

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

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

$$= (\overline{A} + B)(A\overline{D} + B\overline{D}) \quad (\text{since } D\overline{D} = 0)$$

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

$$= \overline{A}B\overline{D} + AB\overline{D} + B\overline{D} \quad (\text{since } A\overline{A} = 0 \text{ and } BB = B)$$

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

$$= B\overline{D} + B\overline{D} \quad (\text{since } \overline{A} + A = 1)$$

$$= B\overline{D}$$

Example 3.12 Simplify the following three-variable expression using Boolean algebra:

$$Y(A, B, C) = \Sigma m\ (0, 1, 3, 4, 7)$$

Solution

Output Y is a summation of minterms m_0, m_1, m_3, m_4, and m_7.

$$m_0 = 000 = \overline{A}\,\overline{B}\,\overline{C} \qquad m_1 = 001 = \overline{A}\,\overline{B}C \qquad m_3 = 011 = \overline{A}BC$$

$$m_4 = 100 = A\overline{B}\,\overline{C} \qquad m_7 = 111 = ABC$$

Boolean expression in standard SOP form:

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

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

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

Example 3.13 Simplify the following three-variable expression using Boolean algebra:

$$Y(A, B, C) = \Sigma m\ (0, 2, 4, 6)$$

Solution

Output Y is a summation of minterms m_0, m_2, m_4, and m_6.

$$m_0 = 000 = \overline{A}\,\overline{B}\,\overline{C} \qquad m_2 = 010 = \overline{A}\,B\,\overline{C}$$

$$m_4 = 100 = A\overline{B}\,\overline{C} \qquad m_6 = 110 = AB\,\overline{C}$$

Boolean expression in standard SOP form:

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

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

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

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

$$= \overline{C}$$

Example 3.14 Simplify the following three-variable expression using Boolean algebra:

$$Y(A, B, C) = \Sigma m\ (0, 1, 2, 3, 4, 5, 6, 7)$$

Solution

Output Y is a summation of minterms m_0, m_1, m_2, m_3, m_4, m_5, m_6, and m_7.

$$m_0 = 000 = \overline{A}\overline{B}\overline{C} \qquad m_1 = 001 = \overline{A}\overline{B}C \qquad m_2 = 010 = \overline{A}B\overline{C}$$

$$m_3 = 011 = \overline{A}BC \qquad m_4 = 100 = A\overline{B}\overline{C} \qquad m_5 = 101 = A\overline{B}C$$

$$m_6 = 110 = AB\overline{C} \qquad m_7 = 111 = ABC$$

Boolean expression in standard SOP form:

$$Y(A, B, C) = \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + AB\overline{C} + ABC$$

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

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

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

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

$$= 1$$

Example 3.15 Simplify the following three-variable expression using Boolean algebra:

$$Y(A, B, C) = \Pi M(1, 3, 5, 7)$$

Solution

Output Y is a product of maxterms M_1, M_3, M_5, and M_7.

$$M_1 = 001 = (A + B + \overline{C}) \qquad M_3 = 011 = (A + \overline{B} + \overline{C})$$

$$M_5 = 101 = (\overline{A} + B + \overline{C}) \qquad M_7 = 111 = (\overline{A} + \overline{B} + \overline{C})$$

Boolean expression in standard POS form:

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

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

$$Y(A, B, C) = (A + A\overline{B} + A\overline{C} + AB + 0 + \overline{C}B)$$
$$(0 + \overline{A}\,\overline{B} + \overline{A}\,\overline{C} + \overline{A}B + 0 + B\overline{C} + \overline{B}\,\overline{C} + 0)$$

$$Y(A, B, C) = (A + (1 + \overline{C}) + A(\overline{B} + B) + \overline{C}B)(\overline{A}\,(\overline{B} + B) + \overline{A}\,\overline{C} + \overline{C}(B + \overline{B}))$$

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

$$Y(A, B, C) = (A + \overline{C}B)(\overline{A} + \overline{C}\,(\overline{A} + 1))$$

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

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

$$Y(A, B, C) = 0 + A\,\overline{C} + \overline{A}\,\overline{C}B + \overline{C}B$$

$$Y(A, B, C) = A\,\overline{C} + (\overline{A} + 1)\,\overline{C}B$$

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

3.7.2 Karnaugh Map Simplification

The algebraic simplification method has a drawback that there is no easy way to tell whether the simplified expression is in its simplest form or it can be simplified further. This problem of algebraic simplification method is overcome in Karnaugh map (K-map) simplification.

K-map is a graphical technique to simplify Boolean expressions. It provides a systematic method for simplifying and manipulating Boolean expressions. K-map contains cells. The number of cells in a K-map depends upon the number of variables in the Boolean expression. Actually, K-maps can be used for any number of variables. But it is used only upto six variables; beyond six variables, it becomes very difficult. Two-variable maps contain four cells, three-variable maps contain eight cells, four-variable maps contain 16 cells, and n-variable maps contains 2^n cells.

Each row and each column of the map is assigned by 0s and 1s as shown in Fig. 3.13.

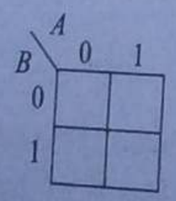


Fig. 3.13(a) Two-variable K-map

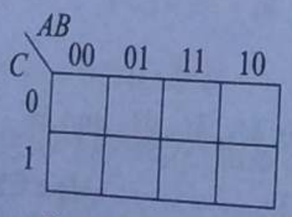


Fig. 3.13(c) Three-variable K-map

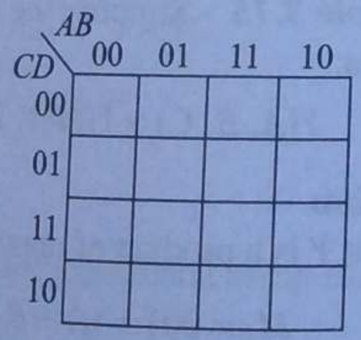


Fig. 3.13(c) Four-variable K-map

Representation of K-map

The logic function can be represented by input and output conditions (truth table) or by Boolean functions. It may be in the form of sum of products or product of sums. These Boolean functions can be simplified by using the K-map. Let us consider the structure of K-maps.

Each cell of a K-map is assigned a cell number; the cell number is calculated from columns and rows as shown in Fig. 3.14.

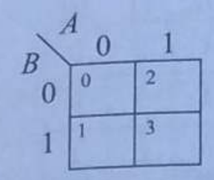


Fig. 3.14(a) Two-variable K-map with cell number

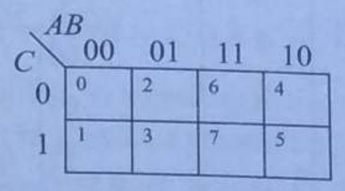


Fig. 3.14(b) Three-variable K-map with cell number

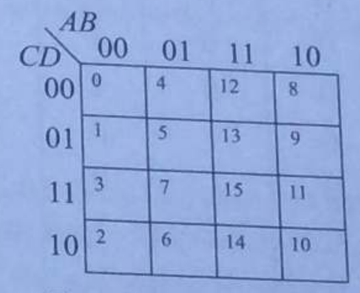


Fig. 3.14(c) Four-variable K-map with cell number

Representation of Truth Table on K-map

The truth table defines how the inputs and outputs are related. Inputs and outputs are either logic 1 or logic 0. The representation of truth table in a K-map marks the output state of a digital system in cells of K-map corresponding to input conditions. For example, the truth table of a two-variable digital system is given in Table 3.7 and its representation on K-map is shown in Fig. 3.15.

Table 3.7 Truth table of two variables

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

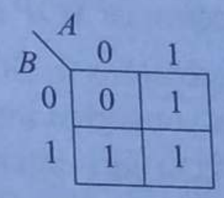


Fig. 3.15

The truth table of three-variable digital systems is given in Table 3.8 and its K-map representation is shown in Fig. 3.16.

Table 3.8 Truth table of three variables

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

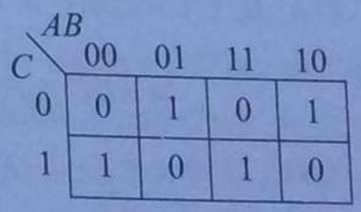


Fig. 3.16

The truth table of four-variable digital systems is given in Table 3.9 and its K-map representation is shown in Fig. 3.17.

Table 3.9 Truth table of four variables

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

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

Fig. 3.17

Representation of SOP on K-map

Boolean expressions in SOP may or may not be in a standard form. First, the expression is converted into standard SOP and then, 1s are marked in each cell corresponding to the minterm of expression and remaining cells are filled with 0s. For example, let us take a Boolean expression

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

In the first term of expression, variable C is missing; in second term, variable A is missing; and the expression is not in standard form. The standard SOP can be obtained by ANDing the first term with $(C + \overline{C})$ and the second term with $(A + \overline{A})$.

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

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

$ABC(111) = m_7$

$AB\overline{C}(110) = m_6$

$\overline{A}B\overline{C}(010) = m_2$

$\overline{A}\,\overline{B}C(001) = m_1$

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

Fig. 3.18 $Y(A, B, C) = AB + B\overline{C} + \overline{A}\overline{B}C$

The Boolean expression can be written as $Y(A, B, C) = \Sigma m(1, 2, 6, 7)$
The K-map representation of the above expression is shown in Fig. 3.18.

Representation of POS on K-map

Boolean expressions in POS may or may not be in a standard form. First, it is converted into standard POS and then, 0s are entered in each cell corresponding to the maxterms of expression and the remaining cells are filled with 1s. For example, let us take a Boolean expression

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

In the first term, variable C is missing; in the second term, variable A is missing, and the expression is not in standard product of sums form. The standard POS can be obtained by ORing the first term with $C\overline{C}$ and the second term with $A\overline{A}$.

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

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

$A + B + C(000) = M_0$ $\quad A + B + \overline{C}(001) = M_1$

$A + \overline{B} + C(010) = M_2$ $\quad \overline{A} + \overline{B} + C(110) = M_6$

The Boolean expression can be written as

$$Y(A, B, C) = \Pi M(0, 1, 2, 6)$$

The K-map representation of the above expression is shown in Fig. 3.19.

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

Fig. 3.19 $Y(A,B,C) = (A+B)(\overline{B}+C)(A+\overline{B}+C)$

Example 3.16 Represent the following Boolean function by K-map:

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

Solution
Given Boolean expression is

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

It is a four-variable Boolean expression in SOP. Variable D is missing in the first term, variable A in the second term, and variables A and C are missing in the last term. The expression is not in a standard form. The standard SOP can be obtained by ANDing the first term with $(D + \overline{D})$, the second term with $(A + \overline{A})$, and the last term with $(A + \overline{A})$ and $(C + \overline{C})$.

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

$\overline{A}\,\overline{B}CD(0011) = m_3$

$\overline{A}B\overline{C}D(0101) = m_5$

$\overline{A}BCD(0111) = m_7$

$A\overline{B}CD(1011) = m_{11}$

$ABC\overline{D}(1110) = m_{14}$

$AB\overline{C}D(1101) = m_{13}$

$ABCD(1111) = m_{15}$

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

Fig. 3.20 $Y(A, B, C, D) = ABC + \overline{B}CD + BD$

The Boolean expression can be written as

$$Y(A, B, C, D) = \Sigma m(3, 5, 7, 11, 13, 14, 15)$$

K-map representation of the expression is shown in Fig. 3.20.

Example 3.17 Represent the following Boolean expression by K-map:

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

Solution

Given Boolean expression is

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

It is a four-variable Boolean expression in POS. Variable D is missing in the first term, B is missing in the second term, and the expression is not a standard POS. The standard POS can be obtained by ORring the first term with $D\overline{D}$ and the second term with $B\overline{B}$.

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

$(A + B + \overline{C} + D)(0010) = M_2$

$(A + B + \overline{C} + \overline{D})(0011) = M_3$

$(\overline{A} + B + C + \overline{D})(1001) = M_9$

$(\overline{A} + \overline{B} + C + \overline{D})(1101) = M_{13}$

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

Fig. 3.21 $Y(A, B, C, D) = (A + B$

The Boolean expression can be written as

$$Y(A, B, C, D) = \Pi M(2, 3, 9, 13)$$

The K-map representation of the expression is shown in Fig. 3.21.

Grouping the adjacent cells

The truth table, SOP, and POS can be represented on a K-map. The minterms are marked by 1 and maxterms are marked by 0.

The simplification of logical functions with K-map is based on the principle of combining terms in adjacent cells. Two cells are said to be adjacent, if they differ in only one variable. Four cells are said to be adjacent, if they differ in two variables. Eight cells are said to be adjacent, if they differ in three variables, and so on. The simplified SOP is obtained by grouping adjacent 1s and the simplified POS is obtained by grouping adjacent 0s.

Grouping of two adjacent cells Two cells are said to be adjacent if they differ in only one variable. For a two-variable K-map, the bottom two cells are adjacent and the top two cells are adjacent. Also the left two cells and right two cells are adjacent as shown in Fig. 3.22.

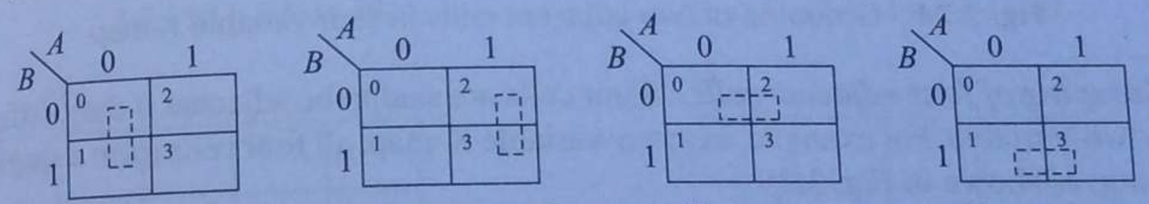


Fig. 3.22 Grouping of two adjacent cells in two-variable K-map

The groups of two adjacent cells are: 0-2, 1-3, 0-1, 2-3.

For a three-variable K-map, the possible groups of adjacent two cells are shown in Fig. 3.23.

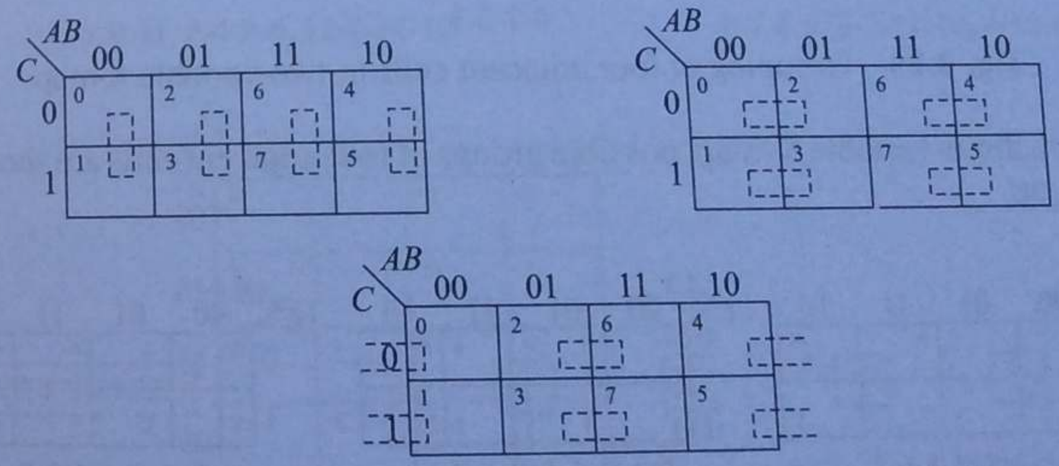
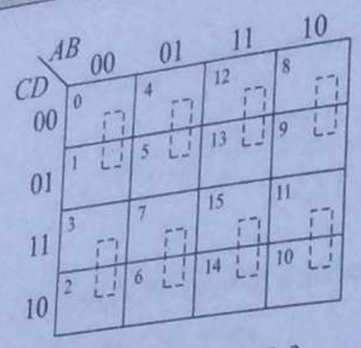


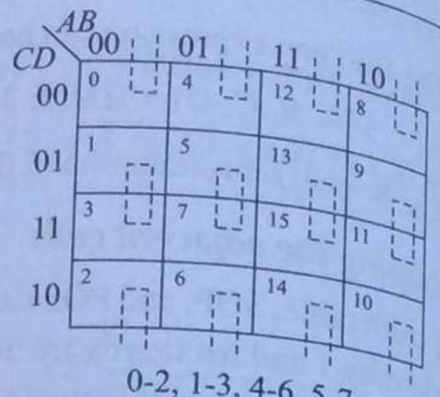
Fig. 3.23 Grouping of two adjacent cells in three-variable K-map

The groups of two adjacent cells are: 0-1, 2-3, 6-7, 4-5, 0-2, 1-3, 4-6, 5-7, 2-6, 3-7, 0-4, 1-5.

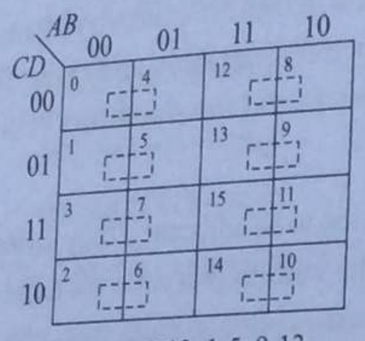
For a four-variable K-map, the possible groups of adjacent two cells are shown in Fig 3.24.



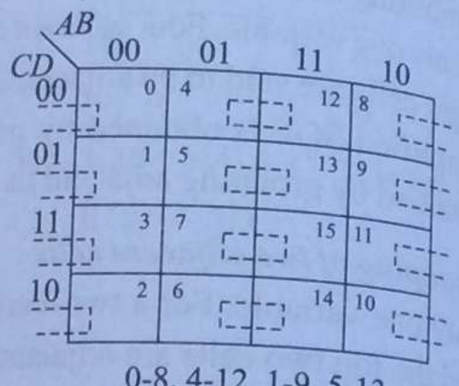
0-1, 4-5, 12-13, 8-9,
2-3, 6-7, 14-15, 10-11



0-2, 1-3, 4-6, 5-7,
12-14, 13-15, 8-10, 9-11



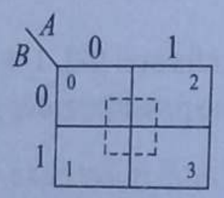
0-4, 8-12, 1-5, 9-13,
3-7, 11-15, 2-6, 10-14



0-8, 4-12, 1-9, 5-13,
3-11, 7-15, 2-10, 6-14

Fig. 3.24 Grouping of two adjacent cells in four-variable K-map

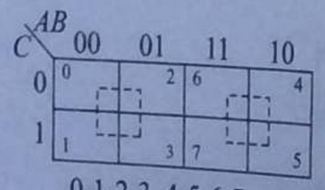
Grouping of four adjacent cells Four cells are said to be adjacent if they differ in two-variable. For example, in a two-variable K-map, all four cells form a single group as shown in Fig. 3.25.



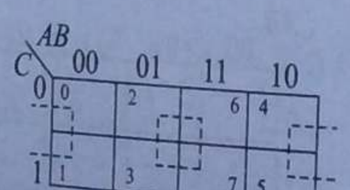
0-1-2-3

Fig. 3.25 Grouping of four adjacent cells in two-variable K-map

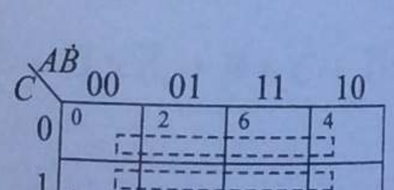
For a three-variable K-map, possible groups of four adjacent cells are shown in Fig. 3.26.



0-1-2-3, 4-5-6-7



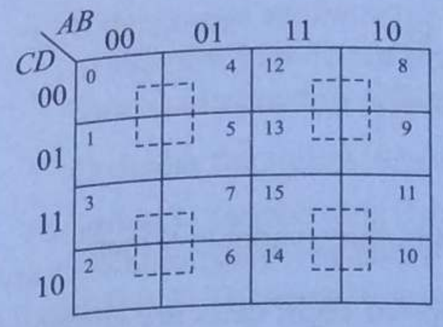
0-1-4-5, 2-3-6-7



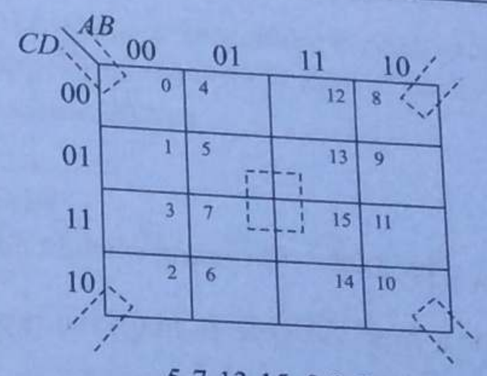
0-2-4-6, 1-3-5-7

Fig. 3.26 Grouping of four adjacent cells in three-variable K-map

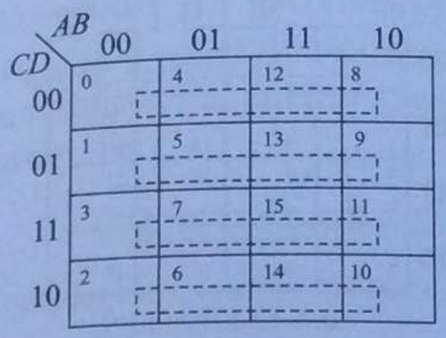
For a four-variables K-map, the possible groups of four adjacent cells are shown in Fig. 3.27.



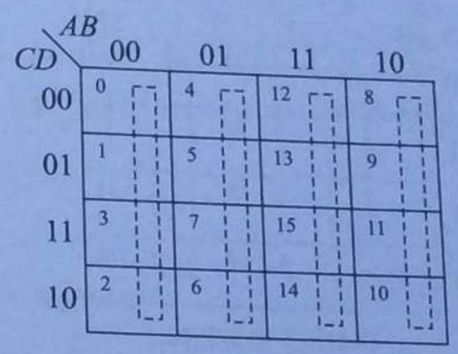
0-1-4-5, 2-3-6-7,
8-9-12-13, 10-11-14-15



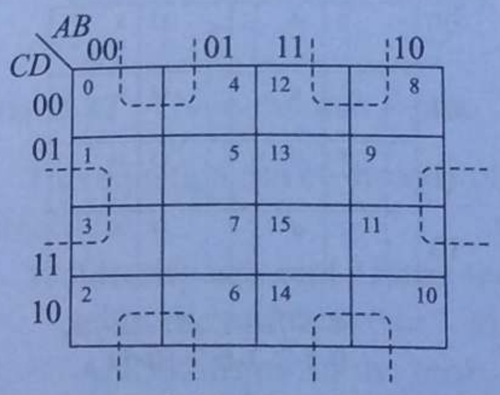
5-7-13-15, 0-2-8-10



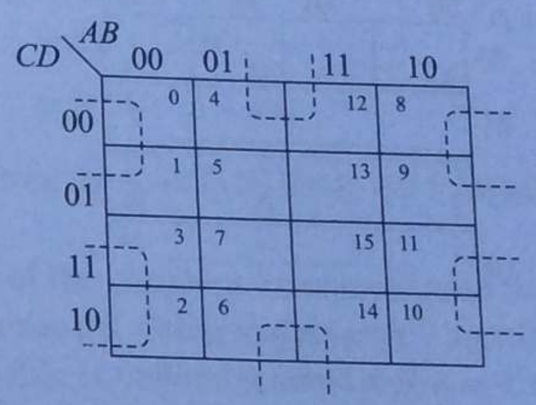
0-4-12-8, 1-5-13-9,
3-7-15-11, 2-6-14-10



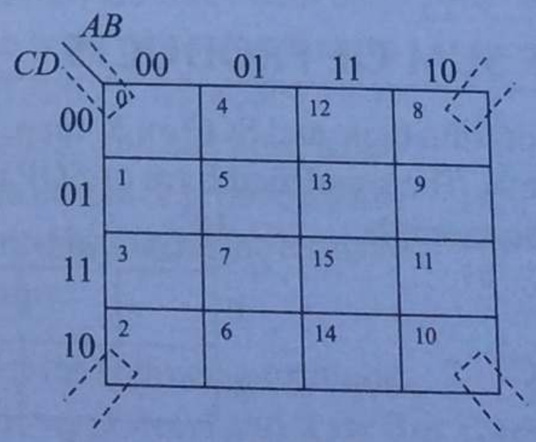
0-1-3-2, 4-5-7-6,
12-13-15-14, 8-5-11-10



1-3-9-11, 0-4-2-6, 12-8-14-10



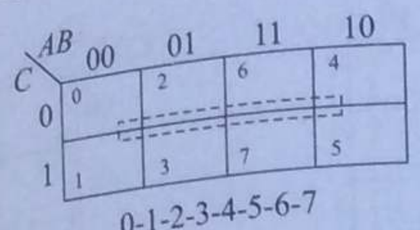
0-1-8-9, 3-2-11-10, 4-12-6-14



0-2-8-10

Fig. 3.27 Grouping of four adjacent cells in four-variable K-map

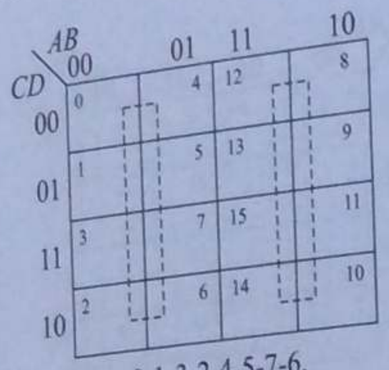
Grouping of eight adjacent cells Eight cells are said to be adjacent, if they differ in three variables. For example, in a three-variable K-map, all the eight cells form a single group as shown in Fig. 3.28.



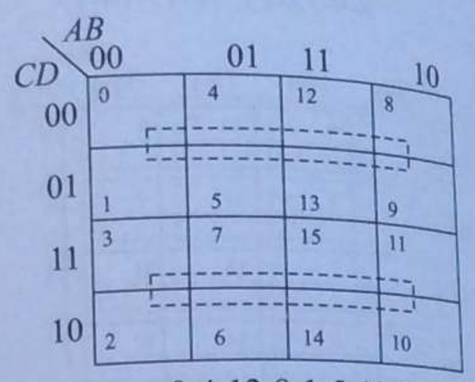
0-1-2-3-4-5-6-7

Fig. 3.28 Grouping of eight adjacent cells in three-variable K-map

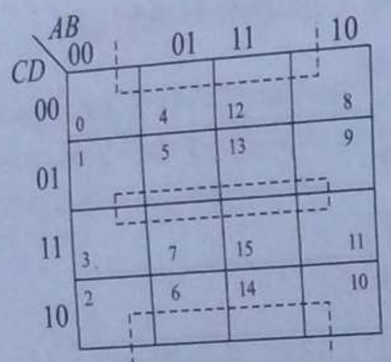
For a four-variable K-map, the possible groups of eight cells are shown in Fig. 3.29.



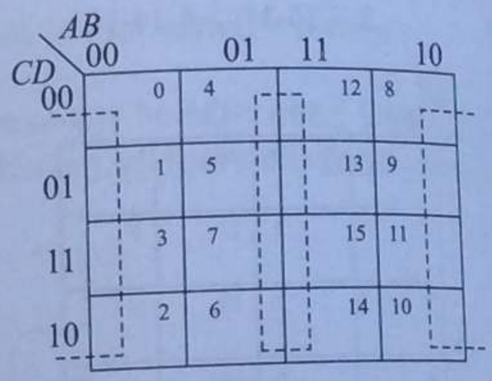
0-1-3-2-4-5-7-6,
12-13-15-14-8-9-10-11



0-4-12-8-1-5-13-9,
3-7-15-11-2-6-14-10



1-5-13-9-3-7-15-11,
0-4-12-8-2-6-14-10

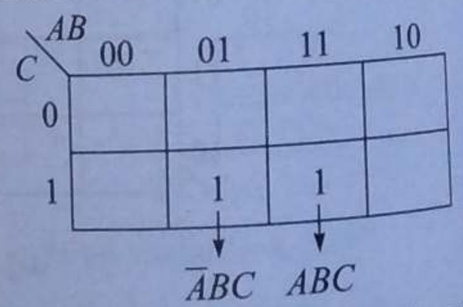


4-5-7-6-12-13-15-14,
0-1-2-3-8-9-10-11

Fig. 3.29 Grouping of eight adjacent cells in four-variable K-map

3.8 SIMPLIFICATION OF SUM OF PRODUCTS

We have seen the representation of truth table and SOP on K-map, and grouping of two, four, and eight adjacent cells. The simplified form of SOP is obtained by grouping adjacent 1s. Consider the example of two cells of 1s form one group:



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

Fig. 3.30 Four-variable K-map

This equation can be directly obtained from the K-map by using the following procedure:

1. Identify groups of adjacent 1s, and see the values of the variables associated. The variables 0 or 1 will get eliminated.

2. If the groups are more than one, then the term corresponding to each group of adjacent 1s is determined; and the simplified equation in sum of products form is obtained by ORing the terms of the groups.

Consider the following example.

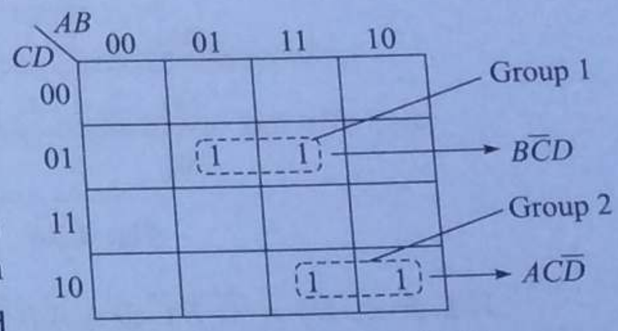


Fig. 3.31 Four-variable K-map

For group 1, variable A is 0 or 1 and hence it is eliminated, variable B is 1, C is 0 and D is 1. The term of group 1 is $B\overline{C}D$. For group 2, variable A is 1, variable B is 1 or 0 and hence it is eliminated, C is 1 and D is 0. The term of group 2 is $AC\overline{D}$. The simplified equation is the sum of these two groups, i.e.

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

Consider the example of four cells of 1s form a group:

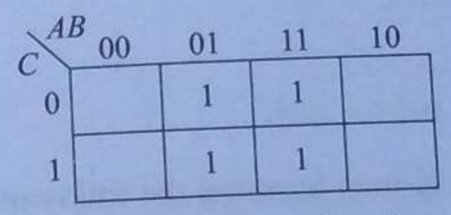


Fig. 3.32 Three-variable K-map

$$Y(A, B, C) = \overline{A}\,B\overline{C} + AB\overline{C} + \overline{A}\,BC + ABC$$
$$= AB(\overline{C} + C) + \overline{A}\,B(\overline{C} + C)$$
$$= AB + \overline{A}\,B$$
$$= B(A + \overline{A}) = B$$

This equation can be directly obtained from the K-map by using the following procedure:

1. Identify adjacent 1s, see the values of the variables associated with these cells, the variables 0 or 1 will get eliminated. Other variables will appear in ANDed form. In the term, the variable is uncomplemented if it is 1 and complemented if it is 0.
2. If the groups are more than one, then the term corresponding to each group of adjacent 1s is determined, and the simplified equation in sum of products form is obtained by ORing the terms of the groups.

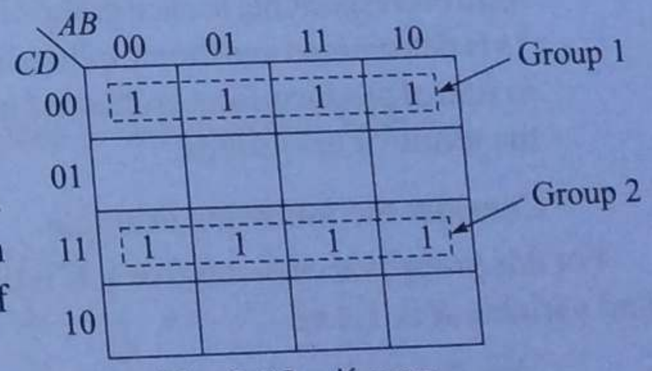


Fig. 3.33 K-map

Consider the following example.

For group 1, variables A and B are 0 or 1; and they are eliminated, C is 0 and D is 0; the term of group 1 is CD. For group 2, variable A and variable B is 0 or 1; and they are eliminated, C is 1 and D is 1; the term of group 2 is CD. The simplified equation is the sum of these two groups, i.e.

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

Consider the example of eight cells of 1s form a group:

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

Fig. 3.34 K-map

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

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

$$= \overline{A}\,\overline{C}D + A\overline{C}D + \overline{A}CD + ACD$$

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

$$= \overline{A}D + AD$$

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

This expression can be directly obtained from K-map by using the following procedure:

1. Identify adjacent 1s, see the value of the variables associated with these cells, the variables those are 0 or 1 will get eliminated. Other variables will appear in ANDed form. In this form, the variable will be in uncomplemented form if it is 1 and in the complemented form if it is 0.
2. If the groups are more than one, then the term corresponding to each group of adjacent 1s is determined and the simplified equation in sum of products form is obtained by ORing the terms of the groups.

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

Fig. 3.35 K-map

Consider the following example.

For this group, variable A is 0 or 1, C is 0 or 1, D is 0 or 1 and they are eliminated; and variable B is 1, i.e.

$$Y = B$$

In the above discussion, groups of 2, 4, and 8 adjacent 1s are considered, and it is found that:

1. Groups of two adjacent 1s result in a term with one variable less than the total number of variables.
2. Groups of four adjacent 1s result in a term with two variables less than the total number of variables.
3. Groups of eight adjacent 1s result in a term with three variables less than the total number of variables.

The same logic can be extended to 16, 32, and 64 adjacent 1s. The procedure to simplify the Boolean expression using K-map is:

1. Represent the sum of products expression on a K-map.
2. Form the groups of possible adjacent 1s as 8, 4, 2, 1.
3. Write the Boolean term of each group.
4. Write the simplified expression in sum of products form.

Example 3.18 Minimize the following expressions using K-map:

(a) $Y(A, B, C) = \Sigma m(1, 3, 5, 7)$

(b) $Y(A, B, C) = \Sigma m(0, 1, 4, 5)$

(c) $Y(A, B, C) = \Sigma m(0, 2, 4, 6)$

Solution

(a) $Y(A, B, C) = \Sigma m(1, 3, 5, 7)$

The representation of SOP by K-map is shown in Fig. 3.36(a).

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

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

Fig. 3.36(a) $Y(A, B, C) = \Sigma m(1, 3, 5, 7)$

(b) $Y(A, B, C) = \Sigma m(0, 1, 4, 5)$

The representation of SOP by K-map is shown in Fig. 3.36(b).

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

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

Fig. 3.36(b) $Y(A, B, C) = \Sigma m(0, 1, 4, 5)$

(c) $Y(A, B, C) = \Sigma m(0, 2, 4, 6)$

The representation of SOP by K-map is shown in Fig. 3.36(c).

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

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

Fig. 3.36(c) $Y(A, B, C) = \Sigma m(0, 2, 4, 6)$

Example 3.19 Minimize the following expressions using K-map:

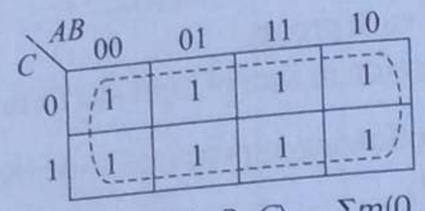
(a) $Y(A, B, C) = \Sigma m(0, 1, 2, 3, 4, 5, 6, 7)$

(b) $Y(A, B, C) = \Sigma m(0, 2, 4)$

Solution

(a) $Y(A, B, C) = \Sigma m(0, 1, 2, 3, 4, 5, 6, 7)$

The K-map representation is shown in Fig. 3.37(a).

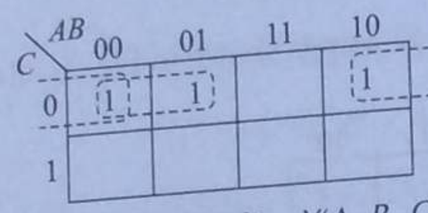


$Y(A, B, C) = 1$

Fig. 3.37(a) $Y(A, B, C) = \Sigma m(0, 1, 2, 3, 4, 5, 6, 7)$

(b) $Y(A, B, C) = \Sigma m(0, 2, 4)$

The K-map representation is shown in Fig. 3.37(b).



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

Fig. 3.37(b) $Y(A, B, C) = \Sigma m(0, 2, 4)$

Example 3.20 Minimize the following expressions using K-map:

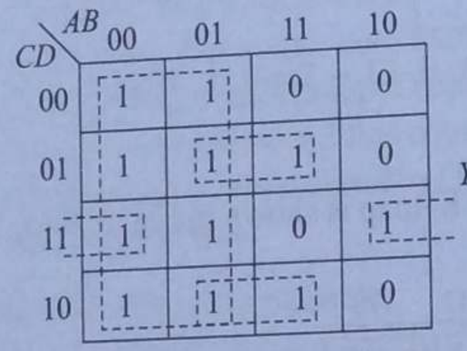
(a) $Y(A, B, C, D) = \Sigma m(0, 1, 2, 3, 4, 5, 6, 7, 11, 13, 14)$

(b) $Y(A, B, C, D) = \Sigma m(0, 2, 4\,7, 8, 10, 12, 13)$

Solution

(a) $Y(A, B, C, D) = \Sigma m(0, 1, 2, 3, 4, 5, 6, 7, 11, 13, 14)$

The K-map representation is shown in Fig. 3.38(a).



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

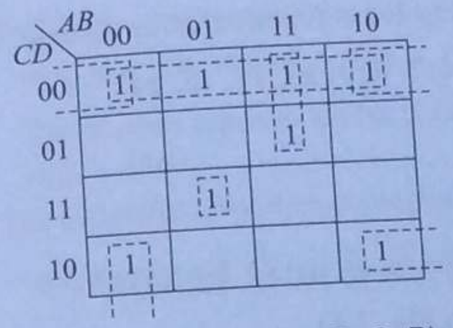
Fig. 3.38(a) $Y(A, B, C, D) = \Sigma m(0, 1, 2, 3, 4, 5, 6, 7, 11, 13, 14)$

The simplified Boolean expression is

$$Y(A, B, C, D) = \overline{A} + \overline{B}CD + B\overline{C}D + BC\overline{D}$$

(b) $Y(A, B, C, D) = \Sigma m(0, 2, 4, 7, 8, 10, 12, 13)$

The K-map representation is shown in Fig. 3.38(b).



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

Fig. 3.38(b) $Y(A, B, C, D) = \Sigma m(0, 2, 4\,7, 8, 10, 12, 13)$

The simplified Boolean expression is

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

Example 3.21 Minimize the following logic function using K-map:

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

and implement it using logic gates.

Solution

The logical function is

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

The K-map representation is shown in Fig. 3.39.

The simplified Boolean expression is

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

This expression can be implemented using AND, OR, and NOT gates.

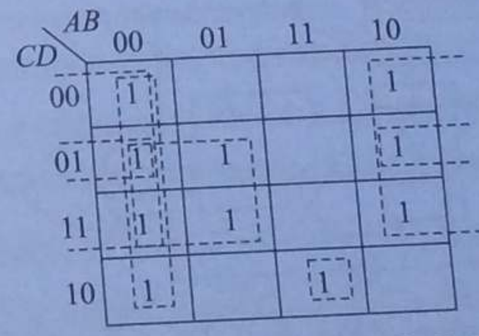


Fig 3.39 $Y(A, B, C, D) = \Sigma m(0, 1, 2, 3, 5, 7, 8, 9, 11, 14)$

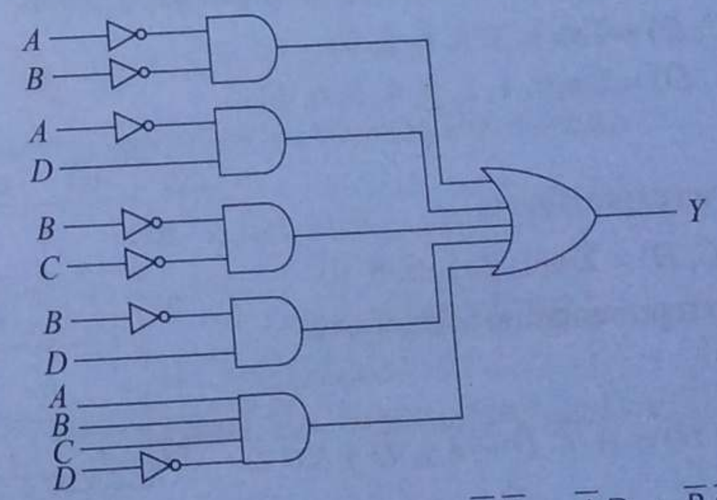


Fig 3.40 Implementation of $Y(A, B, C, D) = \overline{A}\,\overline{B} + \overline{A}D + \overline{B}\,\overline{C} + \overline{B}D + ABC\overline{D}$

Example 3.22 Minimize the following logic function using K-map:

$$Y(A, B, C, D) = \Sigma m(0, 1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 14)$$

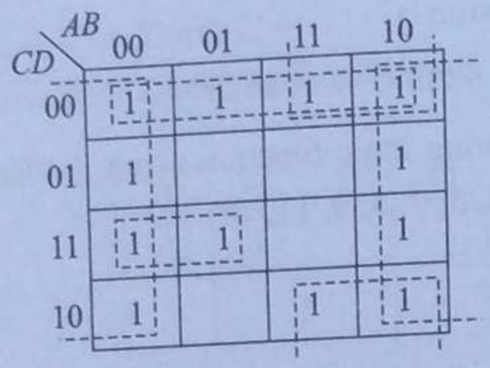
and implement it using logic gates.

Solution

The logical function is

$$Y(A, B, C, D) = \Sigma m(0, 1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 14)$$

The K-map representation is shown in Fig. 3.41.



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

Fig 3.41 $Y(A, B, C, D) = \Sigma m(0, 1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 14)$

Simplified Boolean expression is

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

This expression can be implemented using AND, OR, and NOT gates.

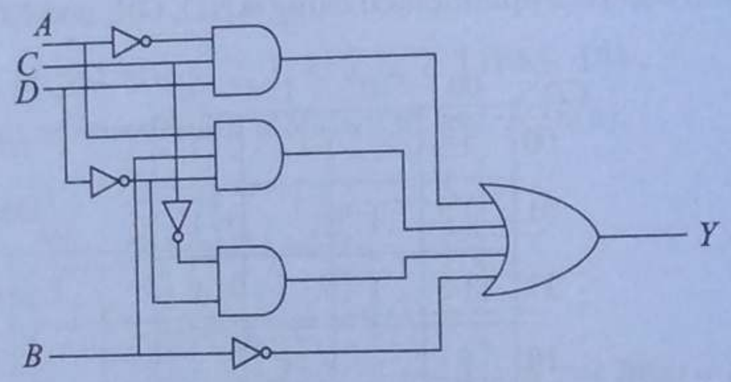


Fig 3.42 Implementation of $Y(A, B, C, D) = \overline{B} + \overline{C}\,\overline{D} + \overline{A}CD + AB\overline{D}$

Example 3.23 Simplify the following Boolean expressions:

(a) $Y(A, B, C, D) = \Sigma m(1, 2, 5, 6, 8, 9)$

(b) $Y(A, B, C, D) = \Sigma m(0, 1, 2, 3, 4, 5, 6, 11)$

Solution

(a) The Boolean expression is

$$Y(A, B, C, D) = \Sigma m(1, 2, 5, 6, 8, 9)$$

The K-map representation is shown in Fig. 3.43(a).

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

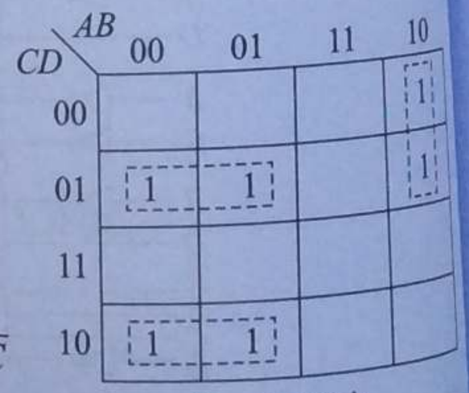


Fig 3.43(a)

(b) The Boolean expression is

$$Y(A, B, C, D) = \Sigma m(0, 1, 2, 3, 4, 5, 6, 11)$$

The K-map representation is shown in Fig. 3.43(b).

The simplified Boolean expression is

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

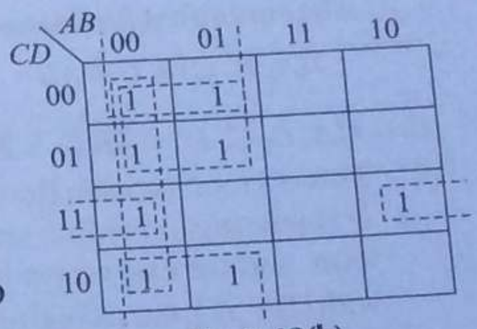


Fig 3.43(b)

Example 3.24 Minimize the following Boolean expressions using K-map:

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

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

Solution

Step 1: If the given SOP is not in a standard form, it should be converted into standard SOP.

Step 2: Write the expression in terms of minterms.

Step 3: Represent the expression by K-map.

Step 4: Form the groups of possible adjacent 1s as 8, 4, 2, 1.

Step 5: Write the Boolean term for each group.

Step 6: Write the simplified expression in sum of products form.

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

This is a three-variable Boolean expression. Variable C is missing in the first term, A is missing in the second term, A and B are missing in the third term, and the expression is not in a standard form. ANDing the first term with $(C + \overline{C})$, second term with $(A + \overline{A})$, and third term with $(A + \overline{A})$ $(B + \overline{B})$, the standard sum of products form is obtained

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

$$= ABC + AB\,\overline{C} + A\overline{B}\,C + \overline{A}\,\overline{B}\,C + ABC + A\overline{B}\,C + \overline{A}\,BC + \overline{A}\,\overline{B}\,C$$

$$= ABC + AB\,\overline{C} + A\overline{B}\,C + \overline{A}\,\overline{B}\,C + \overline{A}\,BC$$

In minterms, the Boolean expression can be written as

$$Y(A, B, C) = \Sigma m(7, 6, 5, 1, 3)$$

The K-map representation is shown in Fig. 3.44(a).

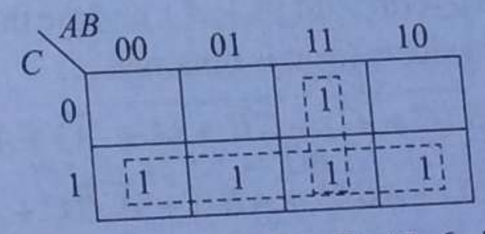


Fig. 3.44(a) $Y(A, B, C) = \Sigma m(7, 6, 5, 1, 3)$

The simplified Boolean expression is

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

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

This is a four-variable Boolean expression in sum of products form. Variable D is missing in the first term, A is missing in the second term and the third term, and the expression is not in a standard form. ANDing the first term with $(D + \overline{D})$, the second term with $(A + \overline{A})$, and the third term with $(A + \overline{A})$, the standard sum of products form is obtained.

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

$$= AB\overline{C}D + AB\overline{C}\,\overline{D} + ABCD + \overline{A}BCD + ABC\overline{D} + \overline{A}BC\overline{D}$$

In minterms, the Boolean expression can be written as

$$Y(A, B, C, D) = \Sigma m(13, 12, 15, 7, 14, 6)$$

The K-map representation is shown in Fig. 3.44(b).

The simplified Boolean expression is

$$Y(A, B, C, D) = AB + BC$$

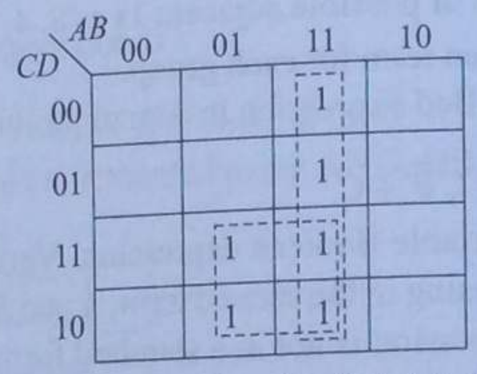


Fig. 3.44(b) $Y(A, B, C, D) = \Sigma m(13, 12, 15, 7, 14, 6)$

Example 3.25 Minimize the following Boolean expressions using K-map:

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

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

Solution

(a) $Y(A, B, C) = AB + C\overline{B} + AC$

This is a three-variable Boolean expression. Variable C is missing in the first term, A is missing in the second term, B is missing in the third term and the expression is not in a standard form. ANDing the first term with $(C + \overline{C})$, the second term with $(A + \overline{A})$, and the third term with $(B + \overline{B})$, the standard sum of products form is obtained.

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

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

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

In minterms, the Boolean expression can be written as

$$Y(A, B, C) = \Sigma m(7, 6, 5, 1)$$

The K-map representation is shown in Fig. 3.45(a).

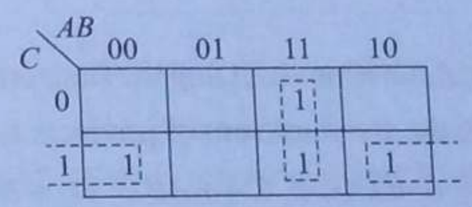


Fig. 3.45(a) $Y(A, B, C) = \Sigma m(7, 6, 5, 1)$

The simplified Boolean expression is

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

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

This is a four-variable Boolean expression in sum of products form. ANDing the first term with $(D + \overline{D})$, the second term with $(A + \overline{A})$, and the third term with $(A + \overline{A})$ and $(C + \overline{C})$, the standard sum of products form is obtained.

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

$$= AB\overline{C}D + AB\overline{C}\,\overline{D} + ABCD + \overline{A}BCD + (AB\overline{D} + \overline{A}B\overline{D})(C + \overline{C})$$

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

$$= AB\overline{C}D + AB\overline{C}\,\overline{D} + ABCD + \overline{A}BCD + ABC\overline{D} + \overline{A}BC\overline{D} + \overline{A}BC\overline{D}$$

In minterms, the Boolean expression can be written as

$$Y(A, B, C, D) = \Sigma m(13, 12, 15, 7, 14, 6, 4)$$

The K-map representation is shown in Fig. 3.45(b).

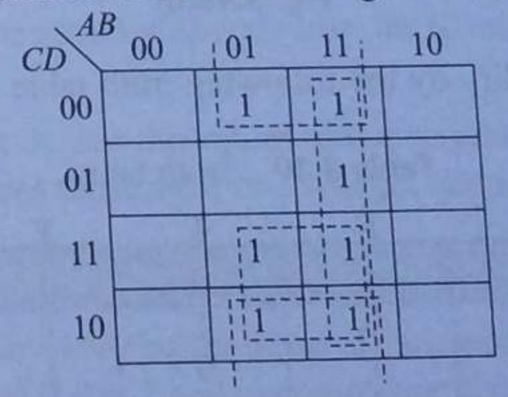


Fig. 3.45(b) $Y(A, B, C, D) = \Sigma m(13, 12, 15, 7, 14, 6, 4)$

The simplified Boolean expression is $Y(A, B, C, D) = CB + AB + B\overline{D}$

Example 3.26 Reduce the following function using K-map technique and implement using basic gates.

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

Solution

It is a four-variable logical function. ANDing the first term with $(C + \overline{C})$, and the third term with $(A + \overline{A})$, the standard sum of products form is obtained.

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

In minterms, the Boolean expression can be written as

$$Y(A, B, C, D) = \Sigma m(3, 1, 12, 15, 7)$$

The K-map representation is shown in Fig. 3.46(a).

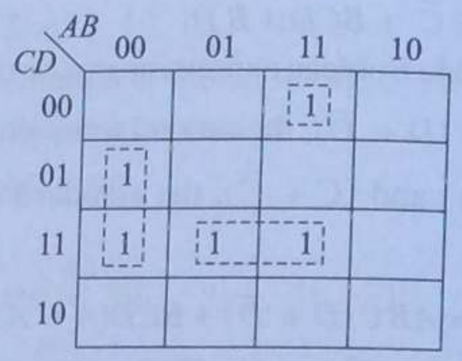


Fig. 3.46(a)

The simplified Boolean expression is $Y(A, B, C, D) = \overline{A}\,\overline{B}D + BCD + AB\overline{C}\,\overline{D}$

Implementation of the above expression is shown in Fig. 3.46(b).

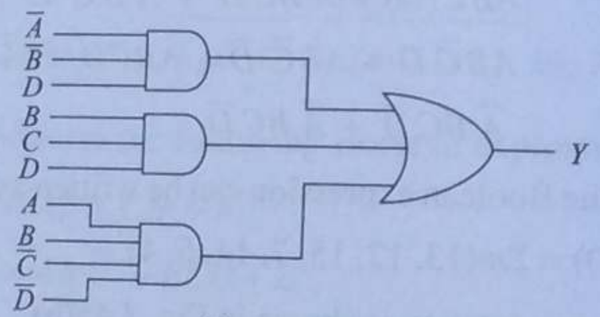


Fig. 3.46(b)

Example 3.27 Simplify by the following truth table using K-map method.

Table 3.10 Truth table

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

Solution

K-map representation of the above truth table is shown in Fig. 3.47.

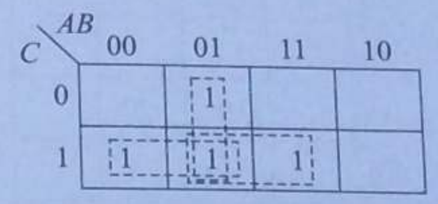


Fig. 3.47 K-map representation of Table 3.10

Simplified Boolean expression is $Y(A, B, C) = \overline{A}\,B + \overline{A}\,C + BC$

3.9 SIMPLIFICATION OF PRODUCT OF SUMS

The simplified product of sums expression is obtained by grouping adjacent 0s. Consider the example of a group of two cells of 0s:

$$Y(A, B, C) = (A + \overline{B} + \overline{C})(A + \overline{B} + C)$$
$$= AA + A\overline{B} + AC + A\overline{B} + \overline{B}\,\overline{B} + \overline{B}C + A\overline{C} + \overline{B}\,\overline{C} + C\overline{C}$$
$$= A + A\overline{B} + AC + \overline{B} + \overline{B}C + A\overline{C} + \overline{B}\,\overline{C}$$
$$= A(1 + \overline{B}) + A(C + \overline{C}) + \overline{B} + \overline{B}(C + \overline{C})$$
$$= A + A + \overline{B} + \overline{B}$$
$$= A + \overline{B}$$

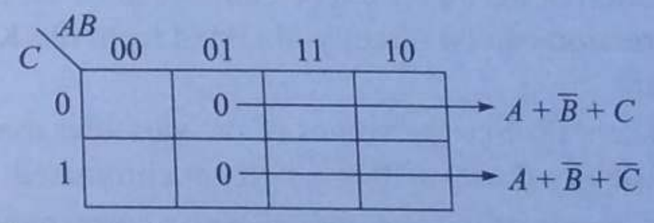


Fig. 3.48 Three-variable K-map with two cells of 0s form one group

This expression can be obtained directly from the K-map by using the following procedure.

1. Identify adjacent 0s; see the values of the variables associated with these cells. The variables those are 0 or 1 will get eliminated.
2. If the groups are more than one, then the term corresponding to each group of adjacent 0s is determined, and the simplified equation in product of sums form is obtained by ANDing the terms of the groups. In the term, the variable is complemented if it is 1 and uncomplemented if it is 0.

Consider the following example.

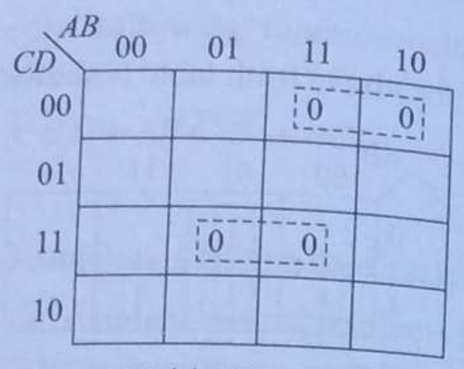


Fig. 3.49 Four-variable K-map with group of 0s

For group 1, variable A is 1, variable B is 0 or 1 and it is eliminated, C is 0, and D is 0. The term of group 1 is $\overline{A}+C+D$. For group 2, variable A is 1 or 0 and it is eliminated, variable B is 1, C is 1, and D is 1. The term of group 2 is $(\overline{B}+\overline{C}+\overline{D})$. The simplified equation is the product of these two groups, i.e.

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

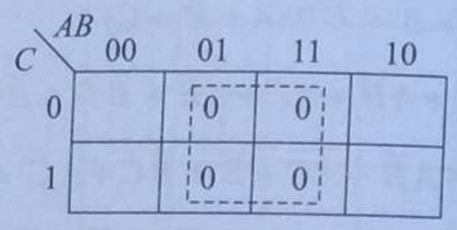


Fig. 3.50 Three-variable K-map with a group of four cells of 0s

Consider the example of a group of four cells of 0s:

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

After simplification of the above expression, we get $Y(A, B, C) = \overline{B}$.

The above expression can be directly obtained from the K-map by using the following procedure.

1. Identify adjacent 0s. See the values of the variables associated with these cells. The variables those are 0 or 1 will get eliminated.
2. If the groups are more than one, then the term corresponding to each group of adjacent 0s is determined, and the simplified equation in product of sums form is obtained by ANDing the terms of the groups. In the term, the variable is complemented if it is 1 and uncomplemented if it is 0.

Consider the following example.

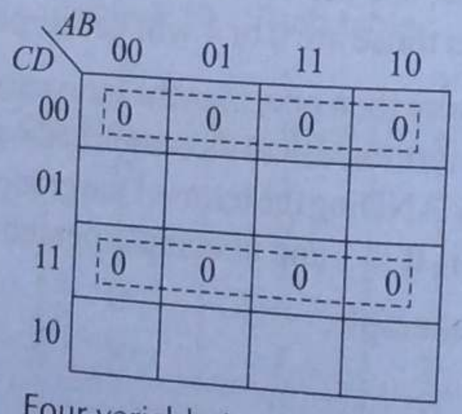


Fig. 3.51 Four-variable K-map with group of 0s

For group 1, variable A is 0 or 1 and it is eliminated, variable B is 0 or 1 and it is eliminated, C is 0, and D is 0. The term of group 1 is $C+D$. For group 2, variable A is 1 or 0 and it is eliminated, variable B is 0 or 1 and it is eliminated, C is 1, and D is 1. The term of group 2 is $\overline{C}+\overline{D}$. The simplified equation is the product of these two groups, i.e.

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

Consider the example of a group of eight cells of 0s:

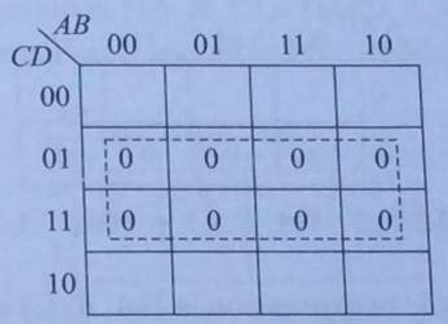


Fig. 3.52 Four-variable K-map with a group of eight cells of 0s

For the group, variable A is 0 or 1 and it is eliminated, variable B is 0 or 1 and it is eliminated, C is 0 or 1 and it is eliminated, and D is 0. The term of the group is D. The simplified equation is

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

Consider the following example.

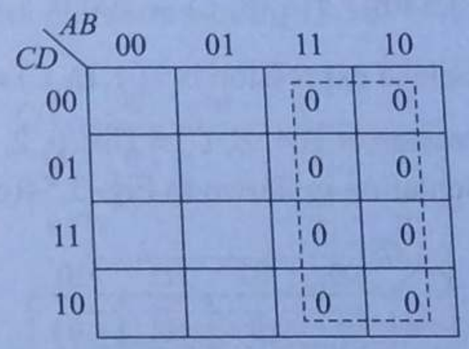


Fig. 3.53 Four-variables K-map with group of 0s

For the group, variable A is 1, variables B, C, and D are either 0 or 1 and hence eliminated. The term of the group is A. The simplified equation is

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

In the above discussion, the groups of 2, 4, and 8 adjacent 0s are considered and it is found that

1. A group of two adjacent 0s results in a term with one variable less than the total number of variables.
2. A group of four adjacent 0s results in a term with two variables less than the total number of variables.
3. A group of eight adjacent 0s results in a term with three variables less than the total number of variables.

The same logic can be extended for groups of 6, 32, and 64 adjacent 0s.

Example 3.28 Minimize the following expressions using K-map.

(a) $Y(A, B, C) = \Pi M(1, 3, 5, 7)$
(b) $Y(A, B, C) = \Pi M(0, 1, 4, 5)$
(c) $Y(A, B, C) = \Pi M(0, 2, 4, 6)$

Solution

(a) The expression is $Y(A, B, C) = \Pi M(1, 3, 5, 7)$
The K-map representation is shown in Fig. 3.54 (a).

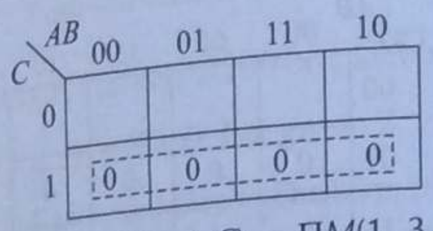


Fig. 3.54(a) $Y(A, B, C) = \Pi M(1, 3, 5, 7)$

The simplified Boolean expression is $Y(A, B, C) = \overline{C}$

(b) The Boolean expression is $Y(A, B, C) = \Pi M\ (0, 1, 4, 5)$
The K-map representation is shown in Fig. 3.54 (b).

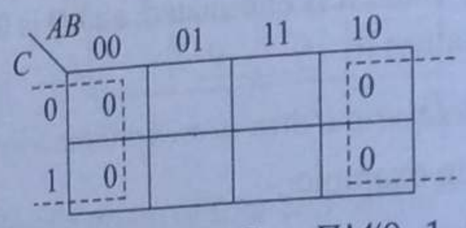


Fig. 3.54(b) $Y(A, B, C) = \Pi M(0, 1, 4, 5)$

The simplified Boolean expression is $Y(A, B, C) = B$

(c) The Boolean expression is $Y(A, B, C) = \Pi M(0, 2, 4, 6)$
The K-map representation is shown in Fig. 3.54(c).

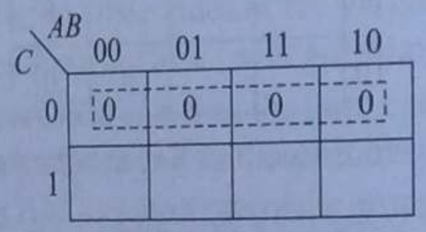


Fig. 3.54(c) $Y(A, B, C) = \Pi M(0, 2, 4, 6)$

The simplified Boolean expression is $Y(A, B, C) = C$

Example 3.29 Minimize the following expressions using K-map.

(a) $Y(A, B, C) = \Pi M(0, 1, 2, 3, 4, 5, 6, 7)$
(b) $Y(A, B, C) = \Pi M(0, 2, 4, 5)$

Solution

(a) The Boolean expression is $Y(A, B, C) = \Pi M(0, 1, 2, 3, 4, 5, 6, 7)$
The K-map representation is shown in Fig. 3.55(a).

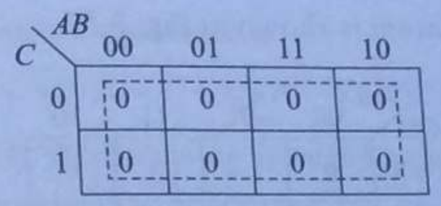


Fig. 3.55(a) $Y(A, B, C) = \Pi M(0, 1, 2, 3, 4, 5, 6, 7)$

The simplified Boolean expression is $Y(A, B, C) = 0$

(b) The Boolean expression is $Y(A, B, C) = \Pi M(0, 2, 4, 5)$
The K-map representation is shown in Fig. 3.55(b).

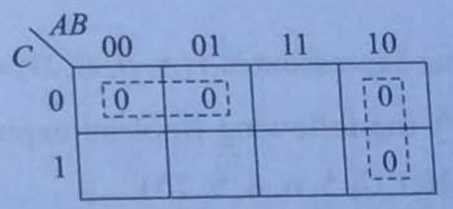


Fig. 3.55(b) $Y(A, B, C) = \Pi M(0, 2, 4, 5)$

The simplified Boolean expression is $Y(A, B, C) = (A + C) \cdot (\overline{A} + B)$

Example 3.30 Reduce the following function using K-map technique.

$$Y(A, B, C, D) = \Pi M(0, 2, 8, 9, 12, 13, 15)$$

Solution

The Boolean expression is $Y\ (A, B, C, D) = \Pi M(0, 2, 8, 9, 12, 13, 15)$
The K-map representation is shown in Fig. 3.56.

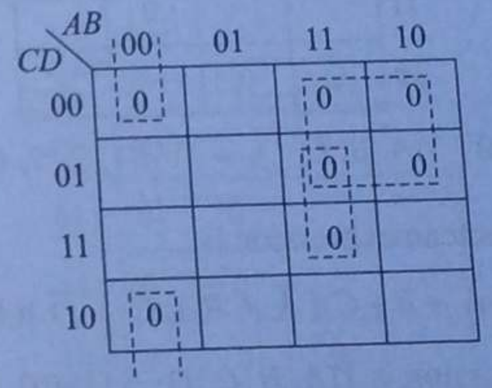


Fig. 3.56 $Y(A, B, C, D) = \Pi M(0, 2, 8, 9, 12, 13, 15)$

The simplified Boolean expression is

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

Example 3.31 Reduce the following function using K-map technique.

$$Y(A, B, C, D) = \Pi M(0, 1, 2, 3, 8, 9, 10, 11, 12, 13)$$

Solution

The Boolean expression is

$$Y(A, B, C, D) = \Pi M(0, 1, 2, 3, 8, 9, 10, 11, 12, 13)$$

The K-map representation is shown in Fig. 3.57.

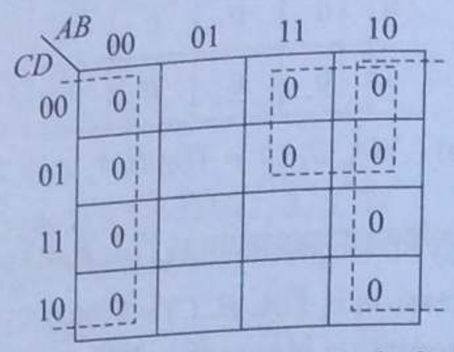


Fig. 3.57 $Y(A, B, C, D = \Pi M(0, 1, 2, 3, 8, 9, 10, 11, 12, 13)$

The simplified Boolean expression is $Y(A, B, C, D) = (\overline{A} + C)B$

Example 3.32 Simplify the following Boolean expressions:

(a) $Y(A, B, C, D) = \Pi M(1, 2, 5, 6, 8, 9, 15)$

(b) $Y(A, B, C, D) = \Pi M(0, 1, 2, 3, 5, 6, 7, 12)$

Solution

(a) The Boolean expression is $Y(A, B, C, D) = \Pi M(1, 2, 5, 6, 8, 9, 15)$
The K-map representation is shown in Fig. 3.58(a).

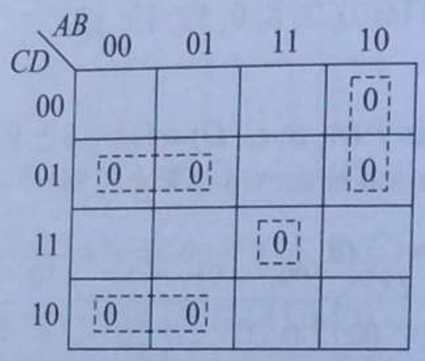


Fig. 3.58(a) $Y(A, B, C, D) = \Pi M(1, 2, 5, 6, 8, 9, 15)$

The simplified Boolean expression is

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

(b) The Boolean expression is $Y(A, B, C, D) = \Pi M(0, 1, 2, 3, 5, 6, 7, 12)$
The K-map representation is shown in Fig. 3.58(b).

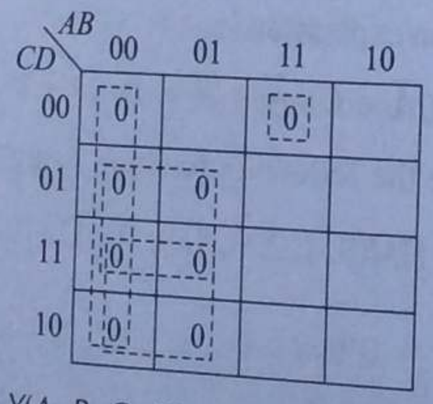


Fig. 3.58(b) $Y(A, B, C, D) = \Pi M(0, 1, 2, 3, 5, 6, 7, 12)$

The simplified Boolean expression is

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

Example 3.33 Simplify the following Boolean expressions:

(a) $Y(A, B, C, D) = \Pi M(1, 2, 5, 6, 8, 9, 10, 11, 15)$

(b) $Y(A, B, C, D) = \Pi M(0, 1, 2, 3, 5, 6, 7, 12, 14)$

Solution

(a) The Boolean expression is $Y(A, B, C, D) = \Pi M(1, 2, 5, 6, 8, 9, 15)$
The K-map representation is shown in Fig. 3.59(a).

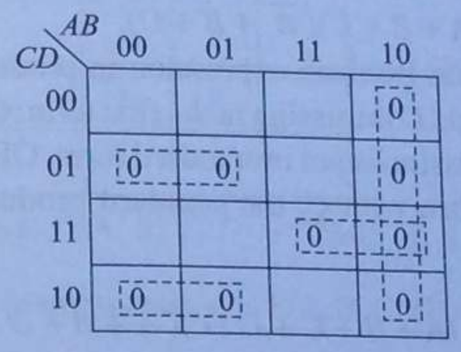


Fig. 3.59(a) $Y(A, B, C, D) = \Pi M(1, 2, 5, 6, 8, 9, 10, 11,15)$

The simplified Boolean expression is

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

(b) The Boolean expression is $Y(A, B, C, D) = \Pi M(0, 1, 2, 3, 5, 6, 7, 12, 14)$
The K-map representation is shown in Fig. 3.59(b).

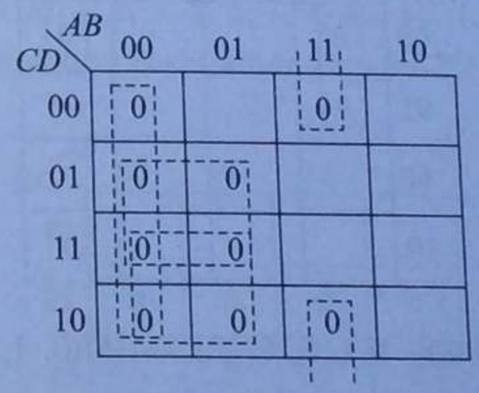


Fig. 3.59(b) $Y(A, B, C, D) = \Pi M(0, 1, 2, 3, 5, 6, 7, 12, 14)$

The simplified Boolean expression is

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

Example 3.34 Minimize the following Boolean expressions using K-map:

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

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

Solution

Step 1: If the given product of sums is not in standard form, it should be converted into standard form.

Step 2: Write the expression in terms of maxterms.

Step 3: Represent the expression by K-map.

Step 4: Form the groups of possible adjacent 0s as 8, 4, 2, and 1.

Step 5: Write the Boolean term for each group.

Step 6: Write the simplified expression in product of sums form.

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

It is a four-variable Boolean expression in product of sums form. In the expression, variable D is missing in the first term, C is missing in the second term and the expression is not in standard form. ORing the first term with D and the second term with C, the standard product of sums form can be obtained.

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

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

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

This can be written as $Y(A, B, C, D) = \Pi M(0, 1, 8, 10)$

The K-map representation is shown in Fig. 3.60(a).

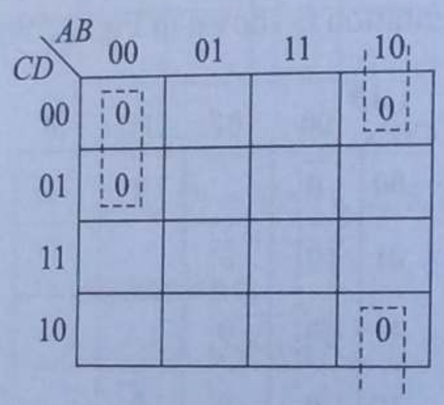


Fig. 3.60(a) $Y(A, B, C, D) = \Pi M(0, 1, 8, 10)$

The simplified Boolean expression is $Y(A, B, C, D) = (A + B + C)(\overline{A} + B + D)$

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

It is a four-variable Boolean expression in product of sums form. In the above expression, variable C is missing in the first term as well as the second term and the expression is not in standard form. ORing the first and the second term with $C\overline{C}$, the standard product of sums form can be obtained.

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

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

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

This can be written as $Y(A, B, C, D) = \Pi M(5, 7, 13, 15)$

The K-map representation is shown in Fig. 3.61.

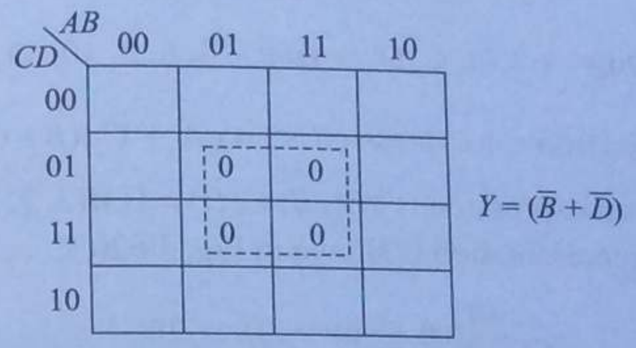


Fig. 3.61 $Y(A, B, C, D) = \Pi M(5, 7, 13, 15)$

The simplified Boolean expression is $Y(A, B, C, D) = (\overline{B} + \overline{D})(\overline{A} + \overline{B} + \overline{C})$

Example 3.35 Minimize the following expressions using K-maps.

(a) $Y(A, B, C, D) = \Pi M(1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15)$

(b) $Y(A, B, C, D) = \Pi M(1, 4, 6, 9, 10, 11, 14, 15)$

(c) $Y(A, B, C, D) = \Pi M(2, 7, 8, 9, 10, 12)$

Solution

(a) Given Boolean expression is

$$Y(A, B, C, D) = \Pi M(1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15)$$

The K-map representation is shown in Fig. 3.62(a).

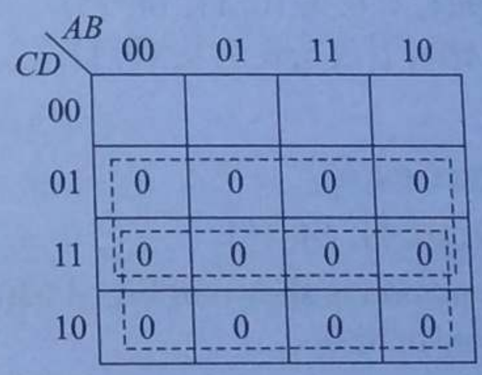


Fig. 3.62(a) $Y(A, B, C, D) = \Pi M(1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15)$

The simplified Boolean expression is $Y(A, B, C, D) = \overline{D}\ \overline{C}$

(b) Given Boolean expression is

$$Y(A, B, C, D) = \Pi M(1, 4, 6, 9, 10, 11, 14, 15)$$

The K-map representation is shown in Fig. 3.62(b).

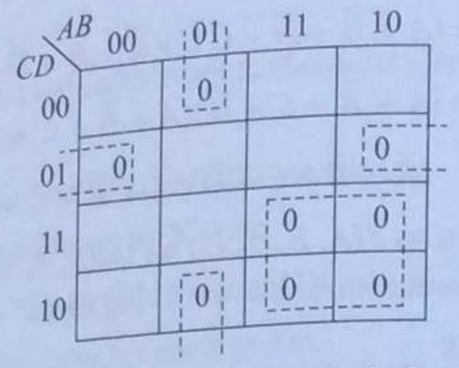


Fig. 3.62(b) $Y(A, B, C, D) = \Pi M(1, 4, 6, 9, 10, 11, 14, 15)$

The simplified Boolean expression is $Y = (\overline{A} + \overline{C})(B + C + \overline{D})(A + \overline{B} + D)$

(c) Given Boolean expression is $Y(A, B, C, D) = \Pi M(2, 7, 8, 9, 10, 12)$
The K-map representation is shown in Fig. 3.62(c).

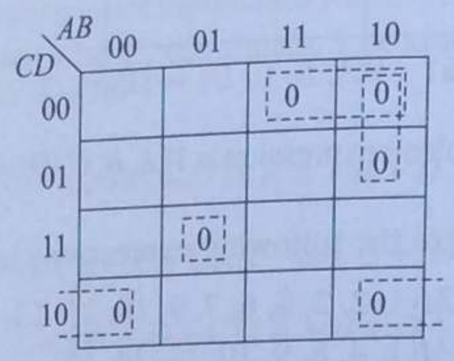


Fig. 3.62 (c) $Y(A, B, C, D) = \Pi M(2, 7, 8, 9, 10, 12)$

Simplified Boolean expression is

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

Example 3.36 Write down the simplified Boolean expression in (a) sum of products form and (b) products of sums form for:

(i) $Y(A, B, C, D) = \Sigma m(1, 4, 6, 9, 10, 11, 14, 15)$
(ii) $Y(A, B, C, D) = \Pi M(0, 1, 3, 5, 6, 7, 9, 10, 11, 12, 13, 15)$

Solution

(i) Given Boolean expression is

$$Y = \Sigma m(1, 4, 6, 9, 10, 11, 14, 15)$$

The K-map representation is shown in Fig. 3.63(a). The cells that do not contain 1 are assumed to be 0.

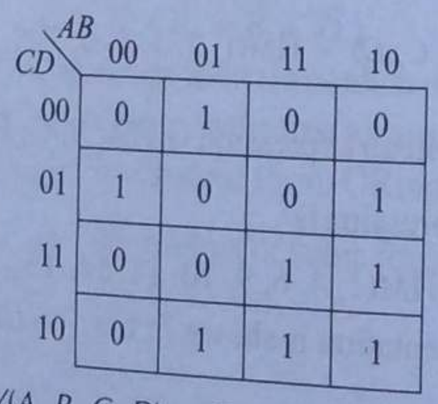


Fig. 3.63(a) $Y(A, B, C, D) = \Sigma m(1, 4, 6, 9, 10, 11, 14, 15)$

(a) To write the simplified expression in sum of products form, we have to form groups of adjacent 1s as shown in Fig. 3.63(b).

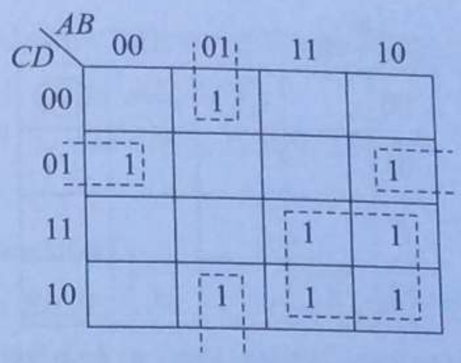


Fig. 3.63(b) $Y(A, B, C, D) = \Sigma m(1, 4, 6, 9, 10, 11, 14, 15)$

The simplified Boolean expression is

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

(b) To write the simplified expression in product of sums form, we have to form groups of 0s as shown in Fig. 3.63(c).
The simplified Boolean expression is

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

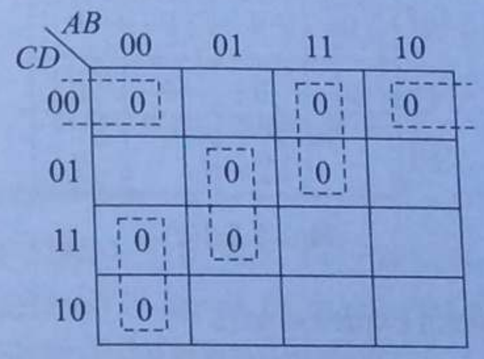
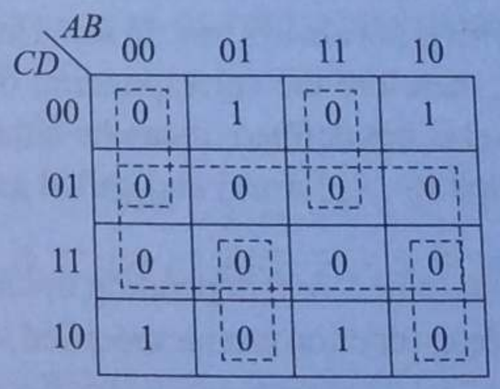


Fig. 3.63(c) $Y(A, B, C, D) = \Sigma m(1, 4, 6, 9, 10, 11, 14, 15)$

(ii) Given Boolean expression is

$$Y(A, B, C, D) = \Pi M(0, 1, 3, 5, 6, 7, 9, 10, 11, 12, 13, 15)$$

The K-map representation for the given function is shown in Fig. 3.64 (a). The cells that do not contain 0 are assumed to be 1.



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

Fig. 3.64(a) $Y(A, B, C, D) = \Pi M(0, 1, 3, 5, 6, 7, 9, 10, 11, 12, 13, 15)$

(a) To write the simplified expression in sum of products form, we have to form groups of adjacent 1s as shown in Fig. 3.64(b).

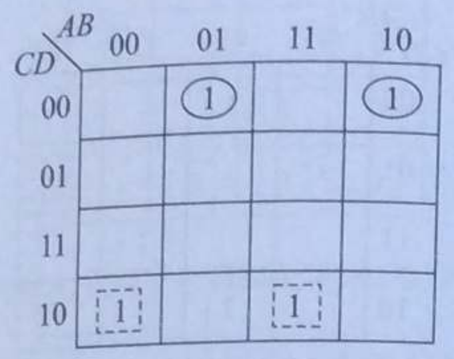


Fig. 3.64(b) $Y(A, B, C, D) = \Sigma M(0, 1, 3, 5, 6, 7, 9, 10, 11, 12, 13, 15)$

The simplified Boolean expression is

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

(b) To write the simplified expression in product of sums form, we have to form groups of 0s as shown in Fig. 3.64(c).

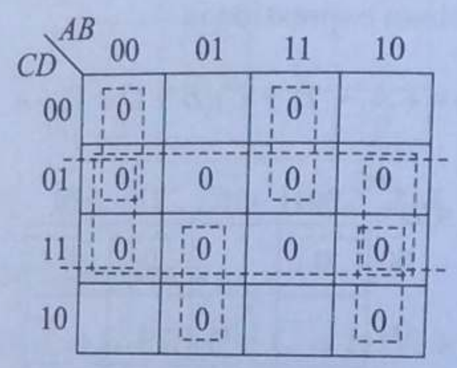


Fig. 3.64(c)

The simplified Boolean expression is

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

3.10 DON'T-CARE CONDITION

When the Boolean function is defined in terms of minterms, 1s are entered in the K-map corresponding to the input conditions; and when the Boolean function is defined in terms of maxterms, 0s are entered. The cells that do not contain 1 are assumed to contain 0 and vice-versa. This is not always true. In some logic circuits, certain input conditions never occur, therefore the corresponding output never appears. In such cases, the output level is not defined, it can be either HIGH or LOW. These output levels are indicated by '×' [cross] and called as don't-care condition.

The × mark in a cell may be assumed to be 1 or 0 depending upon which one leads to a simpler expression. The Boolean function can be specified in one of the following ways:

1. In terms of minterms and don't-care conditions:

$$Y(A, B, C, D) = \Sigma m(0, 1, 2, 4, 8, 10) + d(5, 7)$$

2. In terms of maxterms and don't-care conditions:

$$Y(A, B, C, D) = \Pi M(0, 4, 5, 14, 15) \cdot d(1, 2, 7)$$

Example 3.37 Simplify the following Boolean expressions

(a) $Y = \Sigma m(1, 3, 7, 11, 15) + d(0, 2, 5)$

(b) $Y = \Pi M(4, 5, 6, 7, 8, 12) \cdot d(1, 2, 3, 9, 11, 14)$

Solution

(a) Given Boolean function is

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

The function is defined in terms of minterms and don't-care conditions. K-map representation of the given function is shown in the Fig. 3.65(a). The simplified Boolean expression is

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

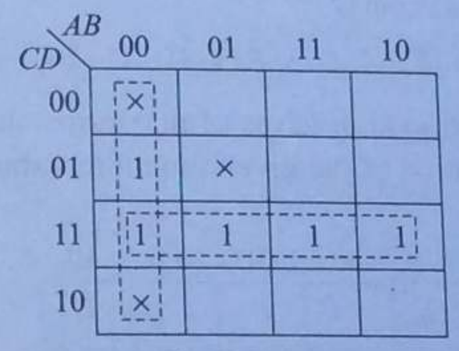


Fig. 3.65(a)

(b) Given Boolean function is

$$Y(A, B, C, D) = \Pi M(4, 5, 6, 7, 8, 12) \cdot d(1, 2, 3, 9, 11, 14)$$

The function is defined in terms of maxterms and don't-care conditions. K-map representation of the given function is shown in Fig. 3.65(b).

The simplified Boolean expression is

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

From the above problem, it is clear that × mark in a cell may be assumed to be 1 or 0 if it is necessary to form a larger group of adjacent ones or zeros, otherwise it is neglected.

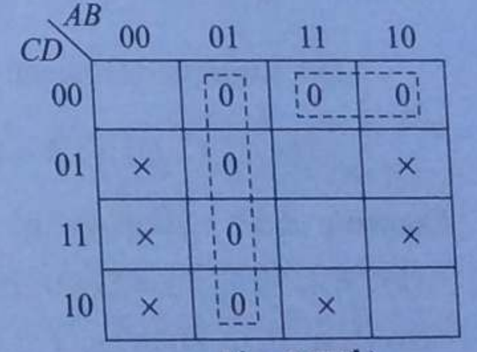


Fig. 3.65(b)

Example 3.38 Minimize the following Boolean functions:

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

(b) $Y(A, B, C, D) = \Pi M(1, 2, 3, 8, 9, 10, 11, 14) \cdot d(7, 15)$

Solution

(a) Given Boolean function is

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

The function is defined in terms of minterms and don't-care conditions. K-map representation of the given function is shown in Fig. 3.66(a).

The simplified Boolean expression is

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

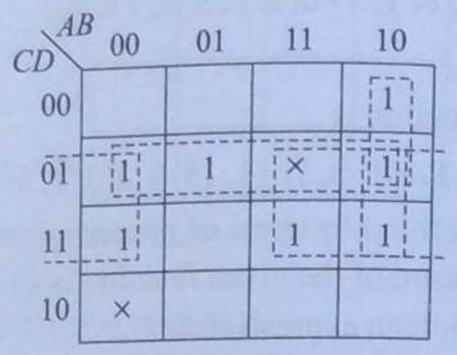


Fig. 3.66(a)

(b) Given Boolean function is

$$Y(A, B, C, D) = \Pi M(1, 2, 3, 8, 9, 10, 11, 14) \cdot d(7, 15)$$

The function is defined in terms of maxterms and don't-care conditions. K-map representation of the given function is shown in Fig. 3.66(b)

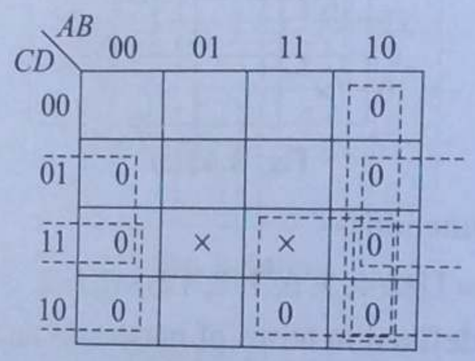


Fig. 3.66(b)

The simplified Boolean expression is

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

Example 3.39 Simplify the following functions:

(a) $Y(A, B, C, D) = \Sigma m(0, 1, 3, 7) + d(2, 5)$

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

Solution

(a) The Boolean function is

$$Y(A, B, C, D) = \Sigma m(0, 1, 3, 7) + d(2, 5)$$

The function is defined in terms of minterms and don't-care conditions. K-map representation of the given function is shown in Fig. 3.67(a).

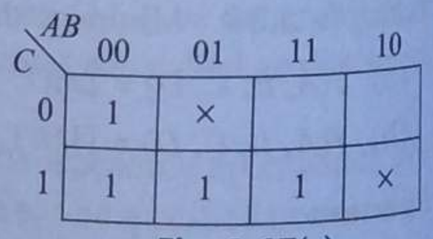


Fig. 3.67(a)

The simplified Boolean expression is $Y(A, B, C, D) = \overline{A} + C$

(b) The Boolean function is

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

The function is defined in terms of minterms and don't-care conditions. K-map representation of the given function is shown in Fig. 3.67(b).

The simplified Boolean function is

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

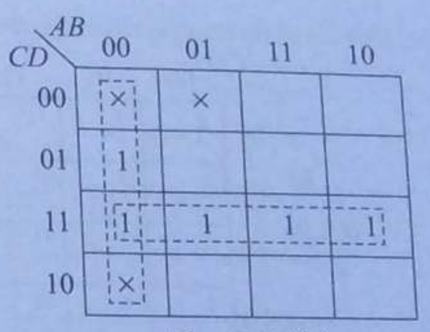


Fig. 3.67(b)

3.11 FIVE- AND SIX-VARIABLES K-MAP

The K-map representation and minimization for 2, 3, and 4 variable functions is discussed in previous sections. K-map can also be used to simplify the Boolean expressions of five and six variables.

A five-variable K-map requires $2^5 = 32$ cells and a six-variable K-map requires $2^6 = 64$ cells. A five-variable K-map is shown in Fig. 3.68.

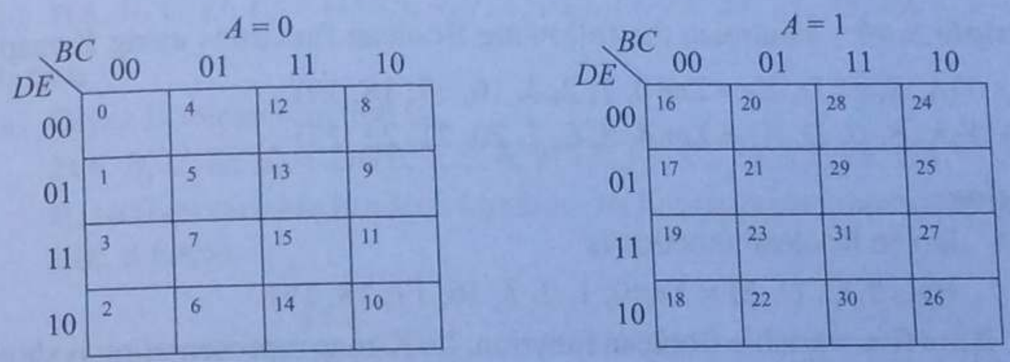


Fig. 3.68 Five-variables K-map

It consists of two four-variable K-maps, the variables are B, C, D and E. For the first map, $A = 0$, and for the second map, $A = 1$.

A six-variable K-map is shown in Fig. 3.69.

It consists of four four-variable K-maps, the variables are C, D, E, and F. For the first map, $A = 0$ and $B = 0$; for the second map, $A = 1$ and $B = 0$; for the third map, $A = 0$ and $B = 1$; and for the fourth map, $A = 1$ and $B = 1$.

In five- and six-variable K-maps, it is very difficult to identify adjacent cells to form the groups. To form the groups of adjacent cells, remember the following points:

1. Two cells are said to be adjacent if they differ by one variable.
2. Four cells are said to be adjacent if they differ by two variables.
3. Eight cells are said to be adjacent if they differ by three variables.
4. Sixteen cells are said to be adjacent if they differ by four variables.

The same logic can be extended for groups of 32 and 64 cells.

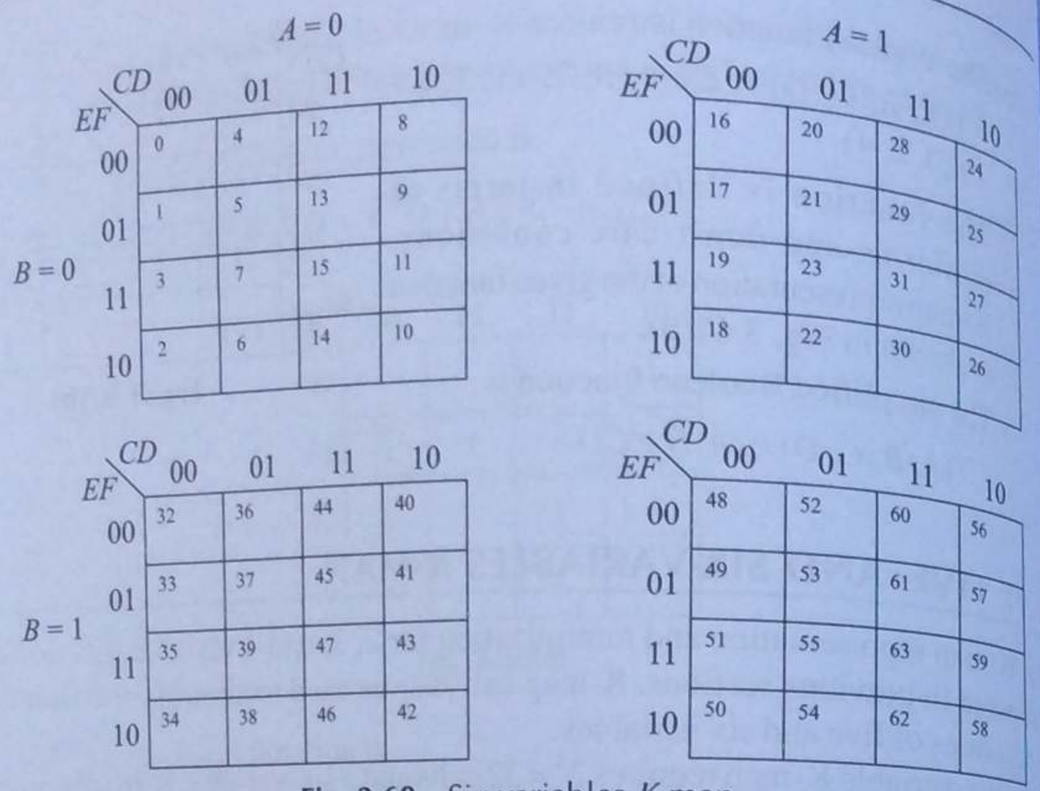


Fig. 3.69 Six-variables K-map

Example 3.40 Minimize the following Boolean functions using K-maps:

(a) $Y(A, B, C, D, E) = \Sigma m(0, 1, 2, 3, 16, 17, 18, 19)$

(b) $Y(A, B, C, D, E) = \Sigma m(4, 5, 6, 7, 20, 21, 22, 23)$

Solution

(a) Given Boolean function is

$Y(A, B, C, D, E) = \Sigma m(0, 1, 2, 3, 16, 17, 18, 19)$

It is a five-variable Boolean function. Its K-map representation is shown in Fig. 3.68(a).

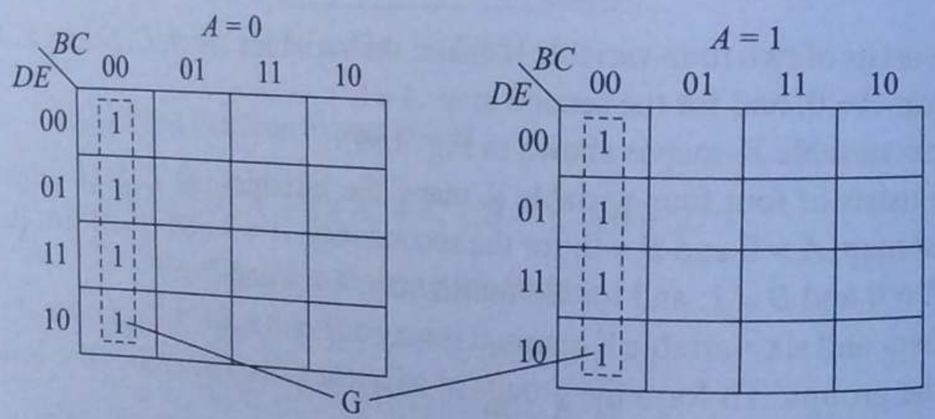


Fig. 3.68 (a) $Y(A, B, C, D, E) = \Sigma m(0, 1, 2, 3, 16, 17, 18, 19)$

The simplified Boolean expression is $Y(A, B, C, D, E) = \overline{B}\,\overline{C}$

(b) Given Boolean function is

$Y(A, B, C, D, E) = \Sigma m(4, 5, 6, 7, 20, 21, 22, 23)$

It is a five-variable Boolean function. Its K-map representation is shown in Fig. 3.68(b).

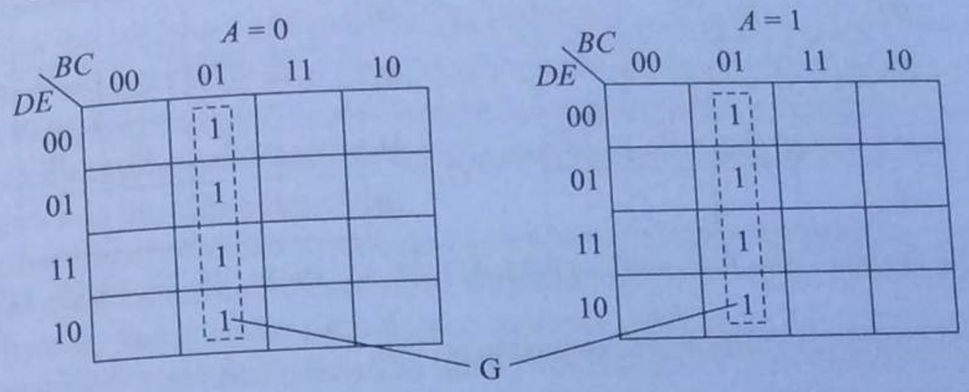


Fig. 3.68(b) $Y(A, B, C, D, E) = \Pi m(4, 5, 6, 7, 20, 21, 22, 23)$

The simplified Boolean expression is $Y(A, B, C, D, E) = \overline{B}\,C$

Example 3.41 Minimize the following Boolean functions using K-maps:

(a) $Y(A, B, C, D, E) = \Sigma m(0, 1, 5, 6, 9, 13, 14, 17, 21, 22, 25, 29)$

(b) $Y(A, B, C, D, E) = \Pi M(3, 4, 7, 11, 15, 19, 21, 23, 27, 28, 29, 31)$

Solution

(a) Given Boolean function is

$Y(A, B, C, D, E) = \Sigma m(0, 1, 5, 6, 9, 13, 14, 17, 21, 22, 25, 29)$

It is a five-variable Boolean function. Its K-map representation is shown in Fig. 3.69(a).

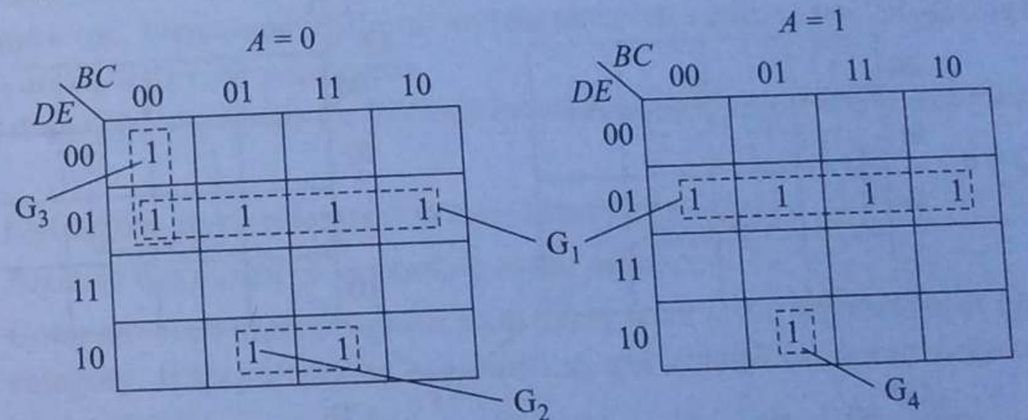


Fig. 3.69(a) $Y(A, B, C, D, E) = \Sigma m(0, 1, 5, 6, 9, 13, 14, 17, 21, 22, 25, 29)$

The simplified Boolean expression is

$$Y(A, B, C, D, E) = \overline{D}\,E + \overline{A}\,CD\overline{E} + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}CD\overline{E}$$

(b) Given Boolean function is

$Y(A, B, C, D, E) = \Pi M(3, 4, 7, 11, 15, 19, 21, 23, 27, 28, 29, 31)$

It is a five-variable Boolean function. Its K-map representation is shown in Fig. 3.69(b).

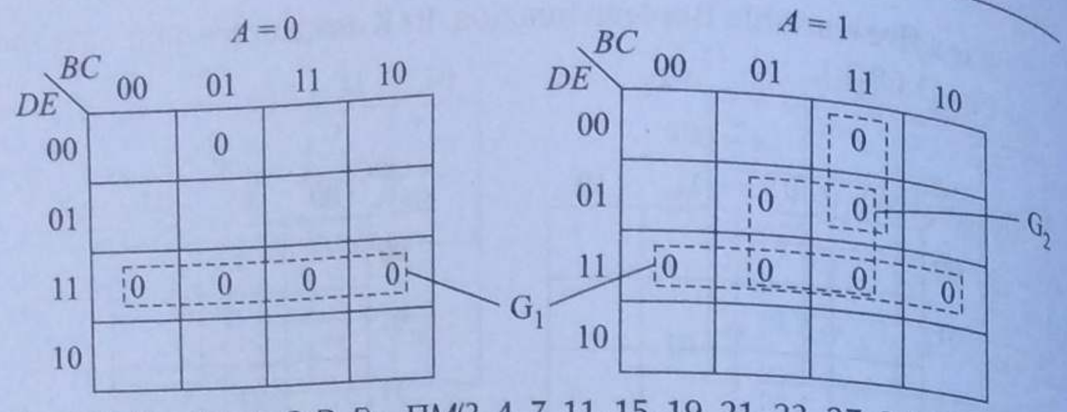


Fig. 3.69(b) $Y(A, B, C, D, E) = \Pi M(3, 4, 7, 11, 15, 19, 21, 23, 27, 28, 29, 31)$

The simplified Boolean expression is

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

Example 3.42 Minimize the following Boolean function:

$$Y(A, B, C, D, E, F) = \Sigma m(0, 1, 2, 3, 5, 7, 13, 15, 21, 23, 29, 31, 37, 39, 45, 47, 53, 55, 61, 63)$$

Solution

Given Boolean function is

$$Y(A, B, C, D, E, F) = \Sigma m(0, 1, 2, 3, 5, 7, 13, 15, 21, 23, 29, 31, 37, 39, 45, 47, 53, 55, 61, 63)$$

It is a six-variable Boolean function. Its K-map representation is shown in Fig. 3.70.

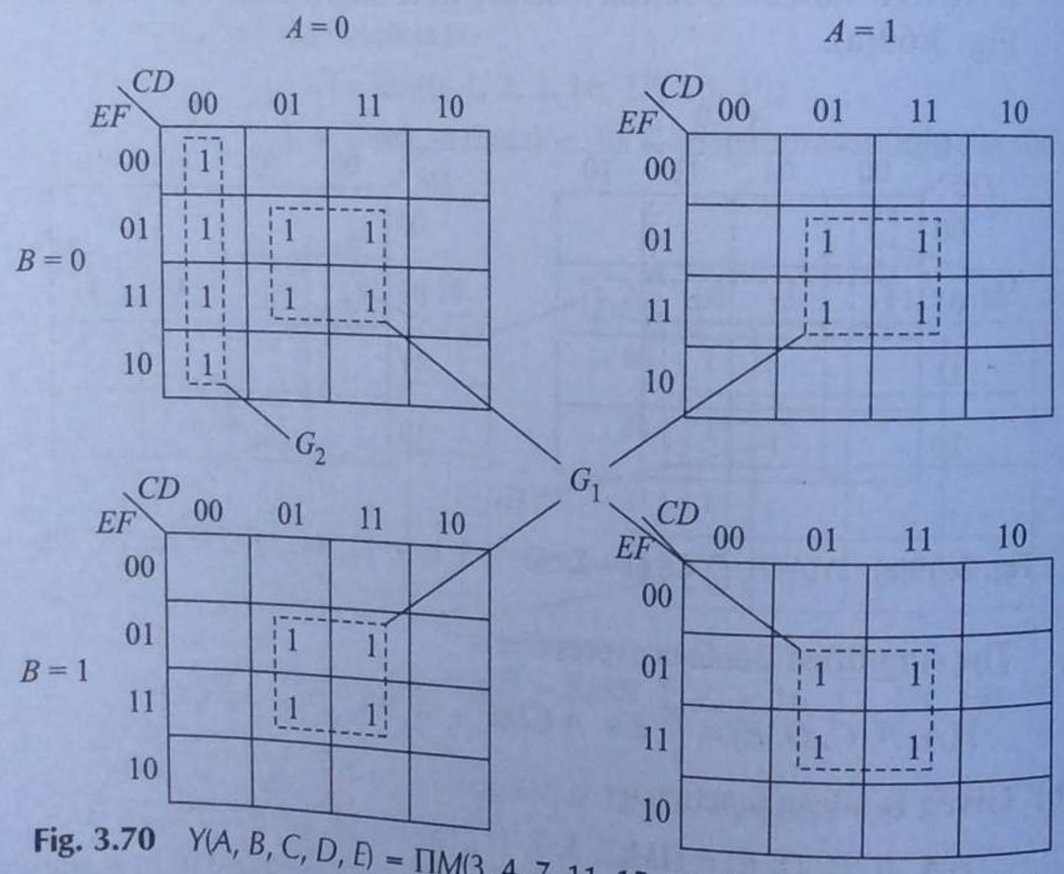


Fig. 3.70 $Y(A, B, C, D, E) = \Pi M(3, 4, 7, 11, 15, 19, 21, 23, 27, 28, 29, 31)$

The simplified Boolean expression is $Y(A, B, C, D, E, F) = DF + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$

3.12 QUINE–McCLUSKEY METHOD

Simplification of a Boolean equation using K-map method is simple and convenient as long as the variables are upto six. As the number of variables becomes more than six, it becomes difficult to form the group and simplify the Boolean expression. Hence W.V. Quine and E.T. McCluskey developed an exact method to simplify the Boolean expression. This method is known as Quine–McCluskey method. It is also known as the *tabular method*.

In simplification of Boolean expressions using K-map, we observed that the adjacent minterms could be reduced. These minterms are reduced because they differ by only one variable (literal). For example, ABC and $AB\overline{C}$ can be reduced because only C variable differs. The binary equivalents of these minterms are 111 and 110. From the binary equivalents, it is clear that the minterms, whose binary equivalent differ only in one place, can be combined to reduce the minterms. This is the basic principle of Quine–McCluskey method.

In this method, the given function of minterms is written in terms of its binary equivalents. These minterms are grouped according to the number of 1s contained and separated by a horizontal line between each number of 1s category. Once the separation is over, each binary number is compared with every term in the next higher category and if they differ by only one position, a check mark is placed beside each of the two terms and then the term is copied in the second column with a mark '•' in the position that they differed. For example, if the terms are 0001 and 0000, then the resultant term will be 000_. This term 000_ is called an *implicant*. It implies that $\overline{A}\,\overline{B}\,\overline{C}$ could be a term in the final expression covering the minterms $\overline{A}\,\overline{B}\,\overline{C}D$ and $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$. This process of comparison is repeated for every minterm. Once this process is completed, the same process is applied to the new resultant terms and so on. The remaining terms and the terms which do not match during the process are called *prime implicants*.

The steps used to simplify the Boolean function using Quine–McCluskey method are:

1. List all the given minterms in their binary equivalent.
2. Arrange the minterms according to the number of 1s.
3. Compare each binary number with every term in the adjacent next higher category. If they differ by one position, put a check mark and write in the next column.
4. Repeat Step 3 for the resultant column and continue these cycles until no further elimination of variables takes place.
5. List the primary implicants.
6. Select the minimum number of primes that must cover all the minterms.

Let us discuss the steps with an example. Consider

$$Y = \Sigma m(0, 2, 3, 6, 7, 8, 10, 12, 13)$$

1. The given minterms in the binary equivalent are

0 = 0000	2 = 0010	3 = 0011	6 = 0110
7 = 0111	8 = 1000	12 = 1100	13 = 1101

Table 3.11 Minterms in the binary equivalent

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

2. Arrangement of minterms according to number of 1s.

Table 3.12 Arrangement of minterms

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

3. Compare each binary number with every term in the adjacent next higher category and if they differ by one position, put a check mark.
When the minterms m_0 and m_2 are compared; they differ in second position and hence put a check mark (_) at the second position and the group (0, 2) is represented by 00_0.
When the minterms m_0 and m_8 are compared, they differ in fourth position, and hence put a check mark (_) at the fourth position and the group (0, 8) is represented by _000.
When the minterms m_2 and m_3 are compared, they differ in first position, and hence put a check mark (_) at the first position and the group (2, 3) is represented by 001_.
Similarly compare the groups (2, 6) and (2, 10).
When the minterms m_2 and m_{12} are compared, they differ in two positions, and hence the group (2, 12) cannot be formed, and so on. The binary representation of a group is given in Table 3.13. Refer columns 1 and 2.
4. When the groups of minterms (0, 2) and (2, 3) are compared, they differ by two positions, and hence the group (0, 2, 3) cannot be formed. Similarly (0, 2, 6) and (0, 2, 0).
When the groups of minterms (0, 2) and (8, 10) are compared, they differ by fourth position, and hence put a check mark (_) at the fourth position. Group (0, 2, 8, 10) is represented by _0 _0. Similarly group (2, 3, 6, 7) is represented by 0_1 0 as given in Table 2.13. Refer columns 3 and 4.

Table 3.13 Binary representation of a group

Minterms	*Binary representation with check mark*	*Minterms*	*Binary representation with check mark*
0, 2	0 0_0 ✓		
0, 8	_0 0 0 ✓		
2, 3	0 0 1_ ✓		
2, 6	0_1 0 ✓		_0_0
2, 10	_0 1 0 ✓	0, 2, 8, 10	
8, 10	1 0_0 ✓	2, 3, 6, 7	0_1_
8, 12	1_0 0		
3, 7	0_1 1 ✓		
6, 7	0 1 1_ ✓		
12, 13	1 1 0_		

5. List the prime implicants. The groups of minterms in the above table without check marks are known as prime implicants and it is given in Table 3.14 with binary representation.

Table 3.14 Prime implicant with binary representation

Prime implicants	*Binary representation*
8, 12	1_0 0
12, 13	1 1 0_
0, 2, 8, 10	_0_0
2, 3, 6, 7	0_1_

6. Select the minimum number of primes that must cover all the minterms. The prime implicants selection chart is shown in Table 3.15.

Table 3.15 Prime implicants selection chart

Prime implicants	m_0	m_2	m_3	m_6	m_7	m_8	m_{10}	m_{12}	m_{13}
8, 12						•		•	
12, 13								•	•
0, 2, 8, 10	•	•				•	•		
2, 3, 6, 7		•	•	•	•				

It is clear from the selection chart that it is sufficient to select the prime implicants (2, 3, 6, 7) and (0, 2, 8, 10) and (12, 13).
It is not necessary to select the prime implicants (8, 12) because 8 is present in (0, 2, 8, 10), and 12 is present in (12, 13) and hence

$$Y(A, B, C, D) = (110_) + (_0\ _0) + (0\ _1\ _)$$
$$= AB\overline{C} + \overline{B}\,\overline{D} + \overline{A}\,C$$

Example 3.43 Simplify the following four-variable Boolean functions using Quine–McCluskey method.

$$Y(A, B, C, D) = \Sigma m(2, 4, 5, 9, 12, 13)$$

Solution

Given Boolean function is

$$Y(A, B, C, D) = \Sigma m(2, 4, 5, 9, 12, 13)$$

Step 1: The given minterms and their binary equivalents are shown in Table 2.16(a). (Refer columns 1 and 2 of Table 3.16(a)).

Step 2: The arrangement of minterms according to categories of 1s is shown in Table 3.16(a). (Refer columns 3 and 4 of Table 3.16(a)).

Table 3.16(a) Binary equivalents of groups of minterms

Minterms	*Binary representation*	*Minterms*	*Binary representation*
m_2	0 0 1 0	m_2	0 0 1 0
m_4	0 1 0 0	m_4	0 1 0 0 ✓
m_5	0 1 0 1	m_5	0 1 0 1 ✓
m_9	1 0 0 1	m_9	1 0 0 1 ✓
m_{12}	1 1 0 0	m_{12}	1 1 0 0 ✓
m_{13}	1 1 0 1	m_{13}	1 1 0 1 ✓

Step 3: Compare each binary number and prepare the term with check mark. (Refer columns 1 and 2 of Table 3.16(b)).

Step 4: Apply the above procedure for the result and repeat until no further elimination takes place.

Table 3.16(b) Binary equivalents of groups of minterms with check marks

Minterms	*Binary representation with check mark*	*Minterms*	*Binary representation with check mark*
4, 5	0 1 0_ ✓	4, 5, 12, 13	_1 0_
4, 12	_1 0 0 ✓		
5, 13	_1 0 1 ✓		
9, 13	1_0 1		
12, 13	1 1 0_ ✓		

Step 5: The list of prime implicants is shown in Table 3.16 (c).

Table 3.16(c) List of prime implicants

Prime implicants	*Binary representation*
2	0 0 1 0
9, 13	1_0 1
4, 5, 12, 13	_1 0_

Step 6: Select the minimum number of primes that must cover all the minterms. The prime implicants selection chart is shown in Table 3.16(d).

Table 3.16(d) Prime implicants selection chart

Prime implicants	m_2	m_4	m_5	m_9	m_{12}	m_3
2		•				
9, 13				•		•
4, 5, 12, 13		•	•		•	•

$$Y(A, B, C, D) = (1_01) + (_10_)$$

Example 3.44 Simplify the following Boolean expression using Quine–McCluskey method.

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

Solution

Given Boolean expression is

$$Y(A, B, C, D) = \underset{(0100)}{\overline{A}\,B\,\overline{C}\,\overline{D}} + \underset{(1000)}{A\,\overline{B}\,\overline{C}\,\overline{D}} + \underset{(0001)}{\overline{A}\,\overline{B}\,\overline{C}\,D} + \underset{(1101)}{AB\,\overline{C}\,D} + \underset{(1100)}{AB\,\overline{C}\,\overline{D}}$$

In minterms, the function can be written as,

$$Y(A, B, C, D) = \Sigma m(1, 4, 8, 12, 13)$$

Step 1: The given minterms and their binary equivalents are shown in Table 3.17(a). (Refer columns 1 and 2 of Table 3.17(a)).

Step 2: Arrangement of minterms according to number of 1s is shown in Table 3.17(a). (Refer columns 3 and 4 of Table 3.17(a)).

Table 3.17(a) Minterms in the binary equivalent and groups of minterms

Minterms	*Binary representation*	*Minterms*	*Binary representation*
m_1	0 0 0 1	m_1	0 0 0 1
m_4	0 1 0 0	m_4	0 1 0 0 ✓
m_8	1 0 0 0	m_8	1 0 0 0 ✓
m_{12}	1 1 0 0	m_{12}	1 1 0 0 ✓
m_{13}	1 1 0 1	m_{13}	1 1 0 1 ✓

Step 3: Compare each binary number with every term in the adjacent next higher category and if they differ by one position, put a check mark. (Refer columns 1 and 2 of Table 3.17(b)).

Step 4: Apply the same procedure for the result and repeat until no further elimination takes place. (Refer columns 1 and 2 of Table 3.17(b)).

Table 3.17(b) Binary equivalents of groups of minterms with check marks

Minterms	*Binary representation*	*Minterms*	*Binary representation*
4, 12	_1 0 0	NIL	NIL
8, 12	1_0 0		
12, 13	1 1 0_		

Step 5: The list of prime implicants is shown in Table 3.17(c).

Table 3.17(c) List of prime implicants

Prime implicants	*Binary representation*
1	0 0 0 1
4, 12	_1 0 0
8, 12	1_0 0
12, 13	1 1 0_

Step 6: Select the minimum number of primes that must cover all the minterms. The prime implicant selection chart is shown in Table 3.17(d).

Table 3.17(d) Prime implicant selection chart

Prime implicants	m_1	m_4	m_8	m_{12}	m_{13}
1	•				
4, 12		•		•	
8, 12			•	•	
12, 13				•	•

$$Y(A, B, C, D) = (0001) + (_100) + (1_00) + (110_)$$

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

Example 3.45 Simplify the following function using Quine–McCluskey method.

$$Y(A, B, C, D) = \Pi M(1, 4, 6, 9, 10, 11, 14, 15)$$

Solution

Given Boolean expression is

$$Y(A, B, C, D) = \Pi M(1, 4, 6, 9, 10, 11, 14, 15)$$

This expression is in maxterms. It can be written in minterms as

$$Y(A, B, C, D) = \Sigma m(0, 2, 3, 5, 7, 8, 12, 13)$$

Step 1: The list of given minterms and their binary equivalents are shown in Fig. 3.18(a). (Refer columns 1 and 2 of Table 3.18(a)).

Step 2: The arrangement of minterms according to number of 1s is shown in Fig. 3.18(a). (Refer columns 3 and 4 of Table 3.18(a)).

Table 3.18(a) Minterms in the binary equivalent and groups of minterms

Minterms	*Binary representation*	*Minterms*	*Binary representation*
m_0	0 0 0 0	m_0	0 0 0 0 ✓
m_2	0 0 1 0	m_2	0 0 1 0 ✓
m_3	0 0 1 1	m_8	1 0 0 0 ✓
m_5	0 1 0 1	m_3	0 0 1 1 ✓
m_7	0 1 1 1	m_5	0 1 0 1 ✓
m_8	1 0 0 0	m_{12}	1 1 0 0 ✓
m_{12}	1 1 0 0	m_7	0 1 1 1 ✓
m_{13}	1 1 0 1	m_{13}	1 1 0 1 ✓

Step 3: Compare each binary number with every term in the adjacent next higher category and if they differ by one position, put a check mark. (Refer columns 1 and 2 of Table 3.18(b)).

Step 4: Apply the same procedure for the result and repeat until no further elimination takes place. (Refer columns 1 and 2 of Table 3.18(b)).

Table 3.18(b) Binary equivalents of groups of minterms with check marks

Minterms	*Binary representation*	*Minterms*	*Binary representation*
0, 2	0 0_0	NIL	NIL
0, 8	_0 0 0		
2, 3	0 0 1_		
8, 12	1_0 0		
3, 7	0_1 1		
5, 7	0 1_1		
5, 13	_1 0 1		
12, 13	1 1 0_		

Step 5: The list of prime implicants is shown in Table 3.18(c).

Table 3.18(c) List of prime implicants

Prime implicants	*Binary representation*
0, 2	0 0_0
0, 8	_0 0 0
2, 3	0 0 1_
8, 12	1_0 0
3, 7	0_1 1
5, 7	0 1_1
5, 13	_1 0 1
12, 13	1 1 0_

Step 6: Select the minimum number of primes that must cover all the minterms. The prime implicant selection chart is shown in Table 3.18(d).

Table 3.18(d) Prime implicant selection chart

Prime implicants	m_0	m_2	m_3	m_5	m_7	m_8	m_{12}	m_{13}
0, 2 ✓	•	•						
0, 8 ✓	•					•		
2, 3		•	•					
8, 12						•	•	
3, 7 ✓			•		•			
5, 7				•	•			
5, 13 ✓				•				•
12, 13 ✓							•	•

$$Y(A, B, C, D) = (110_) + (_101) + (01_1) + (_000) + (00_0)$$

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

Example 3.46 Simplify the following Boolean function using Quine-McCluskey method.

$$Y(A, B, C, D) = \Sigma m(1, 2, 3, 5, 9, 12, 14, 15) + d(4, 8, 11)$$

Solution

Given Boolean expression is

$$Y(A, B, C, D) = \Sigma m(1, 2, 3, 5, 9, 12, 14, 15) + d(4, 8, 11)$$

Step 1: A list of given minterms and don't-care conditions and their binary equivalents is shown in Table 3.19(a). (Refer columns 1 and 2 of Table 3.19(a)).

Step 2: The arrangement of the minterms according to number of 1s is shown in Table 3.19(a). (Refer columns 3 and 4 of Table 3.19(a)).

Table 3.19(a) Minterms, don't-care conditions; their binary equivalents, and groups of minterms

Minterms	*Binary representation*	*Minterms*	*Binary representation*
m_1	0 0 0 1	m_1	0 0 0 1 ✓
m_2	0 0 1 0	m_2	0 0 1 0 ✓
m_3	0 0 1 1	dm_4	0 1 0 0 ✓
m_5	0 1 0 1	dm_8	1 0 0 0 ✓
m_9	1 0 0 1	m_3	0 0 1 1 ✓
m_{12}	1 1 0 0	m_5	0 1 0 1 ✓
m_{14}	1 1 1 0	m_9	1 0 0 1 ✓
m_{15}	1 1 1 1	m_{12}	1 1 0 0 ✓
dm_4	0 1 0 0	dm_{11}	1 0 1 1 ✓
dm_8	1 0 0 0	m_{14}	1 1 1 0 ✓
dm_{11}	1 0 1 1	m_{15}	1 1 1 1 ✓

Step 3: Compare each binary number with every term in the adjacent next higher category. If they differ by one position, put a check mark. (Refer columns 1 and 2 of Table 3.19(b)).

Step 4: Apply the same procedure for the result and repeat until no further elimination takes place. (Refer columns 1 and 2 of Table 2.37(b)).

Table 3.19(b) Binary equivalents of groups of minterms with check marks

Minterms	*Binary representation*	*Minterms*	*Binary representation*
1, 3	0 0_1		
1, 5	0 _0 1		
1, 9	_0 0 1		
2, 3	001_	1, 3, 9, d11	_0_1
4, 5	01 0_		
4, 12	_100		
d8, 9	100_		
d8, 12	1 0 0		
3, d11	_0 1 1		
9, d11	10_1		
12, 14	11 0		
d11, 15	1_11		
14, 15	111_		

Step 5: The list of prime implicants is shown in Table 3.19(a).

Table 3.19(c) List of prime implicants

Prime implicants	*Binary representation*
1, 5	0_0 1
2, 3	0 0 1_
d4,5	0 1 0_
d4, 12	_1 0 0
d8, 9	1 0 0_
d8, 12	1_0 0
12, 14	1 1_0
d11, 15	1_1 1
14, 15	1 1 1_
1, 3, 9, d11	_0_1

Step 6: Select the minimum number of primes that must cover all the minterms. The prime implicant selection chart is shown in Table 3.19 (d).

Table 3.19(d) Prime implicant selection chart

Prime implicants	m_1	m_2	m_3	m_5	m_9	m_{12}	m_{14}	m_{15}	d_{m4}	d_{m8}	d_{m11}
1, 5 ✓	•			•							
2, 3 ✓		•	•								
d4, 5				•					•		
d4, 12						•			•		
d8, 9					•					•	
d8, 12						•				•	
12, 14 ✓						•	•				
d11, 15								•			•
14, 15 ✓							•	•			
1, 3, 9, d11 ✓	•		•		•						•

$$Y(A, B, C, D) = (_0_1) + (111_) + (11_0) + (0_01) + (001_)$$

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

It is important to note that it is not compulsory to include don't-care terms in the final expression.

Example 3.47 Simplify the following four-variable Boolean function using Quine–McCluskey method.

$$Y(A, B, C, D) = \Sigma m(1, 3, 5, 10, 11, 12, 13, 14, 15)$$

Solution

Given Boolean function is

$$Y(A, B, C, D) = \Sigma m(1, 3, 5, 10, 11, 12, 13, 14, 15)$$

Step 1: A list of given minterms and their binary equivalents are shown in Table 3.20(a). (Refer columns 1 and 2 of Table 3.20(a)).

1 = 0001 3 = 0011 5 = 0101 10 = 1010
11 = 1011 12 = 1100 13 = 1101 14 = 1110
15 = 1111

Step 2: Arrangement of the minterms according to number of 1s is shown in Table 3.20(a). (Refer columns 3 and 4 of Table 3.20(a)).

Table 3.20(a) Minterms, their binary equivalents, and groups of minterms

Minterms	*Binary representation*	*Minterms*	*Binary representation*
m_1	0 0 0 1	m_1	0 0 0 1
m_3	0 0 1 1	m_3	0 0 1 1
m_5	0 1 0 1	m_5	0 1 0 1
m_{10}	1 0 1 0	m_{10}	1 0 1 0
m_{11}	1 0 1 1	m_{12}	1 1 0 0
m_{12}	1 1 0 0	m_{11}	1 0 1 1
m_{13}	1 1 0 1	m_{13}	1 1 0 1
m_{14}	1 1 1 0	m_{14}	1 1 1 0
m_{15}	1 1 1 1	m_{15}	1 1 1 1

Step 3: Compare each binary number with every term in the adjacent next higher category; and if they differ by one position, put a check mark. (Refer Table 3.20(b)).

Step 4: Apply the same procedure for the result and repeat until no further elimination takes place. (Refer Tables 3.20(c) and 3.20(d)).

Table 3.20(b) Binary equivalents of groups of minterms with check marks

Minterms	*Binary representation with check mark*
1, 3	0 0_1
1, 5	0_0 1
3, 11	_0 1 1
5, 13	_1 0 1
10, 11	1 0 1_ ✓
10, 14	1_1 0 ✓
12, 13	1 1 0_ ✓
12, 14	1 1_0 ✓
11, 15	1_1 1 ✓
13, 15	1 1_1 ✓
14, 15	1 1 1_ ✓

The list of second reduction table is shown in Table 3.20(c).

Table 3.20(c) Binary equivalents of groups of minterms with check marks

Minterms	*Binary representation*
$m_{10}, m_{11}, m_{14}, m_{15}$	1_1_
$m_{12}, m_{14}, m_{13}, m_{15}$	1 1_ _
$m_{12}, m_{13}, m_{14}, m_{15}$	1 1_ _
$m_{10}, m_{14}, m_{11}, m_{15}$	1_1_

The list of third reduction table is shown in Table 3.20(d).

Table 3.20(d) Binary equivalents of groups of minterms with check marks

Minterms	*Binary representation*	
$m_{10}, m_{11}, m_{14}, m_{15}$	1_1_	These are the essential prime implicants
$m_{10}, m_{14}, m_{11}, m_{15}$		
$m_{12}, m_{14}, m_{13}, m_{15}$	1 1_ _	
$m_{12}, m_{13}, m_{14}, m_{15}$		

Step 5: The list of prime implicants is shown in Table 3.20(e).

Table 3.20(e) List of prime implicants

Prime implicants	*Binary representation*
1, 3	00_1
1, 5	0_01
3, 11	_011
5, 13	_101

Step 6: Select the minimum number of primes that must cover all the minterms. The prime implicant selection chart is shown in Table 3.20(f).

Table 3.20(f) Prime implicant selection chart

Prime implicants	m_1	m_3	m_5	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_{15}
1, 3	•	•							
1, 5	•		•						
3, 11		•			•				
5, 13			•				•		
10, 11, 14, 15				•	•			•	•
12, 13, 14, 15						•	•	•	•
				✓	✓	✓	✓	✓	✓

It is clear from the selection chart that it is sufficient to select the prime implicants (1, 3) and (1, 5) along with (10, 11, 14, 15) and (12, 13, 14, 15).

It is not necessary to select the prime implicants (3, 11) and (5, 13) because 3 is present in (1, 3) and 5 is present in (1, 5). Hence,

$$Y(A, B, C, D) = (00_1) + (0_01) + (1_1_) + (11__)$$

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

SUMMARY

- DeMorgan's first theorem states that the complement of sum of digital signals is equal to the product of its complements, i.e.

$$\overline{A+B+C+...+N} = \overline{A} \cdot \overline{B} \cdot \overline{C} \ldots \overline{N}$$

- DeMorgan's second theorem states that the complement of a product of digital signals is equal to the sum of its complements, i.e.

$$\overline{A \cdot B \cdot C \cdot ... \cdot N} = \overline{A} + \overline{B} + \overline{C} + ... + \overline{N}$$

- If each term in the sum of products contains all the variables (literals), then the expression is known as the standard sum of products or the canonical sum of products. Each individual term in a standard sum of products is called as the minterm.
- If each term in the product of sums contains all the variables, then the expression is known as the standard product of sums or the canonical product of sums. Each individual term in a standard product of sums is called as the maxterm.
- Different methods are used to simplify Boolean expressions. These are:
  1. Algebraic simplification
  2. Karnaugh map method
  3. Quine–McCluskey method

EXERCISES

Review Questions

1. State and prove the Boolean theorems.
2. Write the basic laws of Boolean algebra.
3. Explain the importance of Boolean algebra in digital system.
4. For the function given below

$$f = ABC + B\overline{C}D + AB\overline{C}$$

   (i) Express f in standard SOP form.
   (ii) Simplify using K-map.
   (iii) Realize f using NAND gates only.

5. Reduce the following equation using Quine–McCluskey method of minimization:

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

6. Minimize the following expression using K-map and realize using NOR gates only:

$$f(P, Q, R, S) = \Pi M(1, 4, 6, 9, 10, 11, 14, 15)$$

7. Design and implement a 4-bit BCD to Excess-3 code converter using suitable gates.
8. Using K-map, simplify the equation:

$$F(A, B, C, D) = \Sigma m(0, 2, 8, 9) + d\ (3, 7, 10, 11, 14, 15)$$

   Obtain the minimal SOP expression and implement it in NAND logic.
9. What is meant by standard SOP form? Convert the following function in standard SOP form:

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

10. Simplify the following using K-maps and implement it using basic gates only.
    (i) $f(a, b, c, d) = \Sigma m(2, 3, 5, 13, 14) + d\ (8, 9, 10, 11)$
    (ii) $f(P, Q, R, S) = \Pi M(1, 3, 8, 10, 12, 13, 14, 15)$
11. Simplify the following using K-maps and implement it using basic gates only.
    (i) $f(A, B, C, D) = \Sigma m(0, 2, 8, 10) + d(4, 7, 6, 11, 15)$
    (ii) $f(A, B, C, D) = \overline{C}D + B\overline{C}D + BC\overline{D} + A\overline{B}C\overline{D}$
12. Design a 4-bit gray-to-binary converter using truth table, K-map, and logic circuits.
13. Obtain the simplified expressions in product of sums form using K-maps:
    (i) $F(w, x, y, z) = \Pi M(1, 3, 5, 7, 13, 15)$
    (ii) $F(x, y, z) = \overline{xz} + \overline{yz} + \overline{yz} + xyz$

Multiple Choice Questions

Select the correct alternative.

1. The Boolean equation of AND operation is.....
   (a) $Y = \overline{A}$
   (b) $Y = AB$
   (c) $Y = A + B$
2. In a half adder having two inputs A and B and two outputs (S and C are the sum and carry output bits respectively), the Boolean expression for S and C in terms of A and B is.....
   (a) $S = \overline{A}B + A\overline{B}, C = AB$
   (b) $S = AB + \overline{A}B, C = A + B$
   (c) $S = \overline{AB} + AB, C = A + \overline{B}$
   (c) $S = \overline{A} + \overline{AB}, C = \overline{A} + \overline{B}$
3. For a binary half-subtractor having two inputs A and B, the correct set of logic expressions for the outputs D(= A minus B) and X(= borrow) are .....
   (a) $D = AB + \overline{A}B, X = \overline{A}B$
   (b) $D = \overline{A}B + A\overline{B}, X = A\overline{B}$
   (c) $D = \overline{A}B + A\overline{B}, X = \overline{A}B$
   (d) $D = AB + \overline{AB}, X = A\overline{B}$
4. The logical expression $Y = A + \overline{A}B$ is equivalent to.....
   (a) $Y = AB$
   (b) $Y = \overline{A}B$
   (c) $Y = A + \overline{B}$
   (d) $Y = A + B$
5. The minimum number of NAND gates required to implement $A + A\overline{B} + A\overline{B}C$ is equal to.....
   (a) 0
   (b) 1
   (c) 4
   (d) 7
6. The dual of the Boolean theorem $A \cdot (B + C) = AB + AC$ is
   (a) $A \cdot (B + C) = AB + AC$
   (b) $A \cdot (B + C) = (A + B)(A + C)$
   (c) $A \cdot BC = (A + B)(A + C)$
   (d) None of these
7. A carry look ahead adder is frequently used for addition, because it.....
   (a) is faster
   (b) is more accurate
   (c) uses fewer gates
   (d) costs less

1. b 2. a 3. c 4. d 5. a 6. c 7

Chapter

4

COMBINATIONAL LOGIC CIRCUITS

Chapter Outline

- Design of arithmetic circuit
- Design of adder and subtractor using MSI circuits
- Design of BCD adder using 7483
- Study of ALU
- Study of parity generator and checker
- Study of comparator, multiplexer, and demultiplexer
- Code converter

4.1 INTRODUCTION

Generally, digital circuits are divided into two categories:

1. Combinational logic circuit, and
2. Sequential logic circuit.

In combinational logic circuits, outputs at any instant of time depend upon the inputs present at that instant of time. Combinational logic circuits do not have memory elements (storage device). It can be designed using gates or available ICs. Adder, subtractor, ALU comparators, parity generator and checker, multiplexer, demultiplexer, encoder, and code converters are the examples of combinational logic circuits.

4.2 DESIGN PROCEDURE FOR COMBINATIONAL LOGIC CIRCUIT

The combinational logic circuit can be implemented using gates or available ICs. The requirements of combinational logic circuits may be specified in one of the following ways:

(a) by a set of statements, or
(b) by the Boolean expressions, or
(c) by the truth table.

The steps used to design a combinational logic circuit using gates are:

1. From the set of statements or Boolean expressions or truth tables, determine the number of input and output variables.
2. Write down the expression in the form of sum of products or product of sums.
3. Simplify the Boolean expression using K-map or Quine–McCluskey method.
4. Implement the simplified Boolean expression using the logic diagram.

Now-a-days, most of the combinational logic circuits are available in the form of integrated circuits (IC). ICs of multiplexer, demultiplexer, adder, ALU, parity generator/checker, comparator and code converter are available. It is not required to simplify Boolean expressions to design combinational logic circuits using ICs, and the number of external wires and connections are also reduced. The reliability of the system improves by reducing the number of external wires and connections.

4.2.1 Examples of Combinational Logic Circuits

Example 4.1 Design and implement a 1-bit comparator using logic gates.

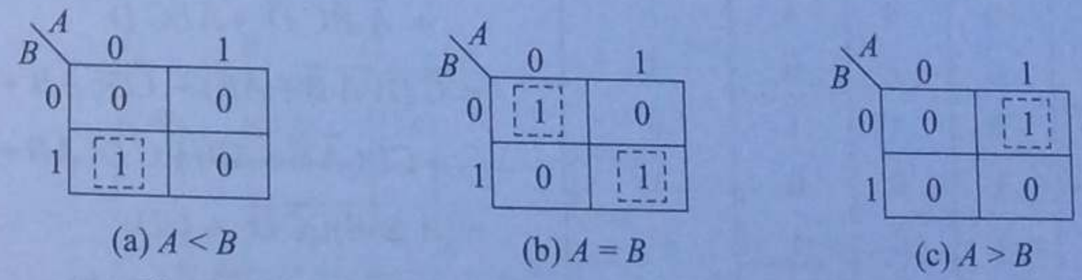
Solution

The comparator compares the magnitude of input numbers and checks whether one input number is greater or less than another input number or both the numbers are equal. Consider A and B are two 1-bit numbers. These two are the inputs of the comparator and $A = B$, $A > B$, and $A < B$ are the outputs of the comparator. The truth table of 1-bit comparator is shown in Table 4.1.

Table 4.1 Inputs and outputs of 1-bit comparator

Inputs		*Outputs*		
A	***B***	***A < B***	***A = B***	***A > B***
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

K-maps for $A < B$, $A = B$, and $A > B$ with inputs A and B are:



(a) $A < B$ (b) $A = B$ (c) $A > B$

Fig. 4.1(a) K-maps for $A < B$, for $A = B$, and $A > B$

From the K-map, we get Boolean expressions as,

(a) $A < B = \overline{A}\,B$ (b) $(A = B) = \overline{A}\,\overline{B} + AB = A \odot B$ (c) $A > B = A\overline{B}$

The logic diagram of 1-bit comparator is shown in Fig. 4.1(b).

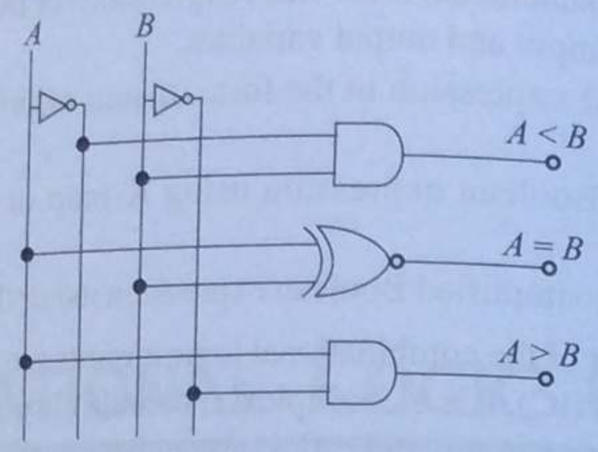


Fig. 4.1(b) Logic diagram of one-bit comparator

Example 4.2 Design a combinational logic circuit with four input variables that will produce logic 1 output when the number of 1s in the inputs is even.

Solution

Let us consider A, B, C, and D are the four inputs and Y is the output. The truth table is shown in Table 4.2.

Table 4.2 Inputs and outputs of the combinational logic circuit

Inputs				*Output*
A	*B*	*C*	*D*	*Y*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

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

Fig. 4.2(a) K-map for output Y

$$
\begin{aligned}
Y = {} & \bar{A}\bar{B}\bar{C}\bar{D} + AB\bar{C}\bar{D} + \bar{A}B\bar{C}D \\
& + A\bar{B}\bar{C}D + \bar{A}\bar{B}CD + ABCD \\
& + \bar{A}BC\bar{D} + A\bar{B}C\bar{D} \\
= {} & \bar{C}\bar{D}(\bar{A}\bar{B} + AB) + \bar{C}D(\bar{A}B + A\bar{B}) \\
& + CD(\bar{A}\bar{B} + AB) + C\bar{D}(\bar{A}B + A\bar{B}) \\
= {} & (A \odot B)(\bar{C}\bar{D} + CD) \\
& + (\bar{C}D + C\bar{D})(A \oplus B) \\
= {} & (A \odot B)(C \odot D) + (A \oplus B)(C \oplus D)
\end{aligned}
$$

The logic diagram is shown in Fig. 4.2 (b).

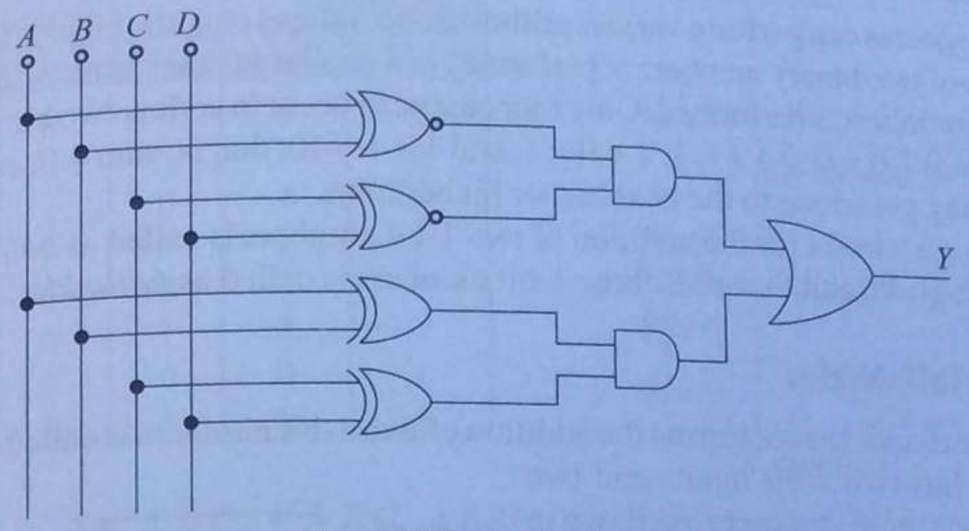


Fig. 4.2(b) Logic diagram

Example 4.3 Design a combinational logic circuit with three input variables that will produce logic 1 output when more than one input variables are logic 0.

Solution

Consider A, B, and C as input variables and Y as the output variable. Truth table for the given problem is shown in Table 4.3.

Table 4.3 Inputs and outputs of the combinational logic circuit

Inputs			*Output*
A	*B*	*C*	*Y*
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	0

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

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

Fig. 4.3(a) K-map for output Y

Logic diagram is shown in Fig. 4.3 (b).

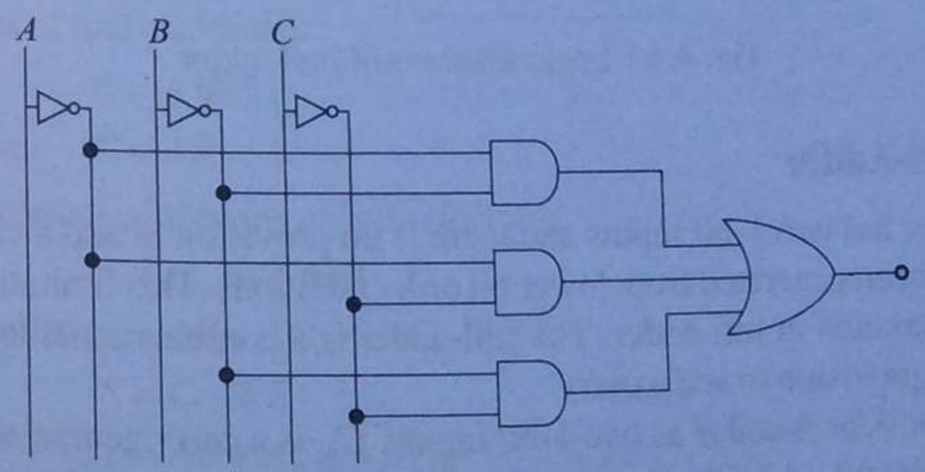


Fig. 4.3(b) Logic diagram

4.3 ADDERS

Digital systems can perform various arithmetic operations over the binary data. The addition of two binary numbers is performed in a similar manner as the addition of decimal numbers is performed. Only four cases can occur in adding two binary bits. They are, $0+0=0$, $0+1=1$, $1+0=1$, and $1+1=10$, that is, sum = 0, carry = 1 which may get added to the next higher bit addition.

The logic circuit for the addition of two 1-bit numbers is called as *half-adder*, and the logic circuit that adds three 1-bit numbers is called as *full-adder*.

4.3.1 Half-Adder

The logic circuit that performs the addition of two 1-bit numbers is called as half-adder. It has two 1-bit inputs and two outputs, viz. sum and carry as shown in Fig. 4.3.

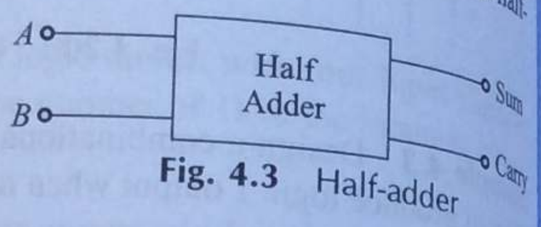
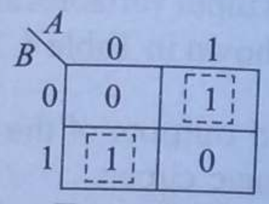


Fig. 4.3 Half-adder

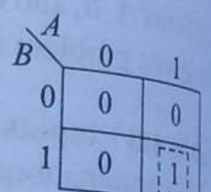
The truth table of half-adder is given in Table 4.4, where A and B are the inputs, and sum and carry are the outputs.

Table 4.4 Truth table for half-adder

Inputs		*Outputs*	
A	*B*	*Sum*	*Carry*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



K-map for Sum



K-map for Carry

Here $S = A\overline{B} + \overline{A}B = (A \oplus B)$ (4.1)

$C = AB$ (4.2)

The logic diagram of half-adder using gates is shown in Fig. 4.4.

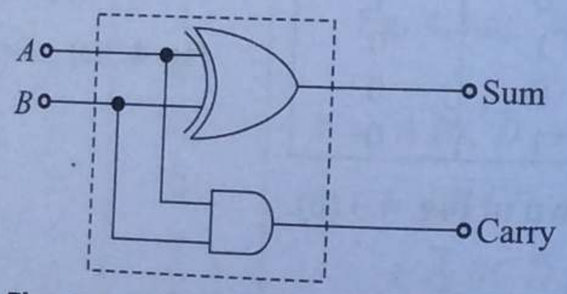


Fig. 4.4 Logic diagram of half-adder

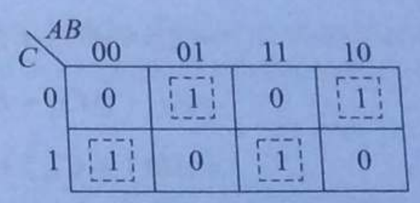
4.3.2 Full-Adder

A half-adder has two 1-bit inputs and there is no provision to add a carry which could have been generated from lower bit order additions. This limitation of half-adder is overcome in full-adder. The full-adder is a combinational logic circuit that has the provision to add a carry.

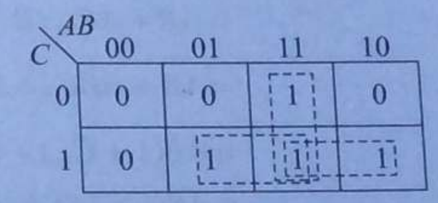
Let us consider A and B as two 1-bit inputs. C_{in} is a carry generated from the previous order bit addition; S (sum) and C_{out} (carry) are the outputs of the full-adder. The truth table of full-adder is given in Table 4.4.

Table 4.4 Truth table of full-adder

Inputs			*Outputs*	
A	*B*	C_{in}	*Sum*	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



K-map for sum *S*



K-map for C_{out}

$$S = \overline{A}\,\overline{B}C_{in} + \overline{A}\,B\overline{C}_{in} + ABC_{in} + A\overline{B}\,\overline{C}_{in}$$

$$= C_{in}(\overline{A}\,\overline{B} + AB) + \overline{C}_{in}(\overline{A}\,B + A\overline{B})$$

$$= C_{in}(A \odot B) + \overline{C}_{in}(A \oplus B)$$

$$= C_{in}(\overline{A \oplus B}) + \overline{C}_{in}(A \oplus B)$$

$$= C_{in} \oplus A \oplus B$$

$$C_{out} = AB + AC_{in} + BC_{in}$$

The logic diagram of full-adder using gates is shown in Fig. 4.5.

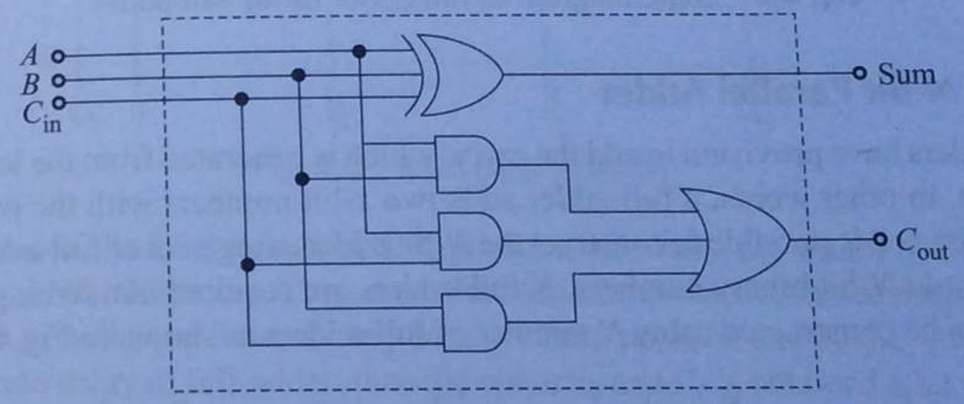


Fig. 4.5 Logic diagram of full-adder

Full-adder using half-adder

A full-adder can be implemented using two half-adders and an OR gate. The Boolean expressions of half-adder are:

$$\left.\begin{aligned} S &= A \oplus B \\ \text{and} \quad C &= AB \end{aligned}\right\} \tag{4.3}$$

The Boolean expressions of full-adder are:

$$\left.\begin{aligned} S &= C_{in} \oplus A \oplus B \\ \text{and } C_{out} &= AB + AC_{in} + BC_{in} \end{aligned}\right\} \tag{4.4}$$

$$S = C_{in} \oplus S$$

$$C_{out} = AB + AC_{in} + BC_{in}$$

$$= AB + AC_{in} + BC_{in}(A + \overline{A}) \qquad (\because\ A + \overline{A} = 1)$$
$$= AB + AC_{in} + BC_{in}A + BC_{in}\overline{A}$$
$$= AB(1 + C_{in}) + AC_{in} + \overline{A}\,BC_{in}$$
$$= AB + AC_{in} + \overline{A}\,BC_{in} \qquad (\because\ 1 + C_{in} = 1)$$
$$= AB + AC_{in}(B + \overline{B}) + \overline{A}\,BC_{in} \qquad (\because\ B + \overline{B} = 1)$$
$$= AB + ABC_{in} + A\overline{B}\,C_{in} + \overline{A}\,BC_{in}$$
$$= AB(1 + C_{in}) + C_{in}(A\overline{B} + \overline{A}\,B) \qquad (\because\ 1 + C_{in} = 1)$$
$$= AB + C_{in}(A \oplus B)$$
$$= AB + C_{in}S \qquad (\because\ A \oplus B = S)$$

The logic diagram of full-adder using half-adder is shown in Fig. 4.6.

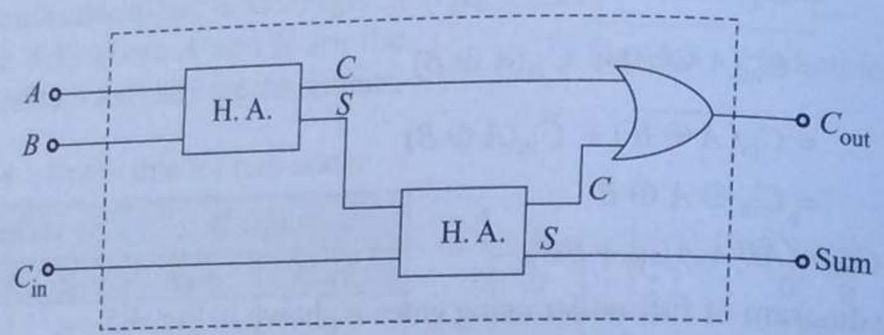


Fig. 4.6 Logic diagram of full-adder using half-adder

4.3.3 *N*-Bit Parallel Adder

Full-adders have provision to add the carry, which is generated from the lower bit addition. In other words, a full-adder adds two 1-bit numbers with the previous carry. Hence it is possible to construct the *N*-bit adder using a set of full-adders. In order to add *N*-bit binary numbers, *N* full-adders are required. An *N*-bit parallel adder can be constructed using *N* number of full-adders as shown in Fig.4.7.

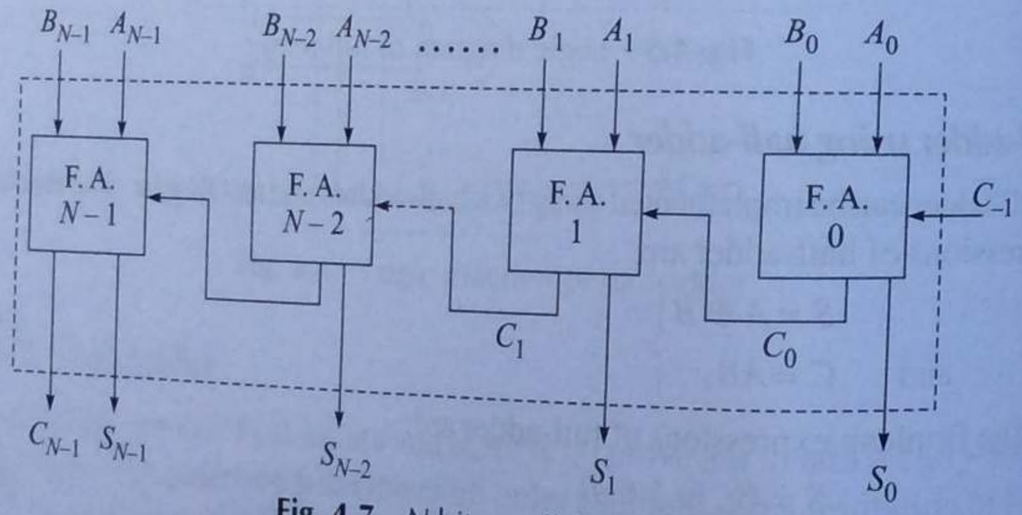


Fig. 4.7 *N*-bit parallel adder

The *N* full-adders are connected in parallel and hence, it is also known as *parallel adder*. The previous carry for adder 0 is set to zero. The carry output of each adder

is connected to the carry input of the next higher order adder. Hence, it is also known as *carry-propagate adder*. The block diagram of *N*-bit adder is shown in Fig. 4.8.

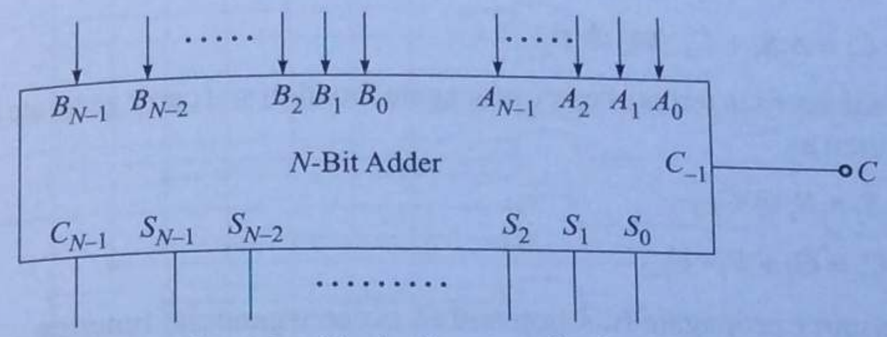


Fig. 4.8 Block diagram of *N*-bit adder

Parallel adders suffer from propagation delay problem, as higher bit additions depend on the carry generated from lower bit addition. It is observed that parallel adders are not often used in digital systems because of their high propagation delay time, which is proportional to the number of bits to be added. Let us consider a 4-bit parallel adder as shown in Fig. 4.9.

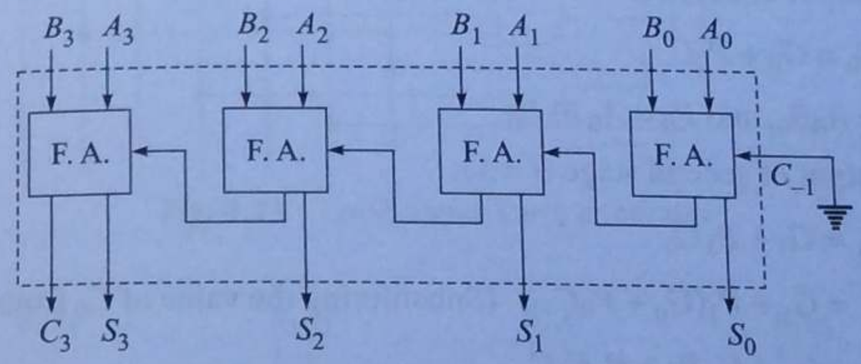


Fig. 4.9 4-bit parallel adder

Addition of the LSB bits A_0 and B_0 produces the carry C_0, which is the input carry for the next full-adder. This carry when added to the bits of the second position (i.e. $C_0 + A_1 + B_1$) produces a carry into the third position and so on. If t_p is the propagation delay of full-adder, then the result S_0 and C_0 is obtained after a delay of t_p, S_1 and C_1 after a delay of $2t_p$, S_2 and C_2 after a delay of $3t_p$, S_3 and C_3 after a delay of $4t_p$ and so on. Hence for an *N*-bit parallel adder, the total delay time is equal to Nt_p.

4.3.4 Carry Look-Ahead Adder

The parallel adder has a larger propagation delay and hence low speed of operation. This limitation is overcome in carry look ahead adder. Consider the circuit of full-adder as shown in Fig.4.10.

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + C_{in}(A \oplus B)$$

where *A* and *B* are the present inputs, C_{in} is the previous carry, *S* is the sum at output, C_{out} is the present carry at output or next input carry.

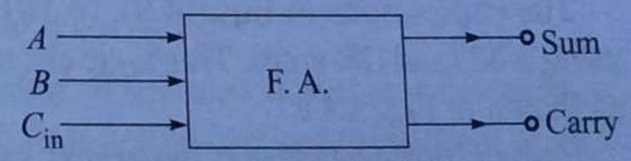


Fig. 4.10 Full-adder

For $(i+1)$th stage of full-adder, we can write

$$S_i = A_i \oplus B_i \oplus C_{i-1} \tag{4.5}$$

$$C_i = A_iB_i + C_{i-1}(A_i \oplus B_i) \tag{4.6}$$

These equations in terms of carry propagate function and carry generate function can be written as

$$S_i = P_i \oplus C_{i-1} \tag{4.7}$$

$$C_i = G_i + P_i \cdot C_{i-1} \tag{4.8}$$

where P_i is carry propagate function and G_i is carry generate function.

From Eqs. (4.5) and (4.7),

$$P_i = A_i \oplus B_i$$

From Eqs. (4.6) and (4.8),

$$G_i = A_i \cdot B_i$$

Carry output of first full-adder $(i = 0)$,

$$C_0 = G_0 + P_0C_{-1} \tag{4.9}$$

where $G_0 = A_0B_0$, and $P_0 = A_0 \oplus B_0$

Carry output of second stage $(i = 1)$,

$$C_1 = G_1 + P_1C_0$$

$$= G_1 + P_1(G_0 + P_0C_{-1}) \quad \text{(Substituting the value of } C_0 \text{ from Eq. (4.9))}$$

$$= G_1 + G_0P_1 + P_0P_1C_{-1} \tag{4.10}$$

Carry output of third stage $(i = 2)$,

$$C_2 = G_2 + P_2C_1$$

$$= G_2 + P_2(G_1 + G_0P_1 + P_0P_1C_{-1})$$

(Substituting the value of C_1 from Eq. (4.10))

$$= G_2 + G_1P_2 + G_0P_1P_2 + P_0P_1P_2C_{-1} \tag{4.11}$$

Carry output of fourth stage $(i = 3)$,

$$C_3 = G_3 + P_3C_2$$

$$= G_3 + P_3(G_2 + G_1P_2 + G_0P_1P_2 + P_0P_1P_2C_{-1})$$

(Substituting the value of C_1 from Eq. (4.15))

$$= G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3 + P_0P_1P_2P_3C_{-1} \tag{4.12}$$

The expressions in Eqs. (4.9), (4.10), (4.11), and (4.12) can be implemented using AND and OR gates. This logic circuit is known as look-ahead carry generator as shown in Fig. 4.11.

Using a look-ahead carry generator, we can construct the 4-bit adder and reduce the propagation delay time. The logic diagram of 4-bit parallel adder with look-ahead carry generator is shown in Fig. 4.12.

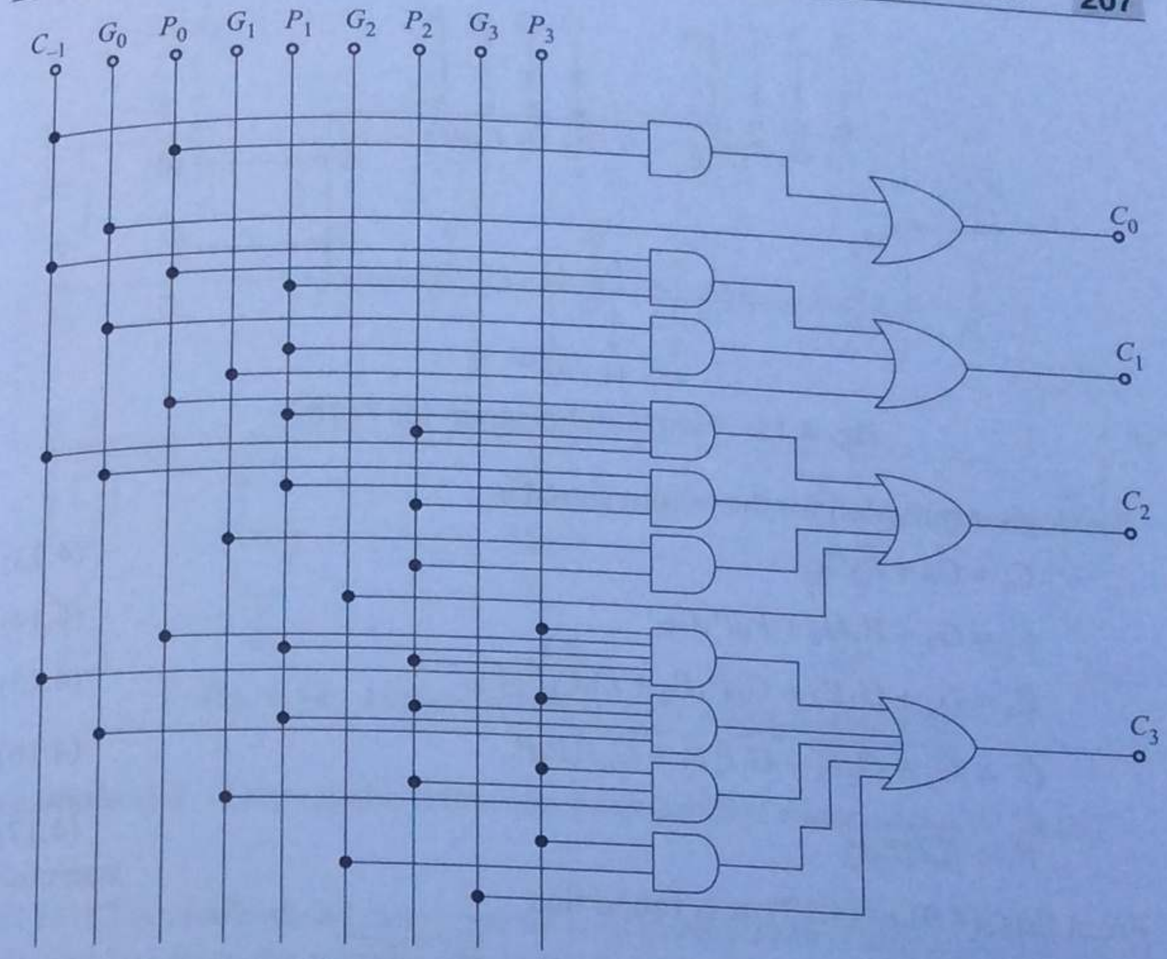


Fig. 4.11 Look-ahead carry generator

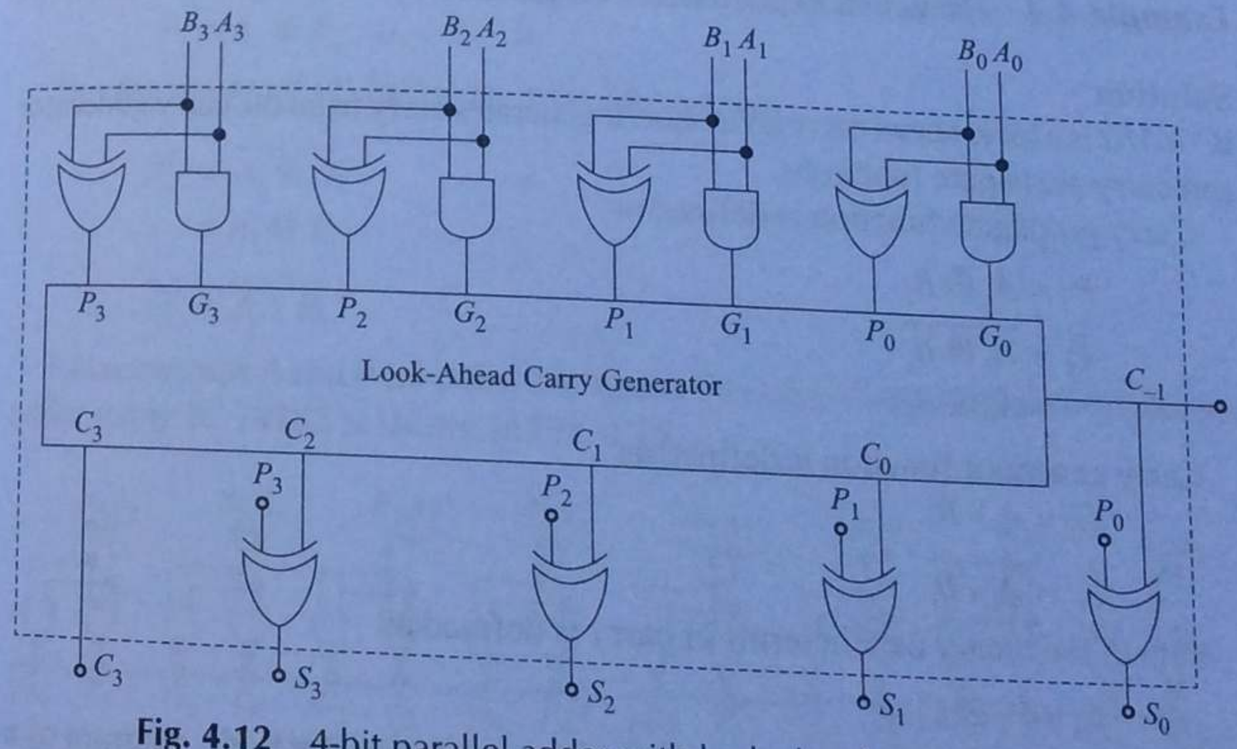


Fig. 4.12 4-bit parallel adder with look-ahead carry generator

Here, $S_i = P_i \oplus C_{i-1}$, $P_i = A_i \oplus B_i$, and $G_i = A_i \cdot B_i$

IC 74182 is a look-ahead carry generator. It has nine inputs and five outputs, four active low propagate input signals, four active low generate input signals, active high carry input, three active high carry output, active high carry propagate, and carry generate outputs as shown in Fig. 4.13.

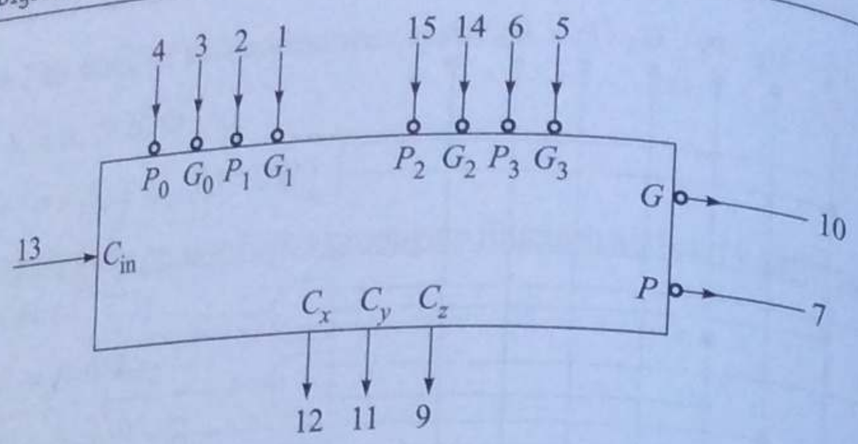


Fig. 4.13 Functional diagram for 74182

The logic expression for the output signal is

$$C_x = G_0 + P_0 C_{in} \tag{4.13}$$

$$C_y = G_1 + P_1 G_0 + P_0 P_1 C_{in} \tag{4.14}$$

$$C_z = G_2 + G_1 P_2 + G_0 P_1 P_2 + P_0 P_1 P_2 C_{in} \tag{4.15}$$

$$\overline{G} = \overline{G_3 + G_2 P_3 + G_1 P_1 P_3 + G_0 P_1 P_2 P_2} \tag{4.16}$$

$$\overline{P} = \overline{P_0 P_1 P_2 P_3} \tag{4.17}$$

From Eqs. (4.9) to (4.17), it is found that

$$C_0 = C_x;\quad C_1 = C_y;\quad C_2 = C_y;\quad C_3 = \overline{\overline{G} + \overline{P} G_{in}}$$

Example 4.4 Draw and explain the 4-bit parallel adder using IC 74182.

Solution

IC 74182 is a look-ahead carry generator. It generates carry from the carry generator and carry propagate functions.

Carry propagate function is defined as

$$P_1 = A_i \oplus B_i$$

$$\overline{P_1} = \overline{A_i \oplus B_i}$$

$$= A_i \odot B_i$$

Carry generator function is defined as

$$G_i = A_i \cdot B_i$$

$$\overline{G_i} = \overline{A_i \cdot B_i}$$

Sum of the binary data in terms of carry is defined as

$$S_i = P_i \oplus C_{i-1}$$

Let us consider A and B as two 4-bit binary numbers. The logic diagram of a 4-bit adder using 74182 is shown in Fig. 4.14.

C_x, C_y, C_z output carries of 74182 are equivalent to carry C_1, C_2, and C_3 respectively and $C_3 = \overline{\overline{G} + \overline{P} C_{in}}$.

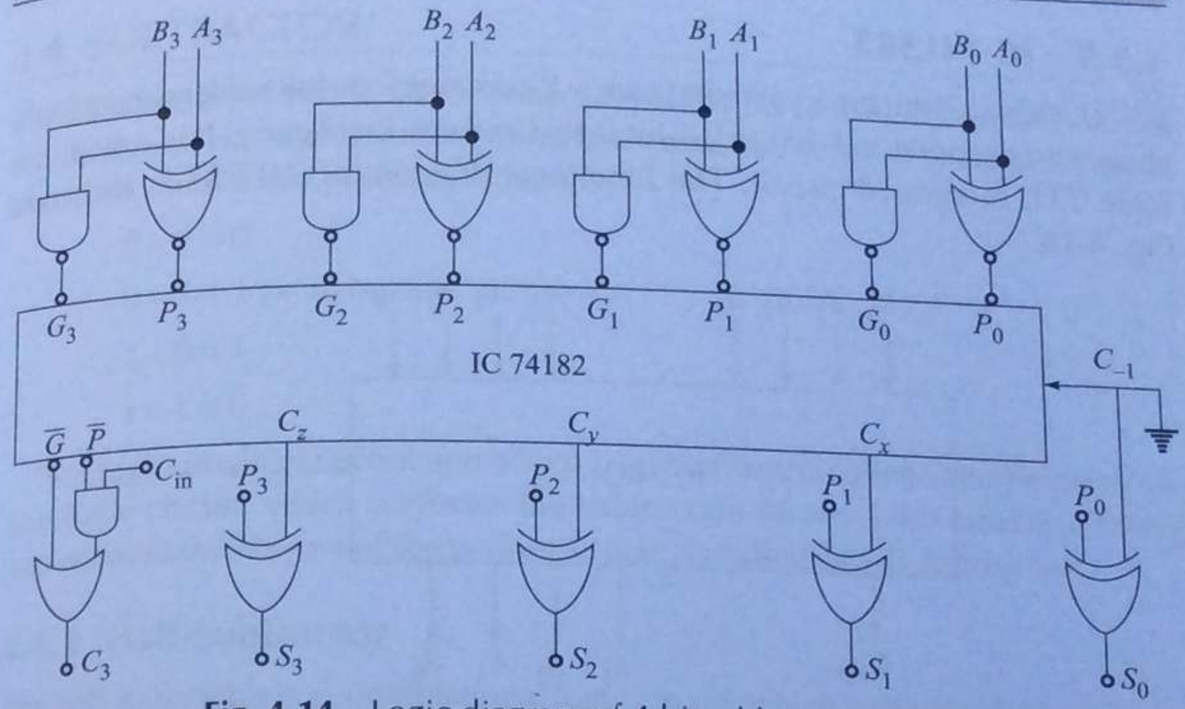


Fig. 4.14 Logic diagram of 4-bit adder using 74182

Example 4.5 Draw and explain the 8-bit parallel adder using IC 74182.

Solution

IC 74182 is a look-ahead carry generator. It generates the carry C_x, C_y, and C_z(C_1, C_2, and C_3) from the carry generator and carry propagate functions. It has four generators and four propagate inputs and hence, to implement 8-bit parallel adder, two IC are needed. The carry propagate and carry generate functions are defined as

$$P_i = A_i \oplus B_i\,,\quad G_i = A_i \cdot B_i$$

But the inputs of IC 74182 are active low, hence,

$$\overline{P_i} = \overline{A_i \oplus B_i}$$

$$= A_i \odot B_i$$

$$\overline{G_i} = \overline{A_i \cdot B_i}$$

Let us assume A and B are two 8-bit inputs. The logic diagram of an 8-bit parallel adder using IC 74182 is shown in Fig. 4.15.

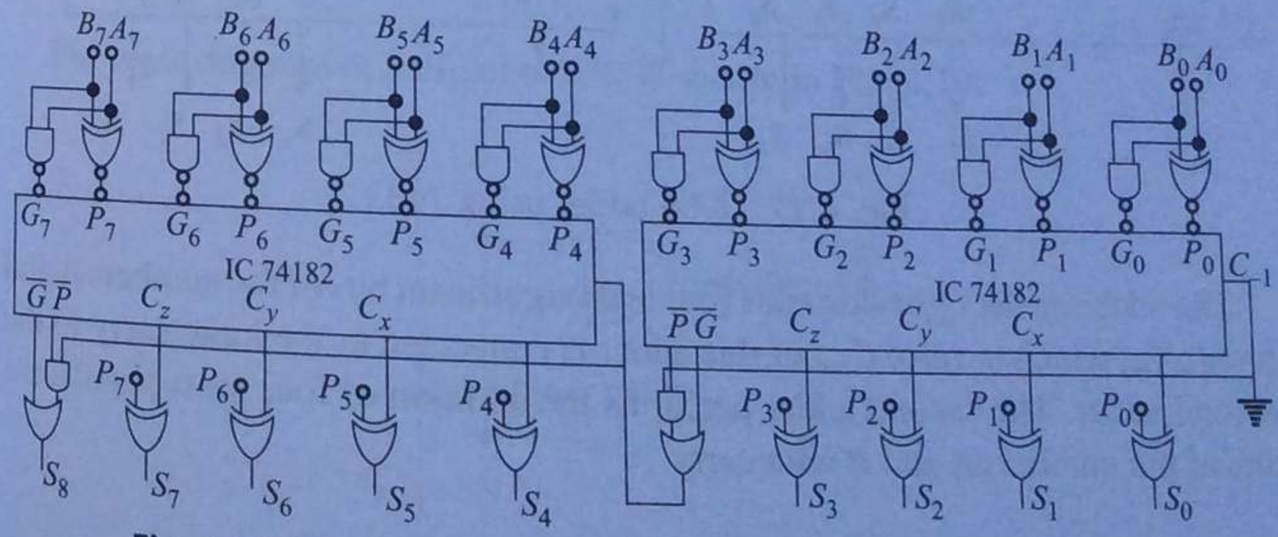


Fig. 4.15 Logic diagram of 8-bit parallel adder using IC 74182

4.3.5 IC 74LS83

IC 74LS83 is a 4-bit binary parallel adder. It contains four full-adders and a look-ahead carry generator. It has two 4-bit inputs and one input carry. It is a medium-scale TTL integrated circuit. The functional diagram of 74LS83 is shown in Fig. 4.16.

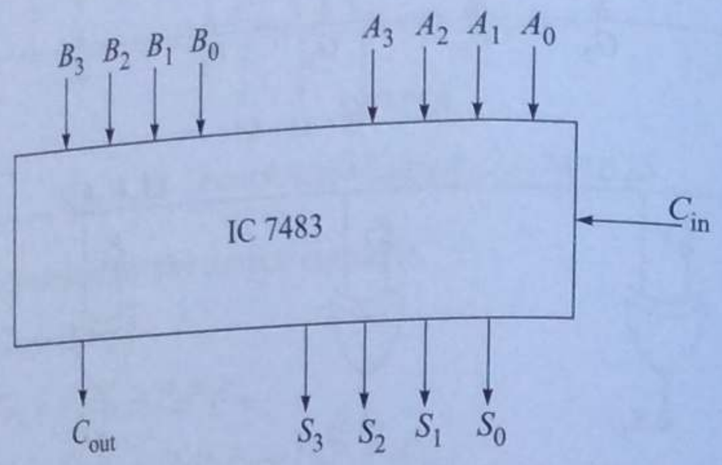


Fig. 4.16 Functional diagram for 74LS83

In Fig. 4.16, A_0 to A_3 and B_0 to B_3 are the two 4-bit inputs. C_{in} is carry input, C_{out} is carry output, and S_0 to S_3 are 4-bit outputs which represent the sum of inputs. Because of carry input and output, it is possible to implement the addition of numbers greater than 4-bit using IC 7483.

***Example* 4.6** Implement an 8-bit binary adder using 4-bit adder 7483 or 74283.

Solution

IC 7483 or IC74283 is a 4-bit parallel adder. To add two 8-bit numbers, we need to use two ICs of 4-bit adder. Figure 4.17 shows the logic diagram of an 8-bit adder using 7483.

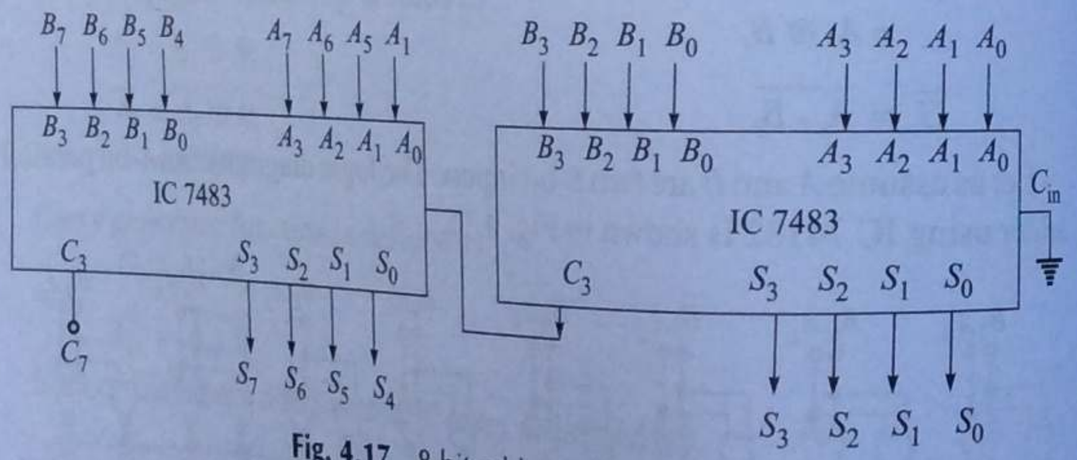


Fig. 4.17 8-bit adder using 7483

The adder on the right side adds four least significant bits of the numbers A and B with C_{in} which is zero. C_{out} of this adder is connected as the input carry to the second adder. The second adder performs the addition of four most significant bits of the numbers A and B with carry.

4.4 SUBTRACTOR

The subtraction of two binary numbers is performed in a similar manner as the subtraction of decimal numbers. Only four cases can occur in the subtraction of two binary bits in any position, those are:

$0 - 0 = 0$

$0 - 1 = 1$ (with borrow 1)

$1 - 0 = 1$

$1 - 1 = 0$

The logic circuit of subtraction of two 1-bit numbers is called as *half-subtractor*. The logic circuit, which performs the subtraction of two 1-bit numbers, taking into account the borrow of the pervious stage, is called as *full-subtractor*.

4.4.1 Half-Subtractor

The half-subtractor is a combinational logic circuit, which performs the subtraction of two 1-bit numbers. It subtracts B (subtrahend) from A (minuend). It has two 1-bit inputs and two outputs as difference and borrow as shown in Fig. 4.18.

The truth table of half-subtractor is given in Table 4.5, where A, B are the inputs and difference (D) and borrow (C) are the outputs.

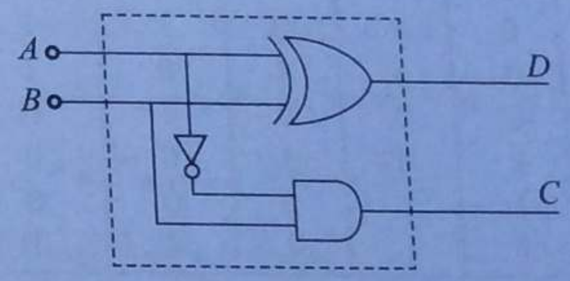
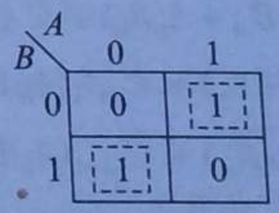


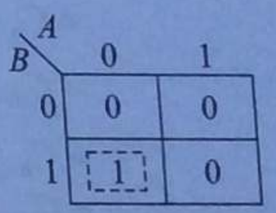
Fig. 4.18 Block diagram of half-subtractor

Table 4.5 Truth table of half-subtractor

Inputs		*Outputs*	
A	*B*	*D*	*C*
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



Difference (D)



Borrow (C)

$$D = A\overline{B} + \overline{A}B = (A \oplus B) \tag{4.18}$$

$$C = \overline{A}B \tag{4.19}$$

The logic diagram of half-subtractor is shown in Fig. 4.19.

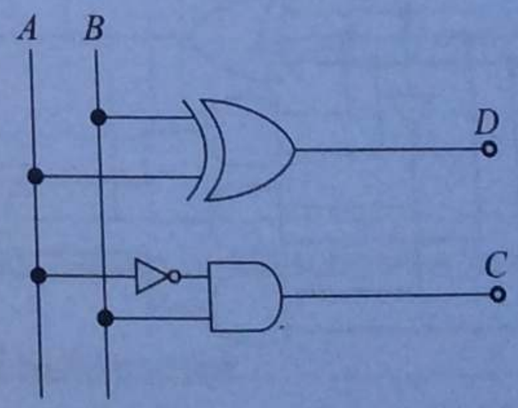


Fig. 4.19 Logic diagram of half-subtractor

A half-subtractor has only two 1-bit inputs and there is no provision for subtraction of borrows which may be generated from lower order bit subtraction. This limitation of half-subtractor is overcome in full-subtractor. The full-subtractor is a combinational logic circuit that has provision to take into account a borrow.

Let us consider A_n is 1-bit minuend, B_n is 1-bit subtrahend, C_{n-1} is borrow from the previous stage, D_n and C_n are the difference and borrow of subtractor. Figure 4.20 shows the block diagram of full-subtractor.

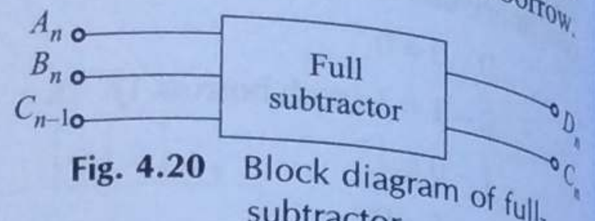
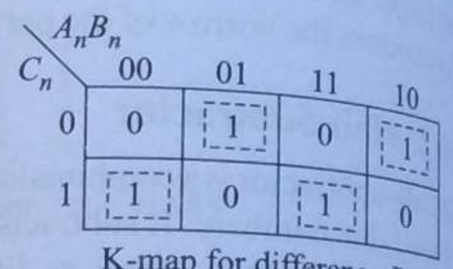


Fig. 4.20 Block diagram of full-subtractor

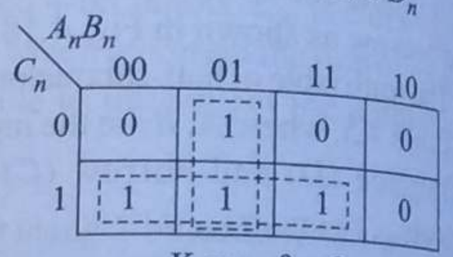
The truth table of full-subtractor is given in Table 4.6.

Table 4.6 Truth table of full-subtractor

Inputs			*Outputs*	
A_n	B_n	C_{n-1}	D_n	C_n
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



K-map for difference D_n



K-map for C_n

$$D_n = \overline{A}_n \overline{B}_n C_{n-1} + A_n B_n C_{n-1} + \overline{A}_n B_n \overline{C}_{n-1} + A_n \overline{B}_n \overline{C}_{n-1}$$

$$= C_{n-1}(\overline{A}\,\overline{B}_n + A_n B_n) + \overline{C}_{n-1}(\overline{A}_n B_n + A_n \overline{B}_n)$$

$$= C_{n-1}(A_n \cdot B_n) + \overline{C}_{n-1}(A_n \oplus B_n)$$

$$= C_{n-1}(\overline{A_n \cdot B_n}) + \overline{C}_{n-1}(A_n \oplus B_n)$$

$$= C_{n-1} \oplus A_n \oplus B_n \quad (4.20)$$

$$C_n = \overline{A}_n B_n + \overline{A}_n C_{n-1} + B_n C_n$$

$$= B_n C_n + \overline{A}_n B_n + \overline{A}_n C_{n-1} \quad (4.21)$$

The logic diagram of full-subtractor is shown in Fig. 4.21.

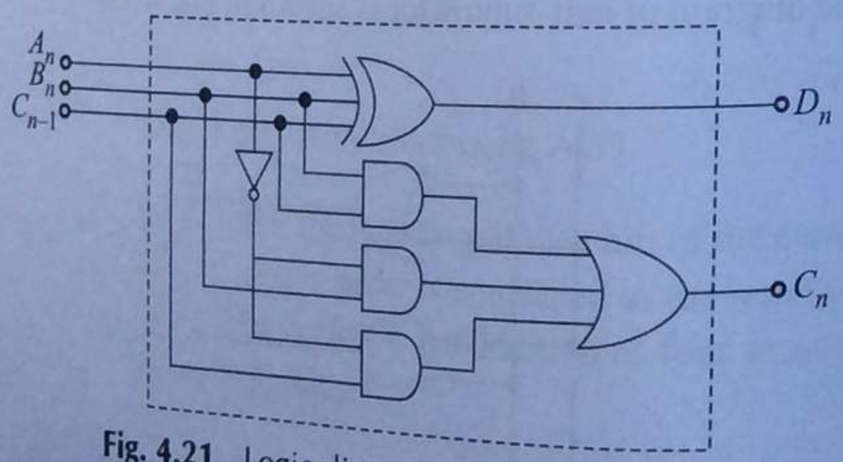


Fig. 4.21 Logic diagram of full-subtractor

Full-subtractor using half-subtractor

The full-subtractor can be implemented using two half-subtractors and an OR gate as explained below. We have seen that, for a half-subtractor,

$$D_{nH} = A_n \oplus B_n \quad (4.22)$$

$$C_{nH} = \overline{A}_n \cdot B_n \quad (4.23)$$

The Boolean expressions of full-subtractor are

$$\text{Difference} = A_n \oplus B_n \oplus C_{n-1}$$

$$\text{Borrow} = B_n C_{n-1} + \overline{A}_n C_{n-1} + \overline{A}_n B_n$$

$$\text{Difference} = D_{nH} \oplus C_{n-1} \quad (\because D_{nH} = A_n \oplus B_n) \quad (4.24)$$

$$\text{Borrow} = \overline{A}_n B_n + \overline{A}_n C_{n-1} + B_n C_{n-1} \quad (B_n + \overline{B}_n = 1)$$

$$= \overline{A}_n B_n + \overline{A}_n C_{n-1}(B_n + \overline{B}_n) + B_n C_{n-1}$$

$$= \overline{A}_n B_n + \overline{A}_n C_{n-1} B_n + \overline{A}_n C_{n-1} \overline{B}_n + B_n C_{n-1}$$

$$= \overline{A}_n B_n [1 + (C_{n-1})] + \overline{A}_n \overline{B}_n C_{n-1} + B_n C_{n-1}$$

$$= \overline{A}_n B_n + \overline{A}_n \overline{B}_n C_{n-1} + B_n C_{n-1} \quad (\because 1 + C_{n-1} = 1)$$

$$= \overline{A}_n B_n + \overline{A}_n \overline{B}_n C_{n-1} + B_n C_{n-1}(A_n + \overline{A}_n)(\because A_n + \overline{A}_n = 1)$$

$$= \overline{A}_n B_n + \overline{A}_n \overline{B}_n C_{n-1} + B_n C_{n-1} A_n + B_n C_{n-1} \overline{A}_n$$

$$= \overline{A}_n B_n (1 + C_{n-1}) + C_{n-1}(\overline{A}_n \overline{B}_n + B_n A_n)$$

$$= \overline{A}_n B_n + C_{n-1}(A_n \cdot B_n)$$

$$= \overline{A}_n B_n + C_{n-1}(\overline{A_n \oplus B_n})$$

Substituting C_{nH} and D_{nH} from equations 4.22 and 4.23,

$$\text{Borrow} = C_{nH} + C_{n-1} \overline{D}_{nH}$$

The logic diagram of full-subtractor using two half-subtractors is shown in Fig. 4.22.

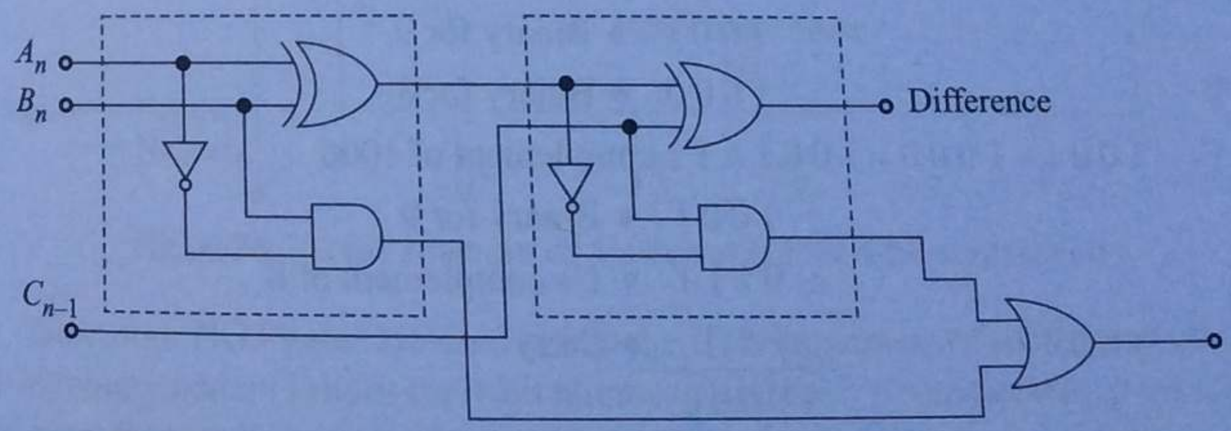


Fig. 4.22 Logic diagram of full-subtractor using half-subtractor

4.4.3 *N*-Bit Parallel Subtractor

The full-subtractor has provision to subtract the borrow which is generated from the lower bit subtraction. In other words, a full-subtractor subtracts 1-bit subtrahend

and the previous borrow from the 1-bit minuend. Hence it is possible to construct N-bit subtractors using full-subtractors. In order to subtract a binary number with more than one bit, more than one full-subtractors are needed. The N-bit parallel subtractor can be constructed using N number of full-subtractors as shown in Fig. 4.23.

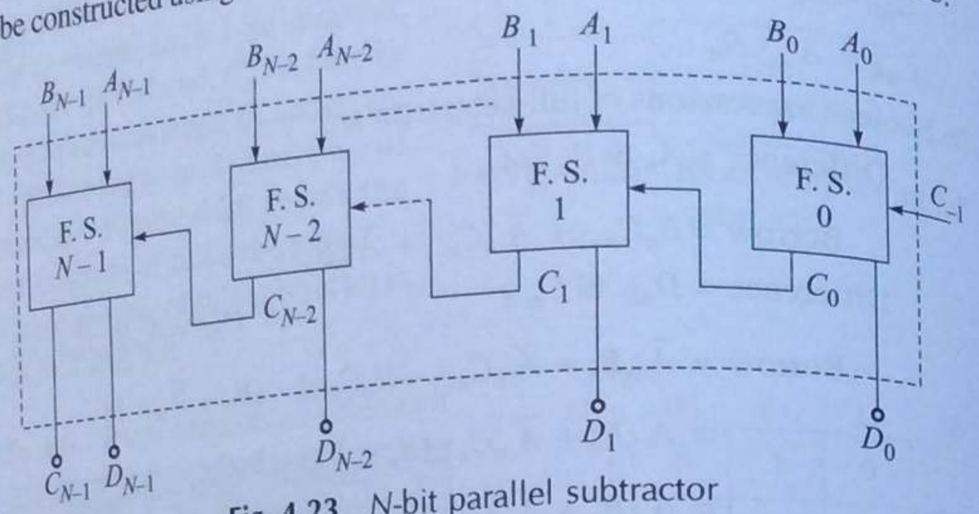


Fig. 4.23 N-bit parallel subtractor

The borrow output of each subtractor is connected to the borrow input of the next highest order subtractor. The previous borrow for subtractor 0 is set to zero.

4.4.4 Four-Bit Subtractor Using Adder

In most of the digital systems, the process of subtraction is performed by using the process of addition. Subtraction of B from A is equivalent to the addition of complement of B to A. In binary subtraction, 1's complement or 2's complement method is used.

1's complement subtraction

The subtraction of B from A is equivalent to the addition of the 1's complement of B to A, i.e. $A - B = A +$ 1's complement of B.

The 1's complement of a binary number is obtained by subtracting each bit from 1 or complementing each bit of the number.

To understand this process, consider the following examples.

(a) 9 − 8 = 1

```
    1001   ♦ Binary for 9
    1000   ♦ Binary for 8
1001−1000 = 1001 + 1's complement of 1000
    1001   ♦ Binary for 9
  + 0111   ♦ 1's complement of 8
  ------
    111    ♦ Carry
  ------
   10000
     └──→ 1  Add end-around carry
  ------
    0001
  ------
```


(b) 8 − 9 = −1

```
    1000   ♦ Binary for 8
    1001   ♦ Binary for 9
1000−1001 = 1000 + 1's complement of 1001
    1000   ♦ Binary for 8
  + 0110   ♦ 1's complement of 9
  ------
    1110   ♦ 1's complement of 1
  ------
```

Carry is zero, the result is negative and it is in 1's complement form.

From the above examples, it is observed that, after the addition, if carry is 1, then the result is positive, and correct result is obtained by adding the end-around carry. If carry is 0, then the result is negative and it is in 1's complement form.

Algorithm of binary subtraction using 1's complement

1. Find the 1's complement of subtrahend.
2. Add the two numbers using binary addition.
3. Check the carry. If carry is generated, the correct result is obtained by adding the carry to the result, otherwise find the 1's complement of the result.

The logic diagram of 1's complement 4-bit subtraction is shown in Fig. 4.24.

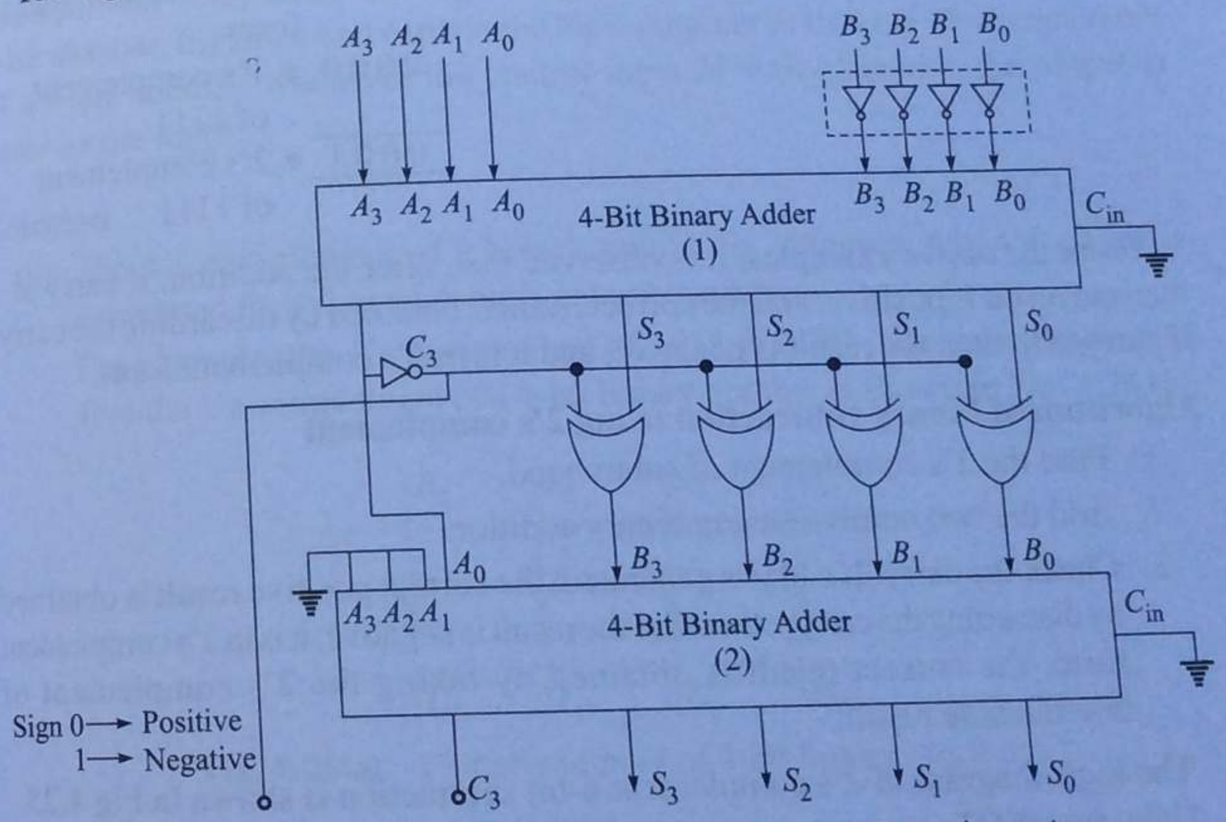


Fig. 4.24 Logic diagram of 1's complement 4-bit subtraction

Here, four NOT gates are used to find the 1's complement of subtrahend (B). A 4-bit binary adder (1) adds the 4-bit number A and the 1's complement of the 4-bit number B with the initial carry zero.

Four Ex-OR gates are used to find the 1's complement of the result, when carry is not generated. The second 4-bit binary adder (2) is used to add the end-around carry, when carry is generated.

2's complement subtraction

The subtraction of B from A is equivalent to the addition of 2's complement of B to A, i.e. $A - B = A +$ 2's complement of B.

The 2's complement of B is obtained by adding 1 to the 1's complement of B.

To understand the process, consider the following examples.

(a) 9 − 8 = 1

```
  1001   ♦ Binary for 9
  1000   ♦ Binary for 8
  0111   ♦ 1's complement of 8
    +1
  ----
  1000   ♦ 2's complement of 8
```

1001 − 1000 = 1001 + 2's complement of 8

```
  1001
 +1000
 -----
 10001   Discard the carry, and get true positive result.
```

(b) 8 − 9 = −1

```
  1000   ♦ Binary for 8
  1001   ♦ Binary for 9
  0110   ♦ 1's complement of 9
    +1
  ----
  0111   ♦ 2's complement of 9
```

1000 − 1001 = 1000 + 2's complement of 8

```
  1000
 +0111
 -----
 01111   Carry is zero. So the result is negative and it is in 2's complement form.
  0000   ♦ 1's complement of 1111
 -----
  0001   ♦ 2's complement of 1111
```

From the above examples, it is observed that, after the addition, if carry is 1 then the result is positive; and the correct result is obtained by discarding the carry. If carry is 0, then the result is negative, and it is in 2's complement form.

Algorithm of binary subtraction using 2's complement

1. Find the 2's complement of subtrahend.
2. Add the two numbers using binary addition.
3. Check the carry. If carry is generated, the correct positive result is obtained by discarding the carry; otherwise the result is negative, it is in 2's complement form, the correct result is obtained by taking the 2's complement of intermediate result.

The logic diagram of 2's complement 4-bit subtraction is shown in Fig 4.25.

Here, four NOT gates are used to find the 1's complement of subtrahend (B). C_{in} is connected to V_{CC} to add 1 to the 1's complement to get the 2's complement of subtrahend (B).

The 4-bit binary adder (1) adds the 4-bit number A and the 2's complement of B(1's complement of $B + 1$). Four Ex-OR gates are used to find the 1's complement of the result, whenever carry is not generated.

The second 4-bit binary adder (2) adds the result of adder 1 and the output of Ex-OR gates with carry as 1, whenever carry is not generated, and gives the correct result.

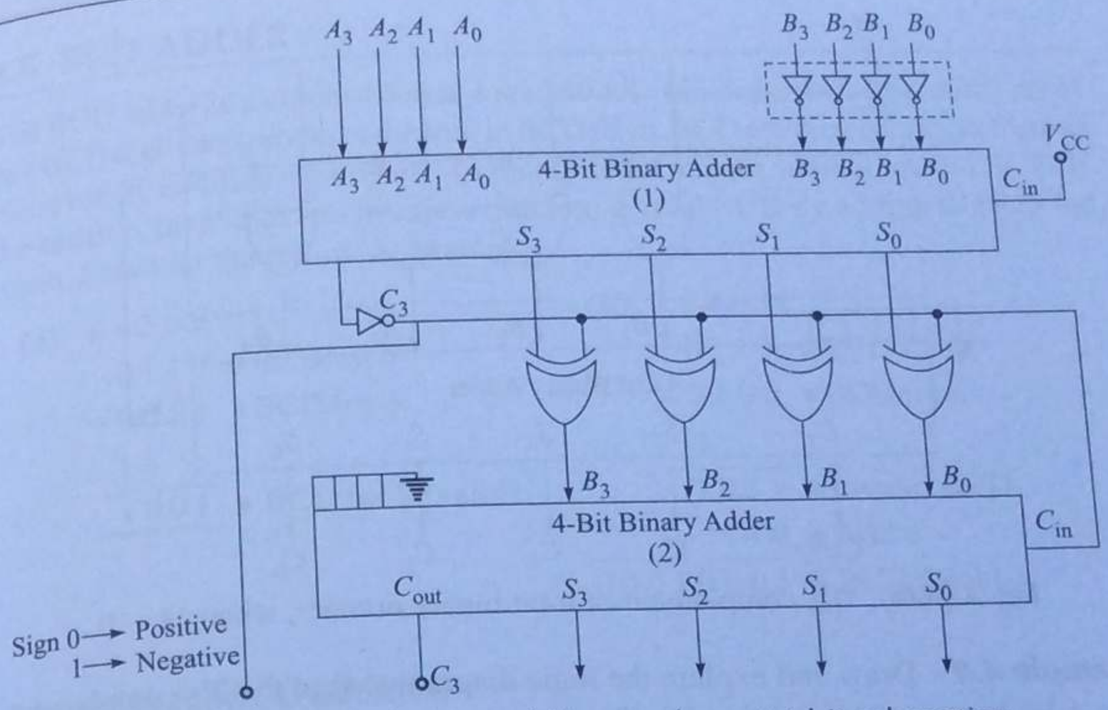


Fig. 4.25 Logic diagram of 2's complement 4-bit subtraction

Example 4.7 (a) Draw the logic diagram to find the 1's complement of a 4-bit number. (b) Draw and explain the logic diagram to find the 1's complement of a 4-bit number, whenever the control input $M = 0$, otherwise the output is same as the input.

Solution

(a) The 1's complement of a binary number is obtained, when each bit is subtracted from 1 or each bit is complemented.

For example, the 1's complement of 1011 is 0100. The logic diagram to find the 1's complement of a 4-bit binary number is shown in Fig. 4.26 (a).

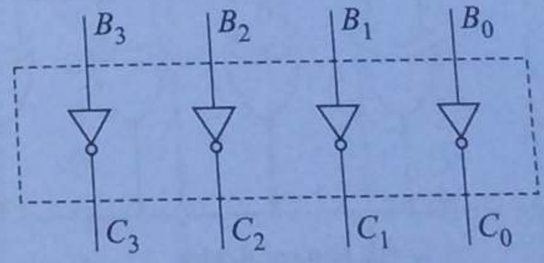


Fig. 4.26(a) 1's complement of 4-bit binary number

(b) When one input of Ex-OR gate is logic 1, then the output of Ex-OR gate is the complement of anther input; otherwise the output of Ex-OR gate is same as the inputs.

The logic diagram to find the 1's complement of a 4-bit binary number when $M = 0$, otherwise same as inputs, is shown in Fig. 4.26 (b).

C is the 1's complement of B, when $M = 0$

C is same as B, when $M = 1$

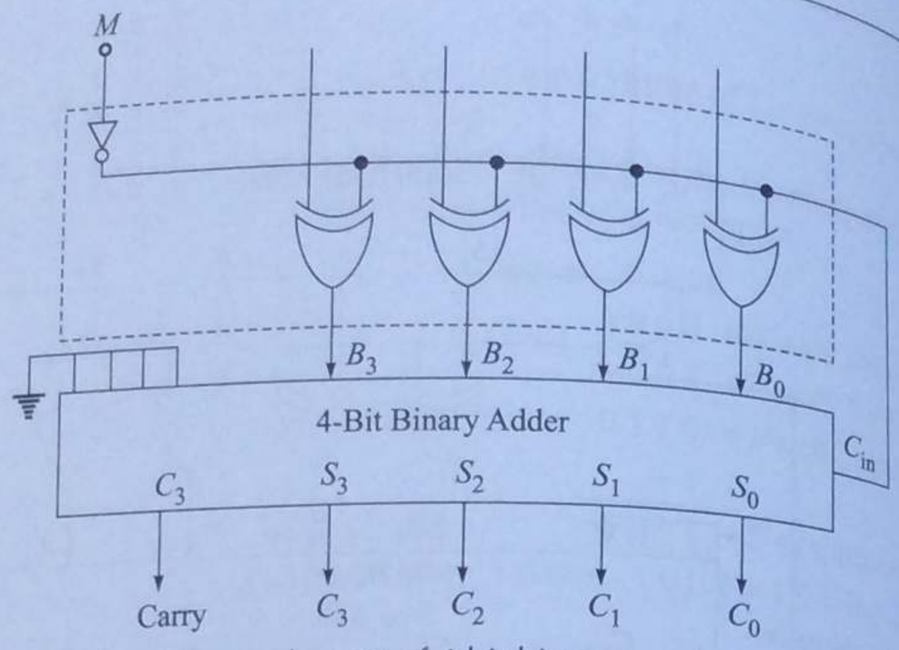


Fig. 4.26(b) 1's complement of 4-bit binary number, when $M = 0$

Example 4.8 Draw and explain the logic diagram to find the 2's complement of a 4-bit binary number when $M = 0$, otherwise the output is same as the input.

Solution

The 2's complement of a number is obtained by adding 1 to the 1's complement of a number. The 1's complement of a number with control input M is shown in Fig. 4.26 (b). To add 1 to the 1's complement, a 4-bit binary adder is used. The logic diagram of 2's complement of a 4-bit binary inputs when $M = 0$, otherwise output is same as inputs, is shown in Fig. 4.27.

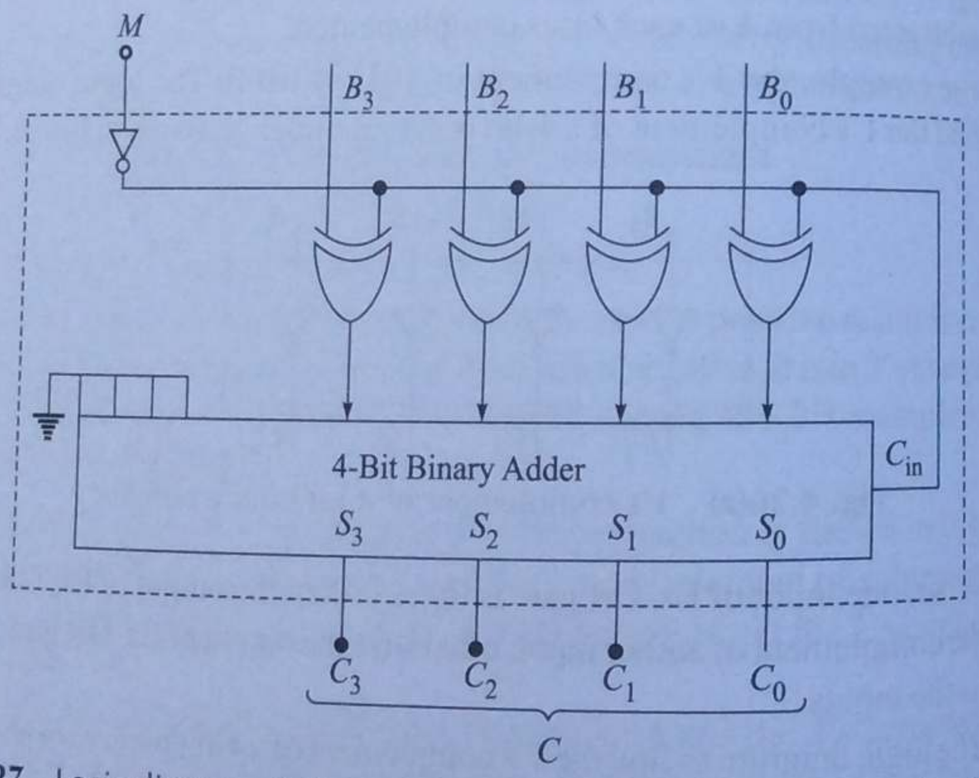


Fig. 4.27 Logic diagram of 2's complement of 4-bit binary inputs when $M = 0$

C is the 2's complement of input B, when $M = 0$
C is same as input B, when $M = 1$

4.5 BCD ADDER

The BCD adder is a combinational logic circuit, which performs the addition of two BCD digits and produces the sum in BCD form. BCD addition can be performed using binary addition. The result of a binary adder may be valid or invalid BCD. If the result is invalid, it can be converted into a valid BCD by adding 0110 to the result. Consider the following examples.

(a) 6 + 3 = 9

```
  0 1 1 0   ♦BCD for 6
+ 0 0 1 1   ♦BCD for 3
  1 1
-------
  1 0 0 1   ♦ BCD for 9 (valid)
```

(b) 7 + 4 = 11

```
    0 1 1 1   ♦BCD for 7
  + 0 1 0 0   ♦BCD for 4
      1
  -------
    1 0 1 1   ♦Invalid BCD
  + 1 1 0     ♦Add 6
  -----------
  0 0 0 1 0 0 0 1   ♦BCD for 11
```

Algorithm for BCD addition

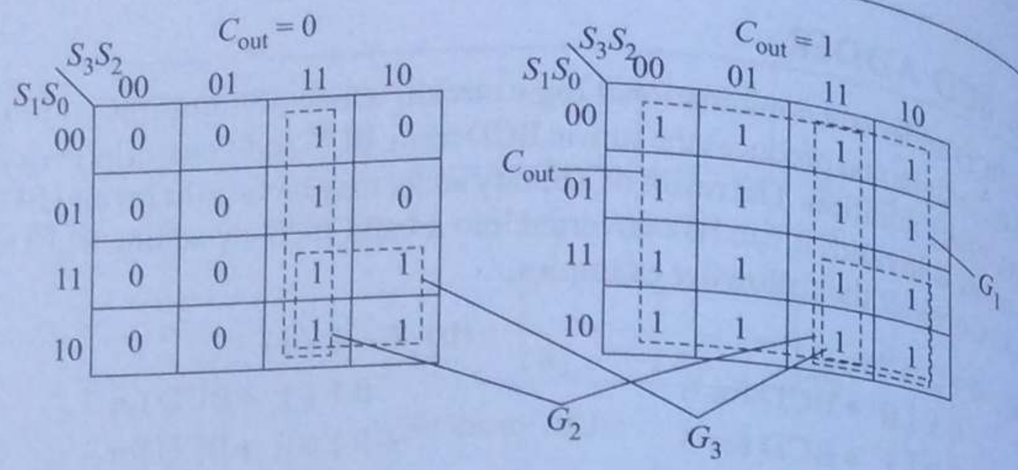
1. Add two BCD numbers using ordinary binary addition.
2. Check whether the result is valid or invalid BCD.
   - If the result is valid (i.e. result ≤ 9), no correction is needed.
   - If the result is invalid, then add '6' to get the valid result.

When the result is invalid, $Y = 1$, otherwise $Y = 0$. See Table 4.7.

Table 4.7

Carry (C_{out})	S_3	S_2	S_1	S_0	Y
0	0	0	0	0	0
0	0	0	1	1	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	1	0	0	0
0	0	1	1	1	0
0	0	1	1	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	1
0	1	0	1	1	1
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
1	1	1	1	1	1

An invalid BCD can be converted to valid BCD by adding 6, i.e. add 6 to the result whenever $Y = 1$.



From the above map, $Y = C_{out} + S_3S_2 + S_3S_1$ (4.25)

The logic diagram of a BCD adder using two 4-bit binarys adders is shown in Fig. 4.28.

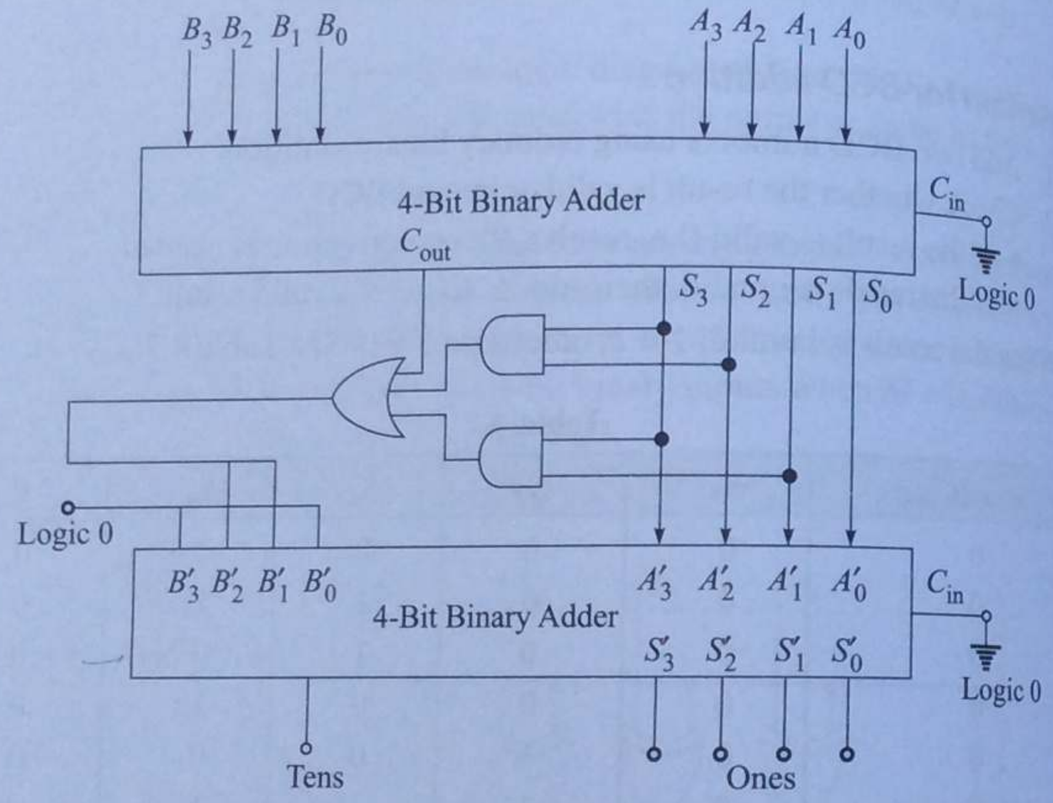


Fig. 4.28 Logic diagram of BCD adder using 4-bit binary adder

The 4-bit binary adder$_1$ adds the BCD numbers A and B with carry = 0. The combinational logic circuit is designed such that $Y = 1$, whenever the result of binary adder$_1$ is invalid. When $Y = 1$, the 4-bit binary adder$_2$ adds 6, otherwise it adds 0 and provides the valid BCD result.

4.6 BCD SUBTRACTOR

In most of the digital systems, the process of subtraction is performed by using the process of addition. The subtraction of BCD B from BCD A is equivalent to the addition of complement of B to A. In BCD subtraction, 9's complement or 10's complement method is used.

4.6.1 9's Complement

The 9's complement of any digit can be obtained by subtracting the digit from 9.
For example, the 9's complement of $4 = 9 - 4 = 5$
9's complement of $3 = 9 - 3 = 6$

Table 4.8 shows the decimal digits and their 9's complements.

Table 4.8

Digit	*9's complement*
0	9 – 0 = 9
1	9 – 1 = 8
2	9 – 2 = 7
3	9 – 3 = 6
4	9 – 4 = 5
5	9 - 5 = 4
6	9 – 6 = 3
7	9 – 7 = 2
8	9 – 8 = 1
9	9 – 9 = 0

The 9's complement can also be obtained by adding $(10)_{10}$ in binary to the 1's complement of that number. For example,

(a)
0 1 0 0 ♦ 4 in binary
1 0 1 1 ♦ 1's complement of 4
+ 1 0 1 0 ♦ Add $(10)_{10}$ in binary
1 0 1 0 1

0101 is 5 in binary and it is the 9's complement of 4.

(b)
1 0 0 1 ♦ 9 in binary
0 1 1 0 ♦ 1's complement of 9
+ 1 0 1 0 ♦ Add $(10)_{10}$ in binary
1 1
1 0 0 0 0

0000 is 0 in binary and it is the 9's complement of 9.

The logic diagram of 9's complement of BCD is shown in Fig. 4.29.

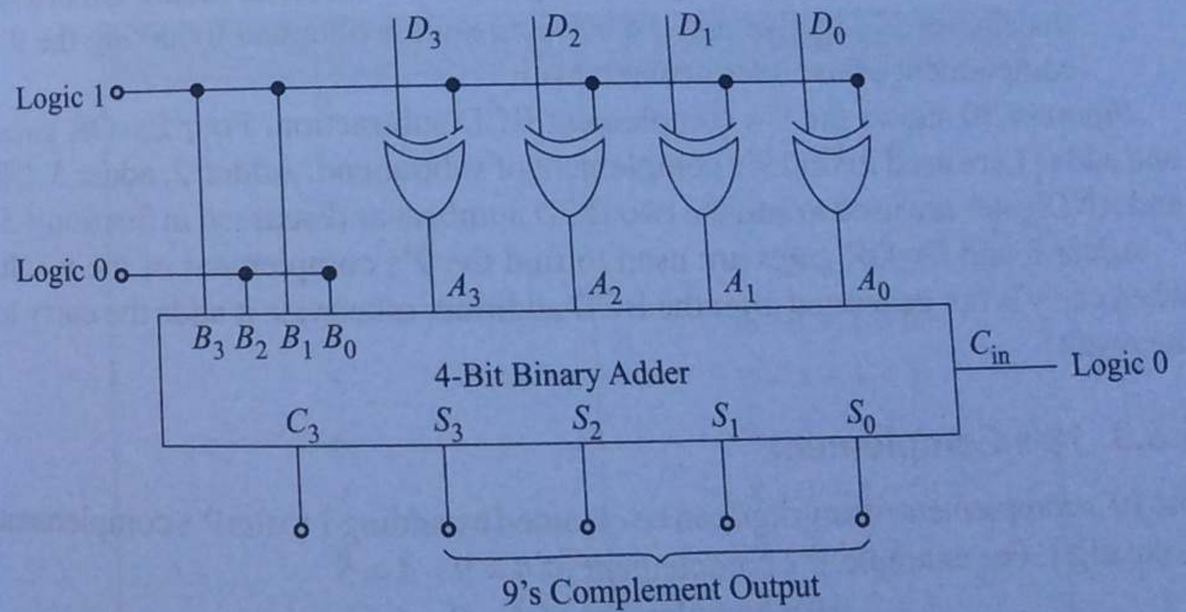


Fig. 4.29 Logic diagram of 9's complement of BCD

The four Ex-OR gates are used to find the 1's complement of the BCD number. A 4-bit binary adder adds $(10)_{10}$ and the 1's complement of BCD number with carry = 0 to get the 9's complement of BCD number at the outputs.

4.6.2 9's Complement Subtraction

The subtraction of two BCD numbers, B from A, is equivalent to addition of 9's complement of B and A, i.e. $A - B = A +$ 9's complement of B.

In 9's complement subtraction, when 9's complement of a smaller number is added to a larger number, carry is generated. This carry is known as end-around carry. This end-around carry is added to the result to get the correct positive result. When the 9's complement of larger number is added to the smaller number, there is no carry. The result is negative and it is in 9's complement form. To obtain the correct result, take the 9's complement of the result. Consider the following examples.

(a) 9 – 3 = 6

```
  1001   ♦ BCD for 9
+ 0110   ♦ 9's complement of 3
------
  1111   ♦ Invalid BCD
+ 0110   ♦ Add 6
    11   ♦ Carry
------
 10101
   └→1   ♦ Add the carry
------
  0110   ♦ BCD for 6
------
∴ Result = + 6
```

(b) 3 – 9 = – 6

```
  0011   ♦ BCD for 3
+ 0000   ♦ 9's complement of 9
------
  0011   ♦ Carry is not generated, result is negative and in 9's complement form
------
∴ Result = – 6
```

Algorithm of 9's complements BCD subtraction

1. Find the 9's complement of subtrahend.
2. Add the two numbers using BCD addition.
3. Check the carry. If carry is generated, the answer is positive and the correct result is obtained by adding the carry to the intermediate result. Otherwise the answer is negative and the correct result is obtained by taking the 9's complement of the intermediate result.

Figure 4.30 shows the 9's complement BCD subtraction. Four Ex-OR gates and adder 1 are used to find 9's complement of subtrahend. Adder 2, adder 3, OR and AND gates are used to add the two BCD numbers as discussed in Section 4.5.

Adder 3 and Ex-OR gates are used to find the 9's complement of the result, when carry is not generated after the BCD addition; otherwise it adds the carry to the result.

4.6.3 10's Complement

The 10's complement of any digit can be obtained by adding 1 to the 9's complement of the digit. For example, 9's complement of 4 = 9 – 4 = 5

10's complement of 4 = 9 – 4 + 1 = 6

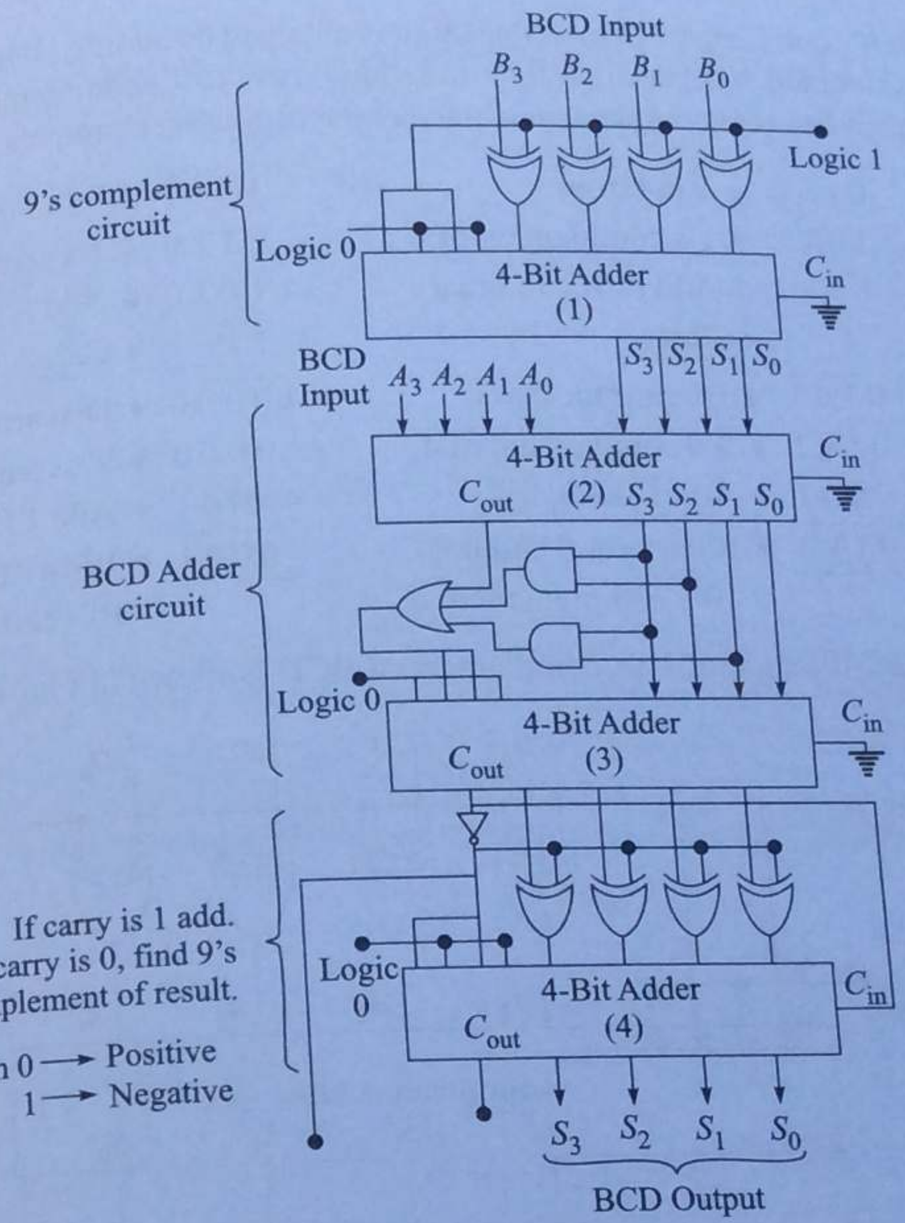


Fig. 4.30 9's complement BCD subtraction

Table 4.9 shows the decimal digits and their 10's complements.

Table 4.9

Digit	*10's complement*
0	9 – 0 + 1 = 10
1	9 – 1 + 1 = 9
2	9 – 2 + 1 = 8
3	9 – 3 + 1 = 7
4	9 – 4 + 1 = 6
5	9 – 5 + 1 = 5
6	9 – 6 + 1 = 4
7	9 – 7 + 1 = 3
8	9 – 8 + 1 = 2
9	9 – 9 + 1 = 1

The 9's complement of a decimal digit is obtained by adding $(10)_{10}$ in binary to the 1's complement of decimal digit, and addition of 1 to 9's complement of decimal digit gives the 10's complement. Consider the following examples.

(a)
```
 0100   ♦ 4 in binary
 1011   ♦ 1's complement of 4
+1010   ♦ Add (10)10 in binary
    1   ♦ Carry
-----
*0101   ♦ Discard the carry
 0101   ♦ 9's complement of 4
+0001   ♦ Add 1 in binary
-----
 0110   ♦ Binary for 6 and it is
          10's complement of 4
```

(b)
```
 1001   ♦ 9 in binary
 0110   ♦ 1's complement of 9
+1010   ♦ Add (10)10 in binary
   11   ♦ Carry
-----
*0000   ♦ Discard the carry
 0000   ♦ 9's complement of 9
+0001   ♦ Add 1 in binary
-----
 0001   ♦ Binary for 1 and it is
          10's complement of 9
```

The logic diagram of 10's complement of BCD is shown in Fig. 4.31.

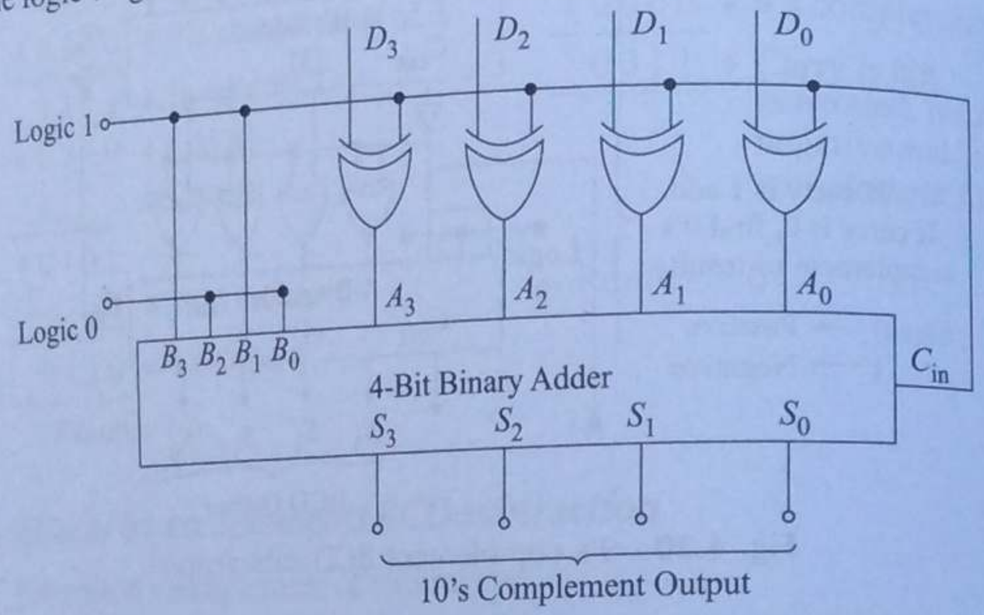


Fig. 4.31 Logic diagram of 10's complement of BCD

The EX-OR gate is used to find the 1's complement of the 4-bit number. The 4-bit binary adder adds the 1's complement and 1010 with the carry to get the 10's complement of the original number.

4.6.4 10's Complement Subtraction

The subtraction of two BCD digits, B from A, is equivalent to the addition of the 10's complement of B and A, i.e. $A - B = A +$ 10's complement of B.

In 10's complement subtraction, when the 10's complement of a smaller number is added to a larger number, a carry is generated. This carry is known as the end-around carry. The carry is discarded to get the correct positive result. When the 10's complement of a large number is added to a smaller number, carry is not generated. The result is negative and it is in 10's complement form. The correct result is obtained by taking the 10's complement of the intermediate result. Consider the following examples.

(a) **9 – 3 = 6**
```
 1001   ♦ BCD for 9
+0111   ♦ 10's complement of 3
-----
  111   ♦ Carry
-----
10000   ♦ Invalid BCD
+0110   ♦ Add 6
-----
*0110   ♦ Discard the carry
 0110   ♦ BCD for 6
```
∴ Result = +6

(b) **3 – 9 = – 6**
```
 0011   ♦ BCD for 3
+0001   ♦ 10's complement of 9
-----
   11   ♦ Carry
-----
 0100   ♦ Carry is not generated, result is negative, and it is in 10's complement form
```
∴ Result = –6

Algorithm of 10's complement BCD subtraction

1. Find the 10's complement of subtrahend.
2. Add two numbers using BCD addition.
3. Check the carry. If carry is generated, the result is positive. The correct result is obtained by discarding the carry. If no carry is generated, the result is negative, it is in 10's complement form, and the correct result is obtained by finding the 10's complement of the result.

Figure 4.32 shows the logic diagram of 10's complement subtraction.

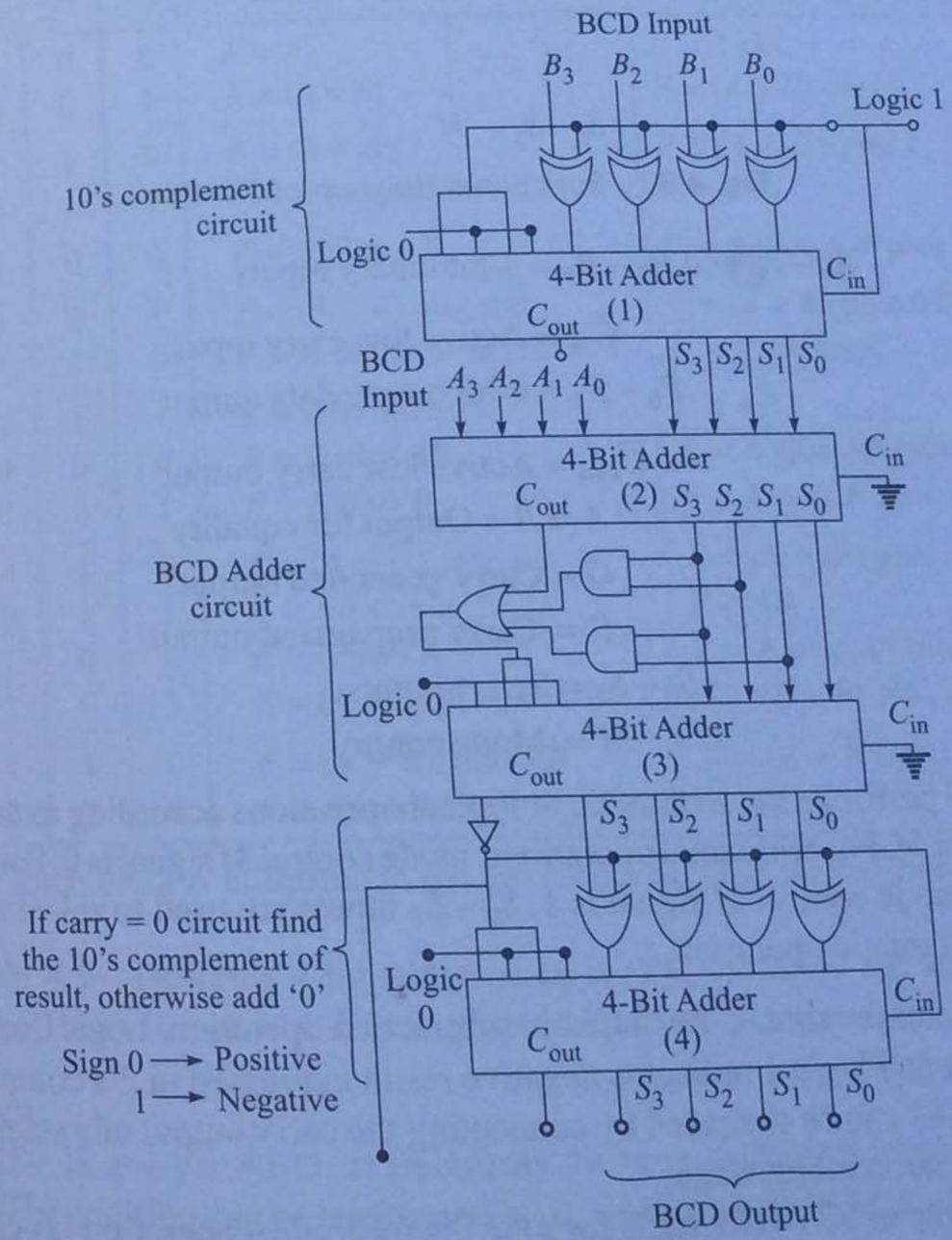


Fig. 4.32 Logic diagram of 10's complement subtraction

Adder (1) and the four EX-OR gates are used to find the 10's complement of the subtrahend as explained in Section 4.6.3.

Adder (2), adder (3), OR and AND gates are used to find the BCD addition of the numbers as explained in Section 4.5.

Adder 4 and EX-OR gates are used to find the 10's complement of the result, when carry is not generated; otherwise it adds 0 in binary to the result.

4.7 ARITHMETIC LOGIC UNIT

The Arithmetic Logic Unit (ALU) is a widely used combinational circuit, which is capable of performing arithmetic as well as logical operations. This is the heart of any microprocessor and microcontroller. IC 74181 is an example of ALU. It is a 4-bit arithmetic logic unit. The functional diagram of 74181 is shown in Fig. 4.33.

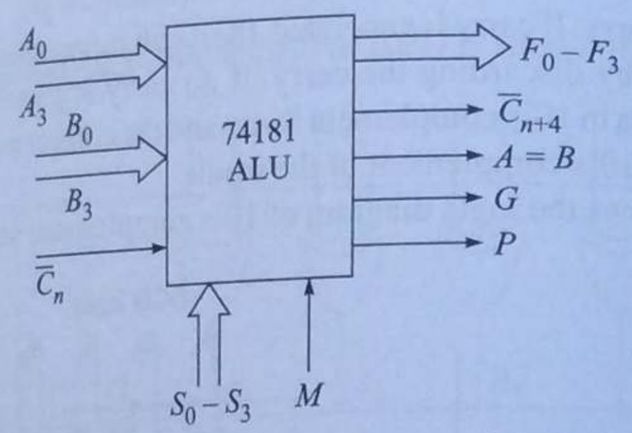


Fig. 4.33 Functional diagram of 74181

$A_0 - A_3$ and $B_0 - B_3$ = 4-bit binary inputs

$\overline{C}_n$ = Active low carry input

$F_0 - F_3$ = 4-bit binary data output

$\overline{C}_{n+4}$ = Active low carry output

$A = B$ = Output for equality

G = Carry generated output

P = Carry propagated output

$S_0 - S_3$ = Select inputs

M = Mode control

The ALU performs the arithmetic or logical operations according to the mode control signal M. For arithmetic operations, mode control M signal is 0. For logical operations, mode control M signal is 1. $S_0 - S_3$ inputs are used to select any one logical or arithmetic operation.

$\overline{C}_{n+4}$ indicates the sign of the output in subtraction operation. Logic 0 indicates positive result and logic 1 indicates negative result expressed in 2's complement form. The 74181 can be cascaded by connecting the carry output of a stage to the carry-in of the succeeding stage.

Function table of IC 74181 ALU for $M = 1$ is for logical operation (as given in Table 4.10) and $M = 0$ is for arithmetic operation (as given in Table 4.11).

Table 4.10 Logical operation ($M = 1$)

Select Inputs				Logic functions ($M = 1$)	Select Inputs				Logic functions ($M = 1$)
S_3	S_2	S_1	S_0		S_3	S_2	S_1	S_0	
0	0	0	0	$F = \overline{A}$	1	0	0	0	$F = \overline{A} + B$
0	0	0	1	$F = \overline{A + B}$	1	0	0	1	$F = \overline{A \oplus B}$
0	0	1	0	$F = \overline{A} \cdot B$	1	0	1	0	$F = B$
0	0	1	1	$F = 0$	1	0	1	1	$F = AB$
0	1	0	0	$F = \overline{AB}$	1	1	0	0	$F = 1$
0	1	0	1	$F = \overline{B}$	1	1	0	1	$F = A + \overline{B}$
0	1	1	0	$F = A \oplus B$	1	1	1	0	$F = \overline{A + B}$
0	1	1	1	$F = A\overline{B}$	1	1	1	1	$F = A$

Table 4.11 Arithmetic operation ($M = 0$)

Select Inputs				$\overline{C}_{n=1}$	$\overline{C}_{n=0}$
S_3	S_2	S_1	S_0		
0	0	0	0	$F = A$	$F = A$ plus 1
0	0	0	1	$F = (A + B)$	$F = (A + B)$ plus 1
0	0	1	0	$F = (A + \overline{B})$	$F = (A + \overline{B})$ plus 1
0	0	1	1	$F = 2$'s complement of A	$F =$ zero
0	1	0	0	$F = A$ plus $A\overline{B}$	$F = A$ plus $A\overline{B}$ plus 1
0	1	0	1	$F = (A + B)$ plus $A\overline{B}$	$F = (A + B)$ plus $A\overline{B}$ plus 1
0	1	1	0	$F = A$ minus B minus 1	$F = A$ minus B
0	1	1	1	$F = A\overline{B}$ minus 1	$F = A\overline{B}$
1	0	0	0	$F = A$ plus AB	$F = A$ plus AB plus 1
1	0	0	1	$F = A$ plus B	$F = A$ plus B plus 1
1	0	1	0	$F = (A + \overline{B})$ plus AB	$F = (A + \overline{B})$ plus AB plus 1
1	0	1	1	$F = AB$ minus 1	$F = AB$
1	1	0	0	$F = A$ plus A^*	$F = A$ plus A^* plus 1
1	1	0	1	$F = (A + B)$ plus A	$F = (A + B)$ plus A plus 1
1	1	1	0	$F = (A + \overline{B})$ plus AB	$F = (A + \overline{B})$ plus AB plus 1
1	1	1	1	$F = A$ minus 1	$F = A$

A^* = Each bit of A is shifted to the next more significant position.

Example 4.9 Design a 4-bit adder using IC 74181. Show the working of the following cases: (a) $A = 4$, $B = 9$, (b) $A = 8$, $B = 9$.

Solution

IC 74181 is a 4-bit ALU. It performs various arithmetic operations when $M = 0$. The addition can be performed by IC 74181 when $M = 0$ and $S = 1001$. The logic diagram of a 4-bit adder using IC 74181 is shown in Fig. 4.34.

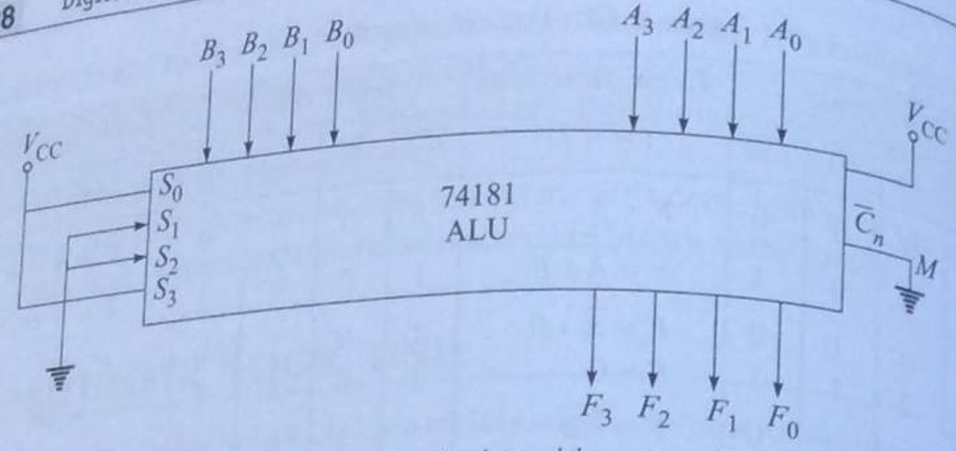


Fig. 4.34 Logic diagram of 4-bit adder using IC 74181

(a) ***F* = *A* plus *B***

0100 ♦ Binary for 4
1001 ♦ Binary for 9
01101

∴ Result is $F = 1101$ and $\overline{C}_{n+4} = 1$

(b) ***F* = *A* plus *B***

1000 ♦ Binary for 8
1001 ♦ Binary for 9
10001

∴ Result is $F = 0001$ and $\overline{C}_{n+4} = 0$

Example 4.10 Design a 4-bit adder using IC 74181. Show the working of the following cases: (a) $A = 97$, $B = 29$, (b) $A = 20$, $B = 57$.

Solution

IC 74181 is a 4-bit ALU. It performs addition when $M = 0$ and $S = 1001$. For an 8-bit addition, it is necessary to use two ICs in cascade. The logic diagram of an 8-bit adder using IC 74181 is shown in Fig. 4.35.

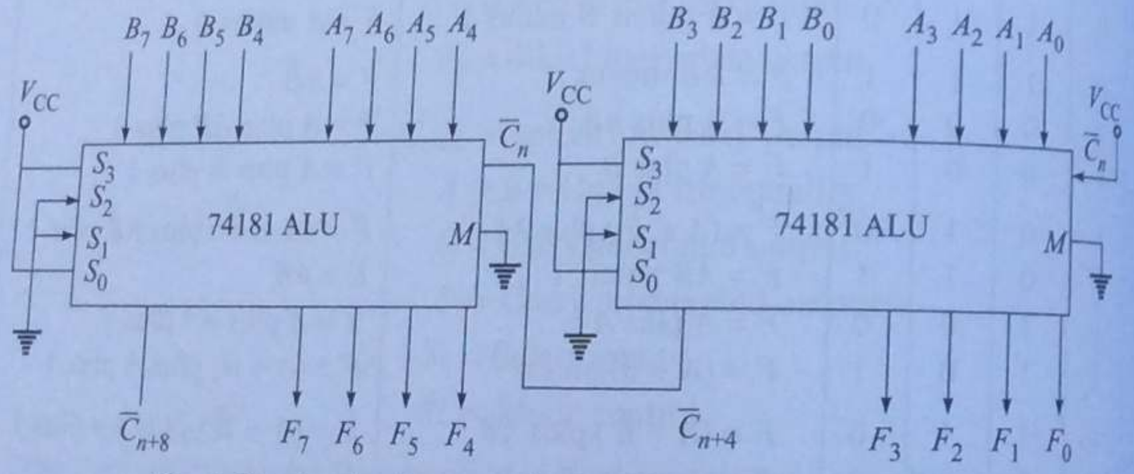


Fig. 4.35 Logic diagram of 8-bit adder using IC 74181

(a) ***F* = *A* plus *B* plus carry**

$F_L = A_L$ plus B_L plus C_{n+0}
$F_H = A_H$ plus B_H plus C_{n+4}
$A = 97$ and $B = 29$

01100001 ♦ Binary for 97
00011101 ♦ Binary for 29
001111110 ♦ Binary for 126

There is no carry after the addition of least significant 4-bit. Hence $C_{n+4} = 0$ and $\overline{C}_{n+4} = 1$. Similarly, there is no carry after the addition of 8-bit. Hence $C_{n+8} = 0$ and $\overline{C}_{n+8} = 1$.

∴ Result is $F = 01111110$, $\overline{C}_{n+4} = 1$, $\overline{C}_{n+8} = 1$.

(b) ***F* = *A* plus *B* plus carry**

$F_L = A_L$ plus B_L plus C_{n+0}
$F_H = A_H$ plus B_H plus C_{n+4}
$A = 20$ and $B = 57$

00010100 ♦ Binary for 20
00111001 ♦ Binary for 57
001001101 ♦ Binary for 77

There is no carry after the addition of least significant 4-bit. Hence $C_{n+4} = 0$ and $\overline{C}_{n+4} = 1$. Similarly, there is no carry after the addition of 8-bit. Hence $C_{n+8} = 0$ and $\overline{C}_{n+8} = 1$.

∴ Result is $F = 01001101$, $\overline{C}_{n+4} = 1$, $\overline{C}_{n+8} = 1$.

4.8 COMPARATOR

The comparator is a combinational logic circuit. It compares the magnitude of two n-bit numbers and provides the relative result as the output. The block diagram of n-bit comparator is shown in Fig. 4.36.

A and B are the two n-bit inputs, and $A > B$, $A = B$, and $A < B$ are the outputs. Depending upon the relative magnitude of the numbers, one of the outputs will be high. The truth table of 2-bit comparator is given in Table 4.12.

Table 4.12 The truth table for 2-bit comparator

Inputs				*Outputs*		
A_1	A_2	B_1	B_2	$A > B$	$A = B$	$A < B$
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

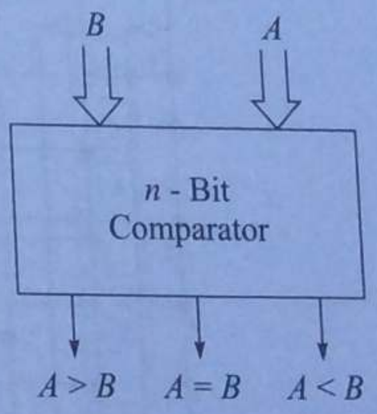


Fig. 4.36 Block diagram of n-bit comparator

B_1B_0 \ A_1A_0	00	01	11	10
00	0	[1]	[1]	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

Map for $A > B$

B_1B_0 \ A_1A_0	00	01	11	10
00	[1]	0	0	0
01	0	[1]	0	0
11	0	0	[1]	0
10	0	0	0	[1]

Map for $A = B$

B_1B_0 \ A_1A_0	00	01	11	10
00	0	0	0	0
01	[1]	0	0	0
11	[1]	[1]	0	[1]
10	[1]	[1]	0	0

Map for $A < B$

$$A > B = A_1\bar{B}_1 + A_0\bar{B}_1\bar{B}_0 + A_1A_0\bar{B}_0$$

$$A = B = \bar{A}_1\bar{B}_1\bar{A}_0\bar{B}_0 + \bar{A}_1A_0\bar{B}_1B_0 + A_1A_0B_1B_0 + A_1\bar{A}_0B_1\bar{B}_0$$

$$= \bar{A}_1\bar{B}_1(\bar{A}_0\bar{B}_0 + A_0B_0) + A_1B_1(A_0B_0 + \bar{A}_0\bar{B}_0)$$

$$= \bar{A}_1\bar{B}_1(A_0 \odot B_0) + A_1B_1(A_0 \odot B_0)$$

$$= (A_0 \odot B_0)(\bar{A}_1\bar{B}_1 + A_1B_1)$$

$$= (A_0 \odot B_0)(A_1 \odot B_1)$$

$$A < B = \bar{A}_1B_1 + \bar{A}_1\bar{A}_0B_0 + \bar{A}_0B_1B_0$$

The logic diagram of a 2-bit comparator using gates is shown in Fig. 4.37.

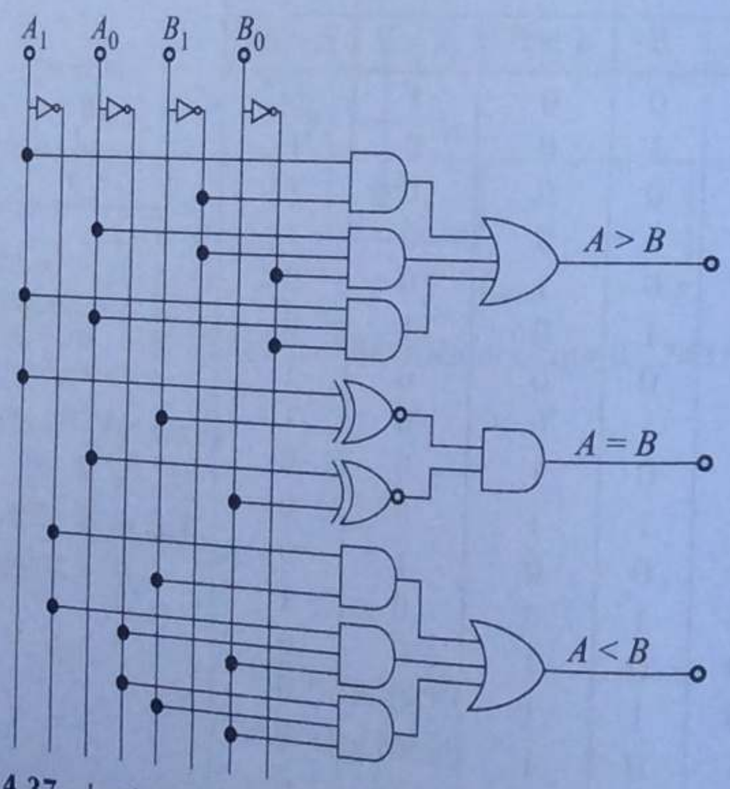


Fig. 4.37 Logic diagram of two-bit comparator using gates

4.8.1 IC 7485—4-Bit Comparator

IC 7485 is a 4-bit comparator. It compares the two 4-bit binary numbers. It has three outputs and three cascading inputs, i.e. $A > B$, $A = B$, and $A < B$. These ICs can be cascaded to compare words of any length. The logic symbol of IC 7485 is shown in Fig. 4.38.

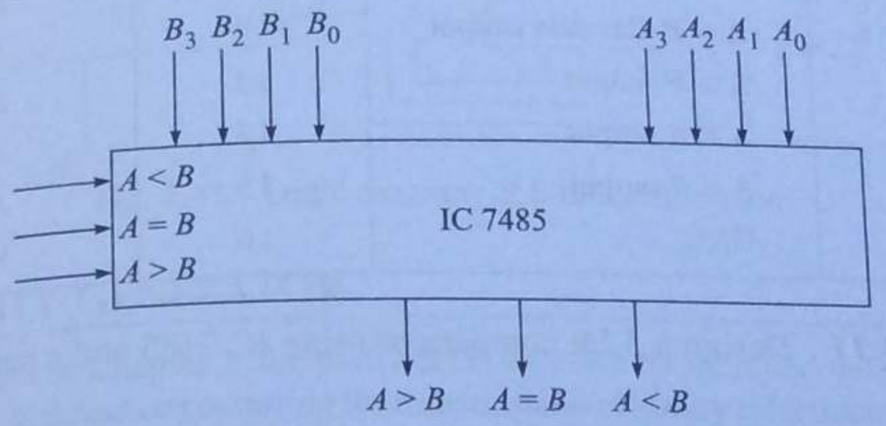


Fig. 4.38 The logic symbol of IC 7485

The truth table of IC 7485 comparator is given in Table 4.13.

Table 4.13 Truth table of IC 7485 comparator

Comparing inputs	*Cascading inputs*			*Outputs*		
	$A > B$	$A = B$	$A < B$	$A > B$	$A = B$	$A < B$
A > B	X	X	X	1	0	0
A = B	1	0	0	1	0	0
	X	1	X	0	1	0
	X	0	0	1	0	1
	0	0	0	1	0	1
	1	0	1	0	0	0
A < B	X	X	X	0	0	1

The functional diagram of IC 7485 is shown in Fig. 4.39 and the description of pins are given in Table 4.14.

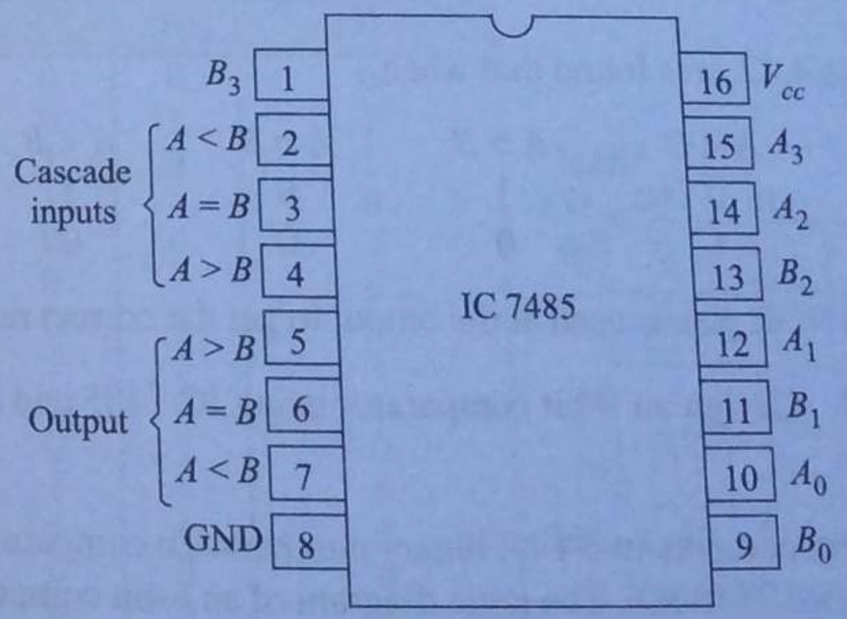


Fig. 4.39 Functional diagram of IC 7485

Table 4.13 Description of pins of IC 7485

Pin no.	Description	Pin no.	Description
1	B_3	9	B_0
2	$A < B$ Cascade input	10	A_0
3	$A = B$ Cascade input	11	B_1
4	$A > B$ Cascade output	12	A_1
5	$A > B$ output	13	B_2
6	$A = B$ output	14	A_2
7	$A < B$ output	15	A_3
8	GND	16	V_{cc}

Example 4.11 Design a 5-bit comparator using IC 7485 and a suitable gate.

Solution

IC 7485 is a 4-bit comparator. It has three outputs and three cascading inputs, viz. $A > B$, $A = B$, and $A < B$. It is possible to design the 5-bit comparator using single IC 7485 with the help of cascading inputs as shown in Fig. 4.40.

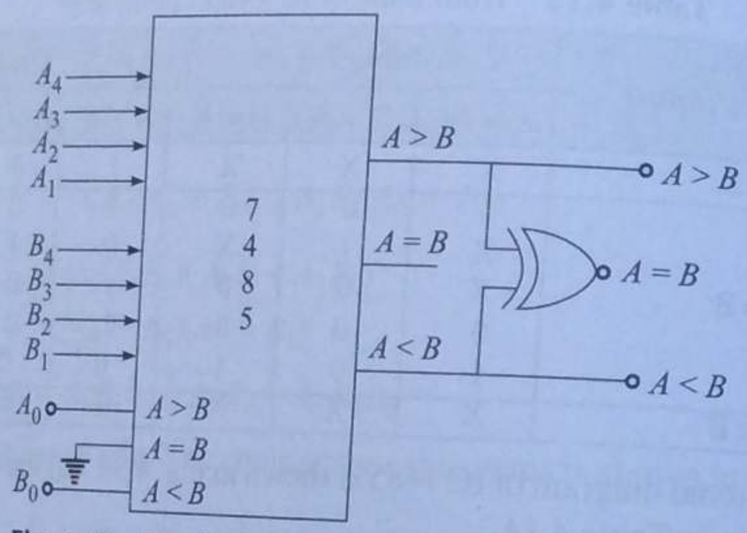


Fig. 4.40 Five-bit comparator using single IC 7485

From Table 4.12, it is found that when,

$A > B$	$A < B$	$A > B$	$A = B$	$A < B$
0	0	1	0	1
$A = B$	1	0	0	0

Hence, EX-NOR gate is used at the output to get the correct result, $A = B$.

Example 4.12 Design an 8-bit comparator using IC 7485 and a suitable gate.

Solution

IC 7485 is used to compare two 4-bit binary numbers. To compare 8-bit numbers, we need to use two 7485 ICs. The logic diagram of an 8-bit comparator is shown in Fig. 4.41.

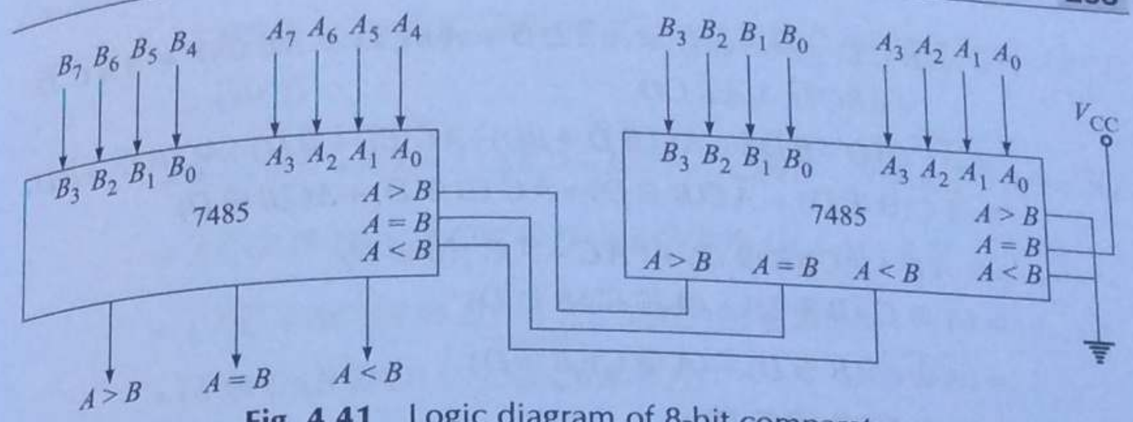


Fig. 4.41 Logic diagram of 8-bit comparator

4.9 PARITY GENERATOR

As mentioned in Chapter 2, the parity bit is an extra bit included with the binary information to detect errors during the transmission of binary information. In digital communication, an extra bit is added in binary message such that the total number of 1s in the message can be either odd or even according to the type of parity used. The parity generator is a combinational logic circuit that generates the parity bit(s). There are two types of parity generators and parity checkers:

1. Even parity generator
2. Odd parity generator

4.9.1 Even Parity Generator

The even parity generator is a combinational logic circuit that generates the parity bit such that the number of 1s in the message becomes even. The parity generator checks the input's binary information and generates the parity bit 0/1 such that after the addition of parity bit, the total number of 1s in the message becomes even. Table 4.15 shows the 4-bit information with even parity.

Table 4.15 4-bit information with even parity

4-bit information				Even parity
A	B	C	D	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	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

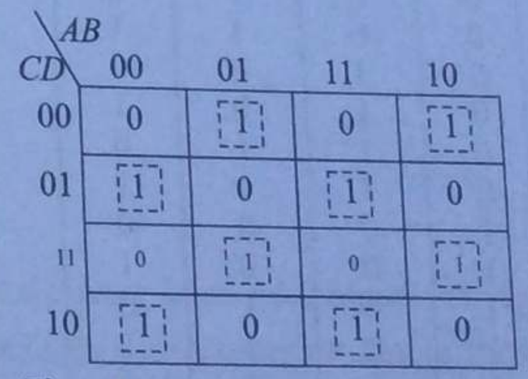


Fig. 4.42 K-map for P with inputs A, B, C, and D

The K-map for P with inputs A, B, C, and D is shown in Fig. 4.42.

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

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

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

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

$$= (A \odot C)(B \oplus D) + (A \oplus C)(B \odot D)$$

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

$$= (A \oplus C) \oplus (B \oplus D)$$

$$= A \oplus B \oplus C \oplus D$$

The logic diagram of even parity generator is shown in Fig. 4.43.

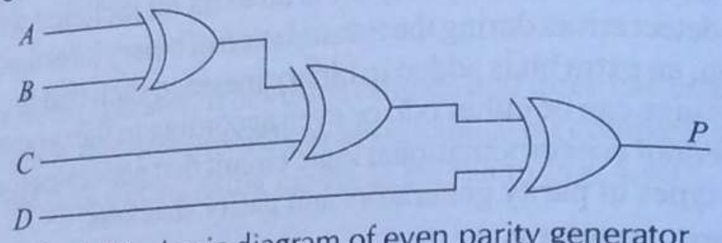


Fig. 4.43 Logic diagram of even parity generator

4.9.2 Odd Parity Generator

The odd parity generator is a combinational logic circuit that generates the parity bit such that the number of 1s in the message becomes odd. The parity generator checks the input's binary information and generates the parity bit 0/1 such that after the addition of parity bit, the total number of 1s in the message becomes odd. Table 4.16 shows the 4-bit information with odd parity.

Table 4.16 4-bit information with odd parity

4-bit information				*Odd parity*
A	*B*	*C*	*D*	*P*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

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

Fig. 4.44 K-map for *P* with inputs *A*, *B*, *C*, and *D*

The K-map for *P* with inputs *A*, *B*, *C*, and *D* is shown in Fig. 4.44.

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

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

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

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

$$= (A \odot C)(B \odot D) + (A \oplus C)(B \oplus D)$$

$$= \left(\overline{(A \oplus C)}\,\overline{(B \oplus D)}\right) + (A \oplus C)(B \oplus D)$$

$$= (A \oplus C) \odot (B \oplus D)$$

The logic diagram of odd parity generator is shown in Fig. 4.45.

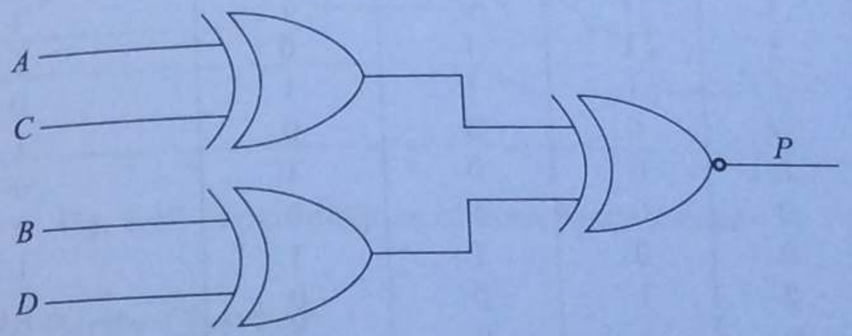


Fig. 4.45 Logic diagram of odd parity generator

4.10 PARITY CHECKER

At the receiving end, a combinational logical circuit is used to check the parity of the received information. It determines whether the error is included in the message or not. The combinational logic circuit used at the receiver to check the parity of received information is known as the *parity checker*.

There are two types of parity checkers:

1. Even parity checker
2. Odd parity checker

4.10.1 Even Parity Checker

The even parity checker is a combinational logical circuit. It has *n*-bit inputs message and a parity error as the output. The circuit checks the parity of inputs and provides the output 0/1. For an even parity checker, if the parity of input message is even, then the output is 0, otherwise the output is 1. When the output is 1, it shows there is error in the message. The truth table of a 5-bit even parity checker is given in Table 4.17.

Table 4.17 5-bit even parity checker

5-bit message					Parity error
A	B	C	D	P	PE
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	1	0	1
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	0
1	1	1	0	0	1
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	1

The K-map for PE with inputs A, B, C, D, and P is shown in Fig.4.46.

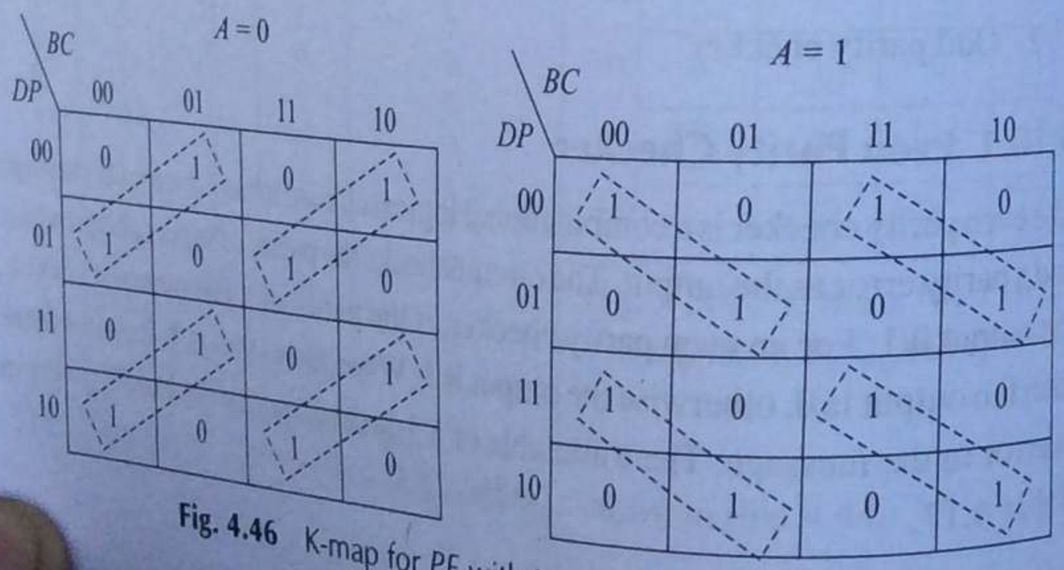


Fig. 4.46 K-map for PE with inputs A, B, C, D, P

$$PE = \overline{A}\,\overline{B}\,\overline{D}\,(C \oplus P) + \overline{A}\,\overline{B}\,D(C \odot P) + \overline{A}\,B\,\overline{D}\,(C \odot P) + \overline{A}\,BD(C \oplus P) + A\overline{B}\,\overline{D}\,(C \odot P) + A\overline{B}\,D(C \oplus P) + AB\,\overline{D}\,(C \oplus P) + ABD(C \odot P)$$

$$= \overline{A}\,\overline{B}[\overline{D}(C \oplus P) + D(\overline{C \oplus P})] + \overline{A}\,B(\overline{D}(\overline{C \oplus P}) + D(C \oplus P)) + A\overline{B}(\overline{D}(\overline{C \oplus P}) + D(C \oplus P)) + AB(\overline{D}(C \oplus P) + D(\overline{C \oplus P}))$$

$$= [(\overline{A}B + A\overline{B})(\overline{D}(\overline{C \oplus P})] + D(C + P) + (\overline{A}\,\overline{B} + AB)(\overline{D}(C \oplus P) + D(\overline{C \oplus P}))((A \oplus B)(D \odot (C \oplus P)) + (A \odot B)(D \oplus C \oplus P)$$

$$= [(A \oplus B)(\overline{D \oplus C \oplus P})] + [(\overline{A \oplus B})(D \oplus C \oplus P)]$$

$$= A \oplus B \oplus C \oplus D \oplus P$$

The logic diagram of an even parity checker is shown in Fig. 4.47.

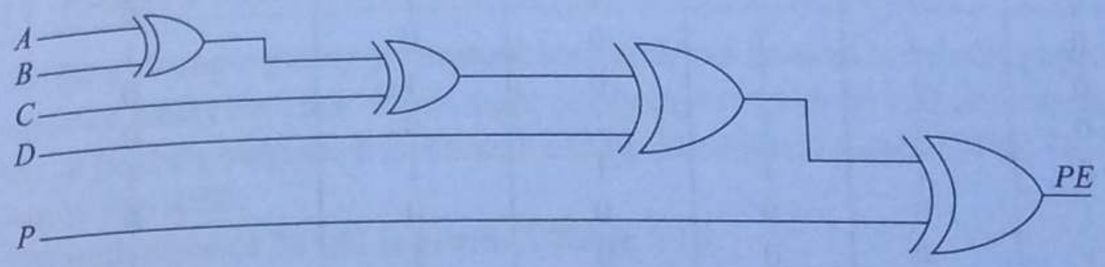


Fig. 4.47 Logic diagram of even parity checker

4.10.2 Odd Parity Checker

The odd parity checker is a combinational logical circuit. It has n-bit inputs message and a parity error as the output. The circuit checks the parity of inputs and provides the output 0/1. For an odd parity checker, if the parity of input message is odd, then the output is 0, otherwise it is 1. When the output is 1, it shows there is error in the message. The truth table of a 5-bit odd parity checker is given in Table 4.18.

$$PE = \overline{A}\,\overline{B}\,\overline{D}\,(C \odot P) + \overline{A}\,\overline{B}\,D(C \oplus P) + \overline{A}\,B\,\overline{D}\,(C \odot P) + \overline{A}\,BD(C \oplus P) + \overline{A}\,BD(C \oplus P) + A\overline{B}\,\overline{D}\,(C \odot P) + AB\,\overline{D}\,(C \odot P) + ABD(C \oplus P)$$

$$= \overline{A}\,\overline{B}[\overline{D}(C \odot P) + D(C \oplus P)] + \overline{A}B[\overline{D}(C \oplus P) + D(C \oplus P)] + A\overline{B}[\overline{D}(C \oplus P) + D(C \oplus P)] + AB[\overline{D}(C \odot P) + D(C + P)]$$

$$= \overline{A}\,\overline{B}[\,\overline{D}\,(\overline{C \oplus P}) + D(C \oplus P)] + \overline{A}B[\,\overline{D}\,(C \oplus P) + D(\overline{C \oplus P})] + A\overline{B}[\,\overline{D}\,(C \oplus P) + D(\overline{C \oplus P})] + AB[\,\overline{D}\,(\overline{C \oplus P}) + D(C \oplus P)]$$

$$= [(\overline{A}\,\overline{B} + AB)(\,\overline{D}(\overline{C \oplus P}) + D(C \oplus P))] + [(A\overline{B} + \overline{A}B)(\,\overline{D}\,(C \oplus P) + D(\overline{C \oplus P}))]$$

$$= [(A\overline{B} + \overline{A}B)]\,\overline{D}\,(C \oplus P) + D(\overline{C \oplus P})$$

$$= [(A \odot B)(D \odot (C \oplus P))] + [(A \odot B)(D \odot (C \oplus P)]$$

$$PE = [(\overline{A \oplus B})(\overline{D \oplus C \oplus P})] + (A \oplus B) \oplus (D \oplus C \oplus P)$$

$$= (A \oplus B) \odot (D \oplus C \oplus P)$$

Table 4.18 5-bit odd parity checker

5-bit message					Parity error
				P	PE
A	B	C	D		
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	0
0	0	1	0	0	0
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	0	1	0
0	1	1	1	0	0
0	1	1	1	1	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	0	1	0	1
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	1
1	1	1	1	0	1
1	1	1	1	1	0

The K-map for *PE* with inputs *A*, *B*, *C*, *D*, and *P* is shown in Fig. 4.48.

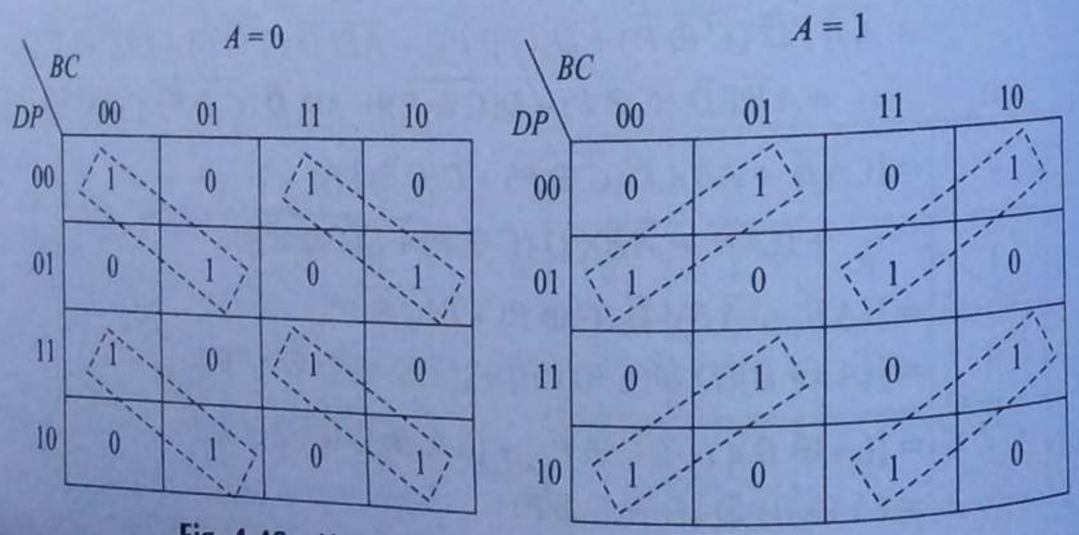


Fig. 4.48 K-map for *PE* with inputs *A*, *B*, *C*, *D*, *P*

The logic diagram of an odd parity checker is shown in Fig. 4.49.

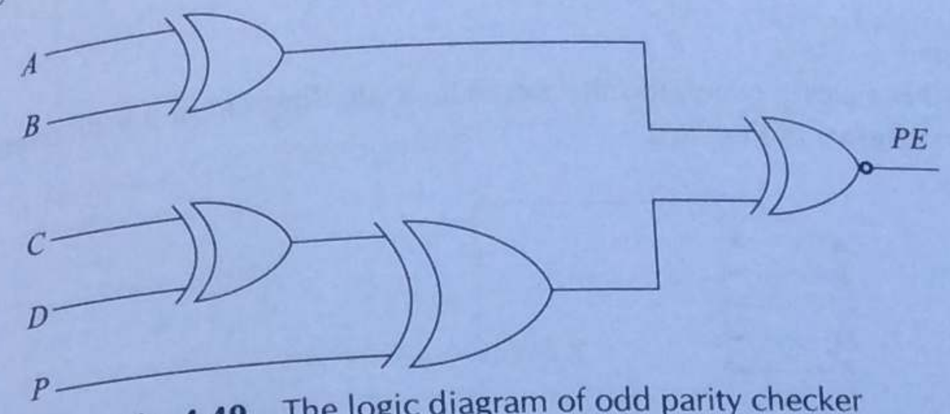


Fig. 4.49 The logic diagram of odd parity checker

4.11 PARITY GENERATOR/CHECKER (IC74180)

IC 74180 is a 9-input parity generator/checker. It can be used as a parity generator as well as a parity checker. It has eight parity inputs (*A* to *H*) and two cascading inputs. It has two outputs, Σ Even and Σ Odd. The logical symbol of IC 74180 is shown in Fig. 4.50.

The truth table of 74180 is given in Table 4.19.

Table 4.19 Truth table of 74180

Parity Inputs	Cascading Inputs		Outputs	
	Even	Odd	Σ Even	Σ Odd
Even	1	0	1	0
Odd	1	0	0	1
Even	0	1	0	1
Odd	0	1	1	0
X	1	1	0	0
X	0	0	1	1

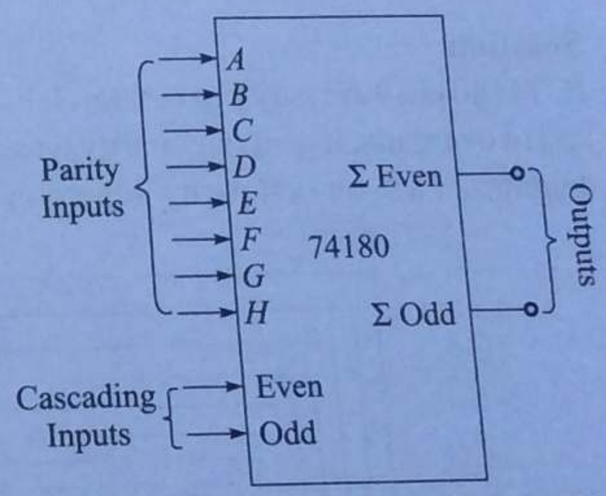


Fig.4.50 Logical symbol of IC 74180

From the truth table of IC 74180, it is observed that

Parity inputs	+	Cascading input	=	Output
Even	+	Even	=	Even
Odd	+	Odd	=	Even
Odd	+	Even	=	Odd
Even	+	Odd	=	Odd

When the cascading inputs are logic 1, then the outputs are logic 0; and when both cascading inputs are logic 0, then the outputs are logic1.

Example 4.13 Design a 9-bit even parity checker using IC 74180 and a suitable gate.

Solution

IC 74180 is a parity generator/checker. The logic diagram of a 9-bit even parity checker is shown in Fig. 4.51.

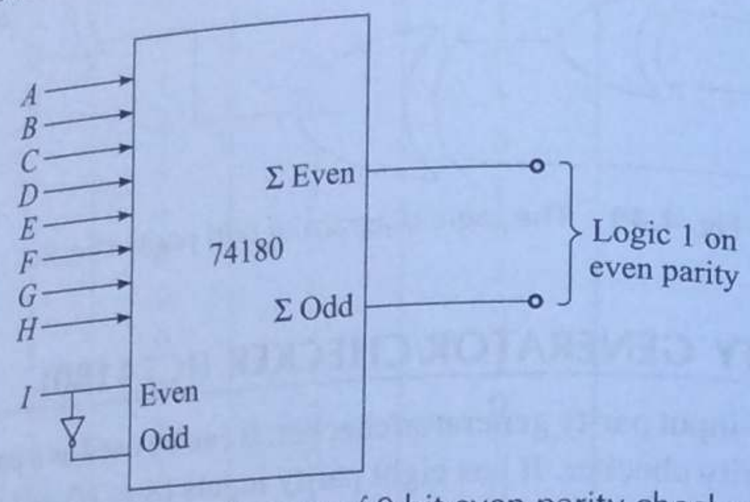


Fig. 4.51 Logic diagram of 9-bit even parity checker

Example 4.14 (a) Design a 9-bit odd parity generator using IC 74180.
(b) Design a 10-bit odd parity generator using IC 74180.

Solution

IC 74180 is a 9-bit parity generator. It has eight parity inputs, two cascading inputs, and two outputs. It generates parity bit according to the 8 input messages. The logic diagram of a 9-bit odd parity generator is shown in Fig. 4.52 (a).

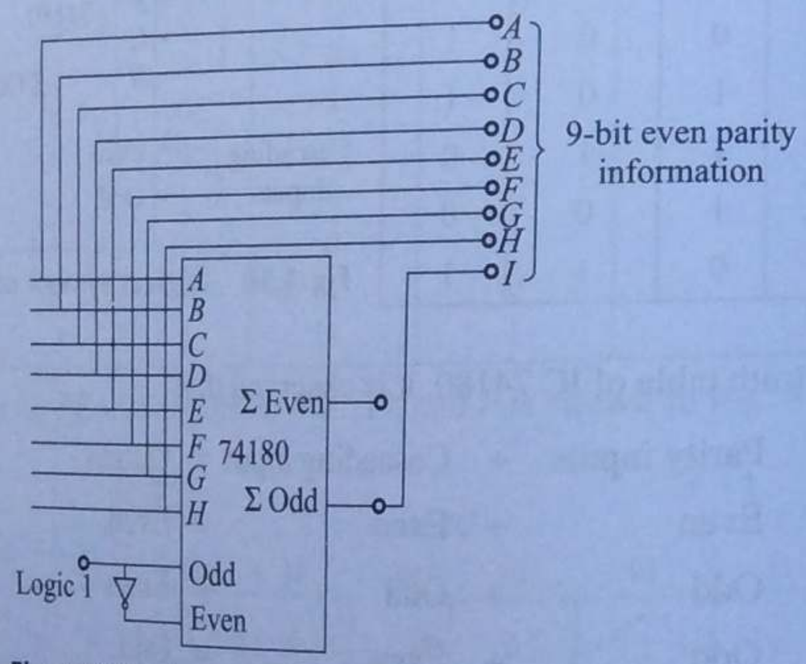


Fig. 4.52(a) Logic diagram of 9-bit odd parity generator

The logic diagram of a 10-bit odd parity generator is shown in Fig. 4.52 (b).

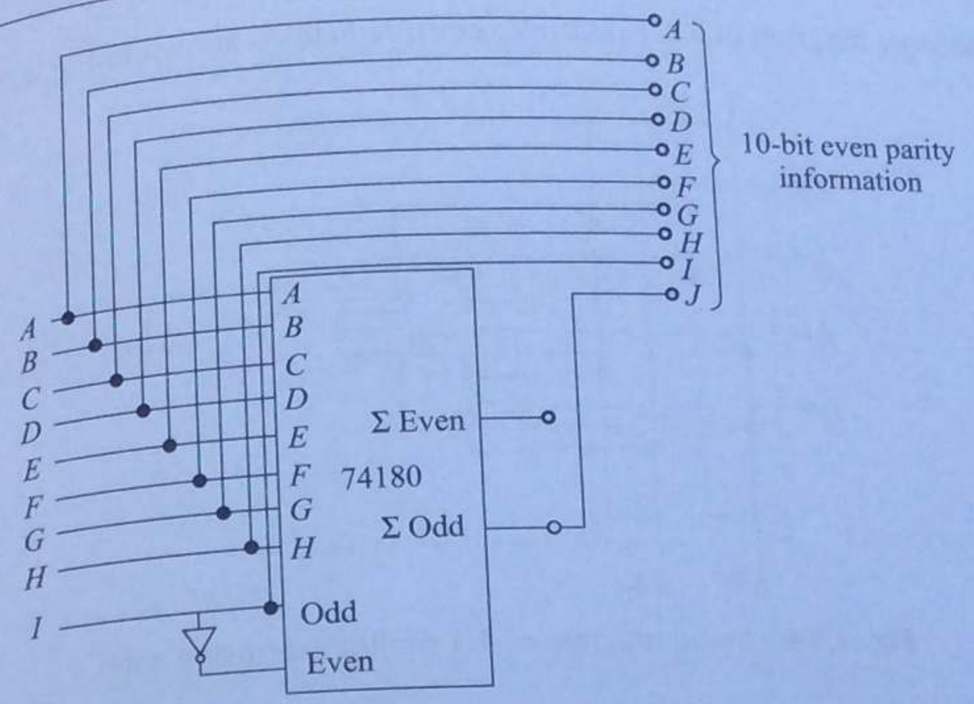


Fig. 4.52(b) Logic diagram of 10-bit odd parity generator

4.12 MULTIPLEXER

The multiplexer is a combinational logic circuit with multiple inputs and a single output. Selectors select one input at a time and send it to the output line. For an n-input multiplexer, m select inputs are required, where $n = 2^m$. The block diagram of an n-input multiplexer is shown in Fig. 4.53.

In Fig. 4.53, E is active low enable input. When this input is low, the chip is enabled.

A multiplexer is also known as a *data selector*. Depending upon the digital code applied at the selector lines, one of the data inputs is selected and passed to a output channel.

Let us consider a 4-input multiplexer as an example. For a 4-input multiplexer, the number of select lines is two ($2^2 = 4$). The truth table of a 4:1 multiplexer is given in Table 4.20.

Fig. 4.53 Block diagram of n-input multiplexer

Table 4.20 Truth table of 4:1 multiplexer

Select lines		*Outputs*
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

The expression for output Y can be written as

$$Y = \overline{S}_1 \overline{S}_0 I_0 + \overline{S}_1 S_0 I_1 + S_1 \overline{S}_0 I_2 + S_1 S_0 I_3$$

The logic diagram of a 4:1 multiplexer using gates is shown in Fig. 4.54.

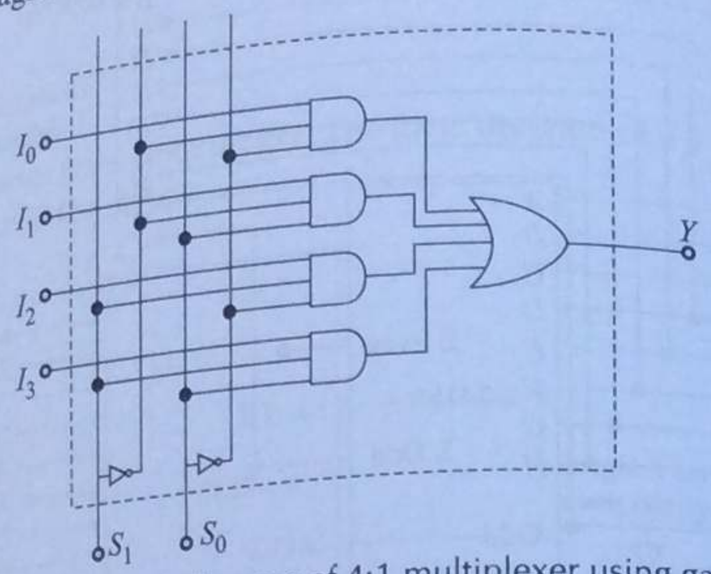


Fig. 4.54 Logic diagram of 4:1 multiplexer using gates

4.12.1 Multiplexer Tree

The 4:1, 8:1, and 16:1 multiplexers are available in the form of ICs as given in Table 4.21.

Table 4.21 Multiplexers ICs

IC No.	*Description*	*Output*
74157	Quad 2:1	Same as input
74158	Quad 2:1	Inverted input
74153	Dual 4:1	Same as input
74352	Dual 4:1	Inverted input
74151A	8:1	Complementary output
74152	8:1	Inverted input
74150	16:1	Inverted input

The largest available ICs of multiplexers are 16:1. Therefore, to implement a multiplexer with more than 16 inputs, there should be provision for expansion. The multiplexer has an enable (strobe) input, which helps in its expansion.

Let us consider the example of an 8:1 multiplexer using two 4:1 multiplexers as shown in Figure 4.55. The expansion of a multiplexer is referred to as *multiplexer tree*.

For an 8:1 multiplexer, the number of select inputs should be 3. The 4:1 multiplexer has only two select inputs and hence, the enable input of 4:1 multiplexer is used as the third select input S_2. Whenever S_2 is logic 0, the IC M_1 will be enabled and M_2 will be disabled, depending upon the binary inputs at S_1 and S_0, and one of the inputs of M_1 is sent to the output. Whenever S_2 is logic 1, M_2 is enabled and M_1 is disabled, depending upon the binary inputs at S_1 and S_0, and one of the inputs of M_2 is sent to the output. The truth table of 8:1 multiplexer using two 4:1 multiplexers is shown in Table 4.21.

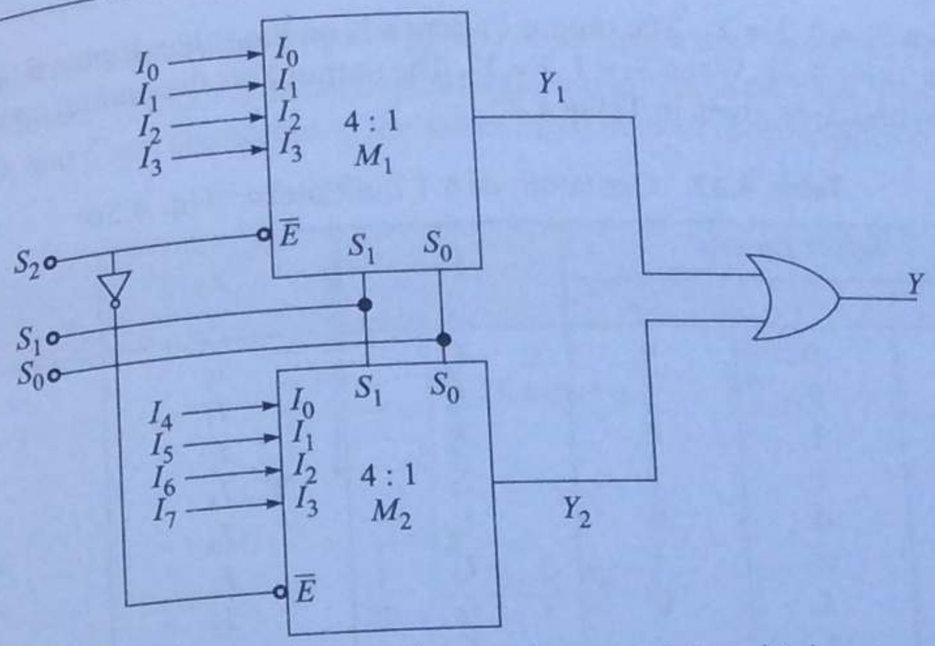


Fig. 4.55 8:1 multiplexer using two 4:1 multiplexers

Table 4.21 Truth table of 8:1 multiplexer using two 4:1 multiplexers

Select lines			*Enabled*	*Disabled*	*Output*
S_2	S_1	S_0			*Y*
0	0	0	m_1	m_2	I_0
0	0	1	m_1	m_2	I_1
0	1	0	m_1	m_2	I_2
0	1	1	m_1	m_2	I_3
1	0	0	m_2	m_1	I_4
1	0	1	m_2	m_1	I_5
1	1	0	m_2	m_1	I_6
1	1	1	m_2	m_1	I_7

It is also possible to implement an 8:1 multiplexer using 4:1 and 2:1 multiplexers as shown in Fig. 4.56.

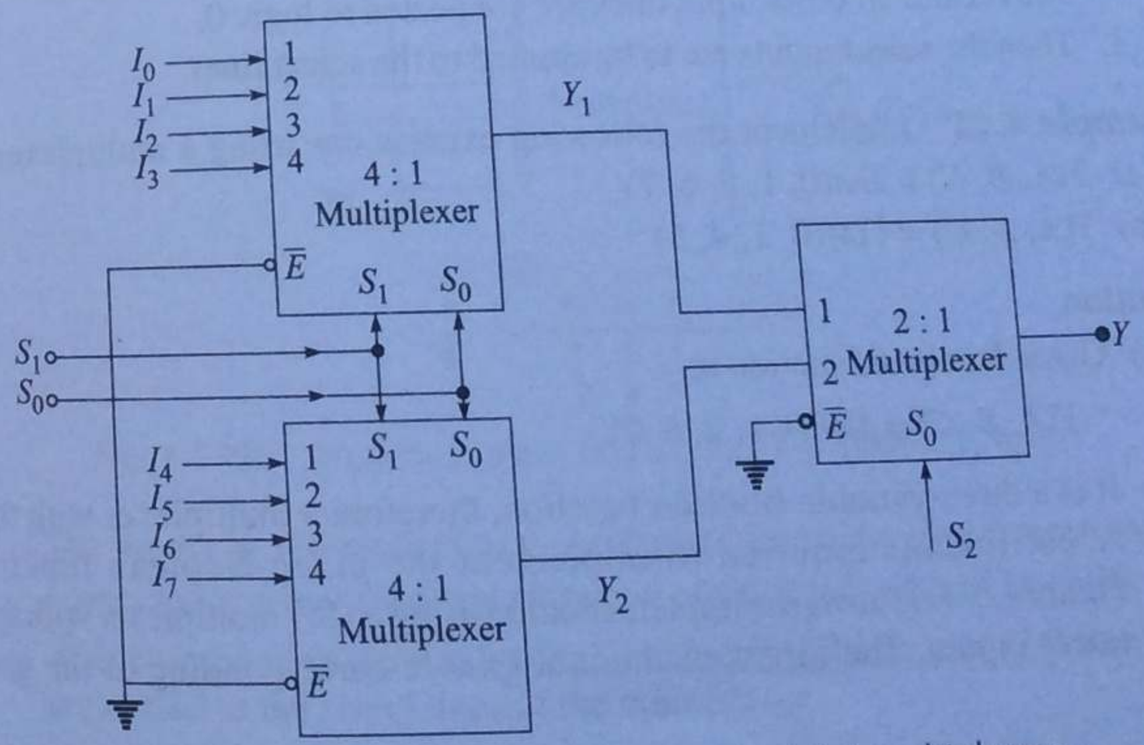


Fig. 4.56 8:1 multiplexer using 4:1 and 2:1 multiplexers

When $S_2 = 0$, $Y = Y_1$. The output Y_1 depends on the select inputs S_1 and S_0 as given in Table 4.21. When $S_2 = 1$, $Y = Y_2$. The output Y_2 is depending on the select inputs S_1 and S_0 as given in Table 4.22.

Table 4.22 Operation of 8:1 multiplexer (Fig. 4.56)

Select Lines			Y_2	Y_1	Y
S_2	S_1	S_0			
0	0	0	I_4	I_0	I_0
0	0	1	I_5	I_1	I_1
0	1	0	I_6	I_2	I_2
0	1	1	I_7	I_3	I_3
0	0	0	I_4	I_0	I_4
1	0	1	I_5	I_1	I_5
1	1	0	I_6	I_2	I_6
1	1	1	I_7	I_3	I_7

4.12.2 Multiplexer Applications

Multiplexer means *many into one*. It is a device that allows placing digital information from several inputs on a single output line as per the select inputs. Any application of combinational logic circuit can be implemented by using multiplexers and external gates. Multiplexers are widely used and they offer the following advantages:

1. Simplification of logic expression is not required,
2. It minimizes the IC count, and
3. Logic design is simplified.

The design procedure of a combinational logic circuit using multiplexer is as follows:

1. Identify the decimal number corresponding to each minterm in the expression.
2. The input lines corresponding to these numbers are to be connected to logic 1 level and all other input lines are connected to logic 0.
3. Then the select inputs are to be applied to the select lines.

Example 4.15 Implement the following expressions using a multiplexer.

(a) $Y(A, B, C) = \Sigma m(0, 1, 2, 6, 7)$

(b) $Y(A, B, C) = \Pi M(0, 1, 4, 5)$

Solution

(a) Given Boolean function is

$$Y(A, B, C) = \Sigma m(0, 1, 2, 6, 7)$$

It is a three-variable Boolean function, therefore a multiplexer with three select lines is required to implement the given Boolean function. Figure 4.57(a) shows the implementation using an 8:1 multiplexer with three select inputs. The inputs of the multiplexer corresponding to the given minterms are connected to V_{CC} and other inputs are connected to ground. The enable input $\overline{E}$ is connected to ground to enable the chip. The inputs A, B, and C are connected to the select input lines of the multiplexer.

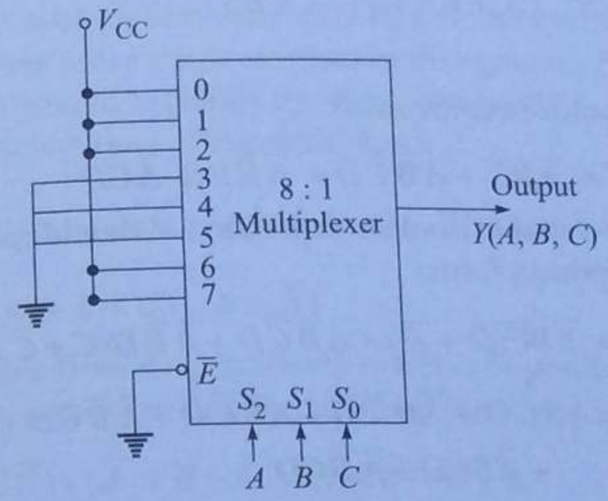


Fig. 4.57(a) Implementation of $Y(A, B, C) = \Sigma m(0, 1, 2, 6, 7)$

(b) Given Boolean function is

$$Y(A, B, C) = \Pi M(0, 1, 4, 5)$$

It is a three-variable Boolean function in terms of maxterms. It can be expressed in minterms as

$$Y(A, B, C) = \Sigma m(2, 3, 6, 7)$$

It is a three-variable Boolean function, therefore a multiplexer with three select lines is required to implement the given Boolean function. Figure 4.57(b) shows the implementation using an 8:1 multiplexer with three select inputs.

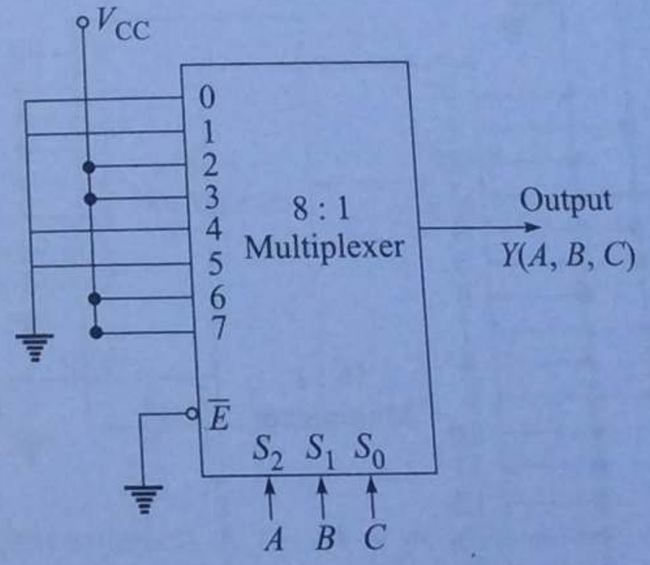


Fig. 4.57(b) Implementation of $Y(A, B, C) = \Pi M(0, 1, 4, 5)$

The inputs of the multiplexer corresponding to the given maxterms are connected to ground and other inputs are connected to V_{CC}. The enable input $\overline{E}$ is connected to ground to enable the chip. The inputs A, B, and C are connected to the select lines of the multiplexer.

Example 4.16 Implement the following Boolean expressions using multiplexer.

(a) $Y = \overline{A}B\overline{C} + A\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}D + \overline{A}CD$

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

Solution

(a) Given Boolean expression is

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

It is a four-variable Boolean expression. It should be converted into standard sum of products form.

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

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

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

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

$\overline{A}\,\overline{B}\,\overline{C}D = 0001 = 1$ $\quad$ $\overline{A}B\overline{C}\,\overline{D} = 0100 = 4$

$A\overline{B}\,\overline{C}D = 1001 = 9$ $\quad$ $\overline{A}\,\overline{B}CD = 0011 = 3$

$\overline{A}B\,CD = 0111 = 7$ $\quad$ $\overline{A}\,B\overline{C}D = 0101 = 5$

The above Boolean expression can be written as

$$Y(A, B, C, D) = \Sigma m(1, 3, 4, 5, 7, 9)$$

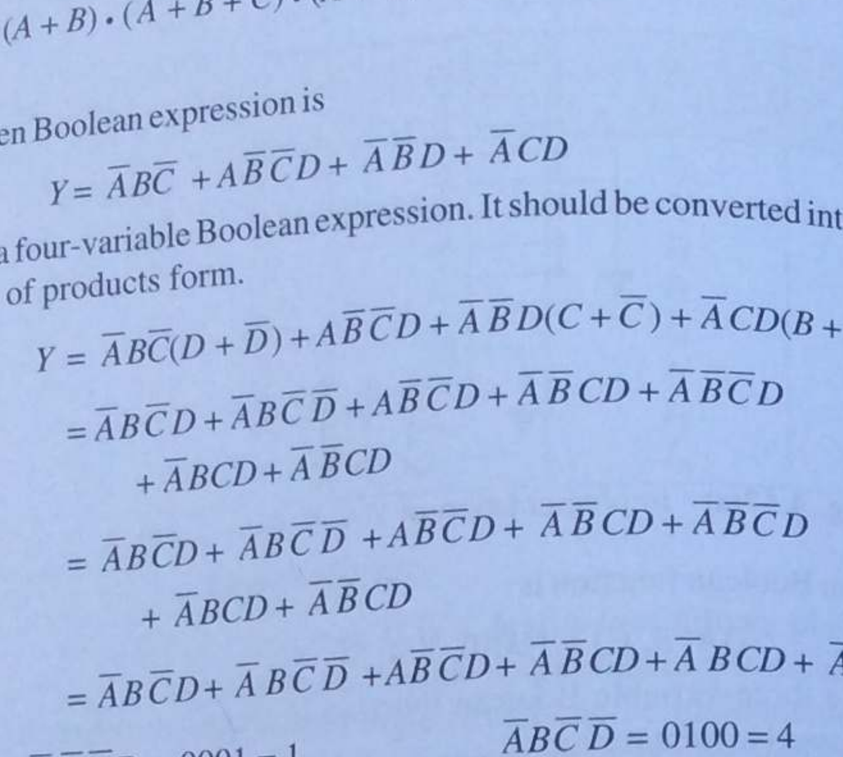


Fig. 4.58(a) Implementation of $Y = \overline{A}B\overline{C} + A\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}D + \overline{A}CD$

It is a four-variable Boolean expression, therefore a multiplexer with four select lines is required to implement the given Boolean function. Figure 4.58(a) shows the implementation using a 16:1 multiplexer. The inputs of the multiplexer corresponding to the given minterms are connected to V_{CC} and other inputs are connected to ground. The enable input $\overline{E}$ is connected to ground to enable the chip. The inputs A, B, C, and D are connected to the select lines of the multiplexer.

(b) Given Boolean expression is

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

It is a three-variable Boolean expression in terms of product of sums form. First, it should be converted into the standard form.

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

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

$A + B + C = 000 = 0$ $\quad$ $A + B + \overline{C} = 001 = 1$

$\overline{A} + B + C = 100 = 4$ $\quad$ $A + \overline{B} + C = 010 = 2$

$A + \overline{B} + \overline{C} = 011 = 3$

The above Boolean expression can be written as

$$Y(A, B, C) = \Pi M(0, 1, 2, 3, 4)$$

It is a three-variable Boolean expression, therefore a multiplexer with three select lines is required to implement the given Boolean function. Figure 4.58(b) shows the implementation using an 8:1 multiplexer.

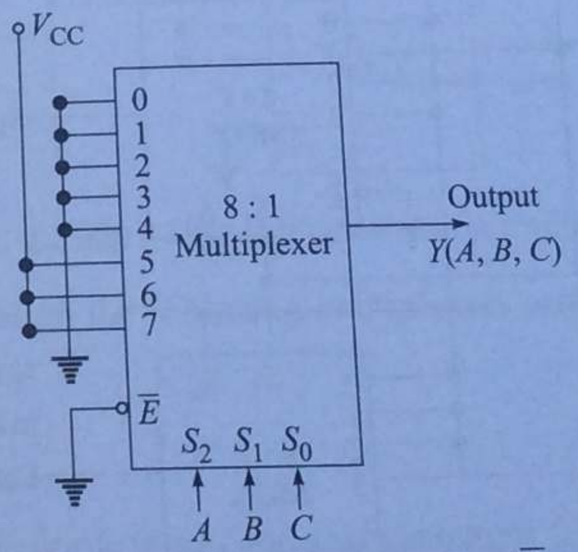


Fig. 4.58(b) Implementation of $Y = (A + B) \cdot (\overline{A} + B + C) \cdot (A + \overline{B})$

The inputs of the multiplexer corresponding to the given maxterms are connected to ground and other inputs are connected to V_{CC}. The enable input $\overline{E}$ is connected to ground to enable the chip. The inputs A, B, and C are connected to the select lines of the multiplexer.

Example 4.17 Implement the following combinational logic circuits using multiplexer:

(a) Half-adder

(b) Half-subtractor

Solution

(a) The half-adder is a combinational logic circuit that adds two 1-bit binary numbers and provides the sum and carry as outputs. The truth table of half-adder is given in Table 4.23(a).

Table 4.23(a) Truth table of half-adder

Inputs		*Outputs*	
A	*B*	*Sum*	*Carry*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

The outputs, sum and carry, are the functions of inputs, A and B. The Boolean expressions can be written as

$\text{Sum}\ (A, B) = \Sigma m\ (1,2)$

$\text{Carry}\ (A, B) = \Sigma m\ (3)$

These can be implemented using two 4:1 multiplexers as shown in Fig. 4.59(a).

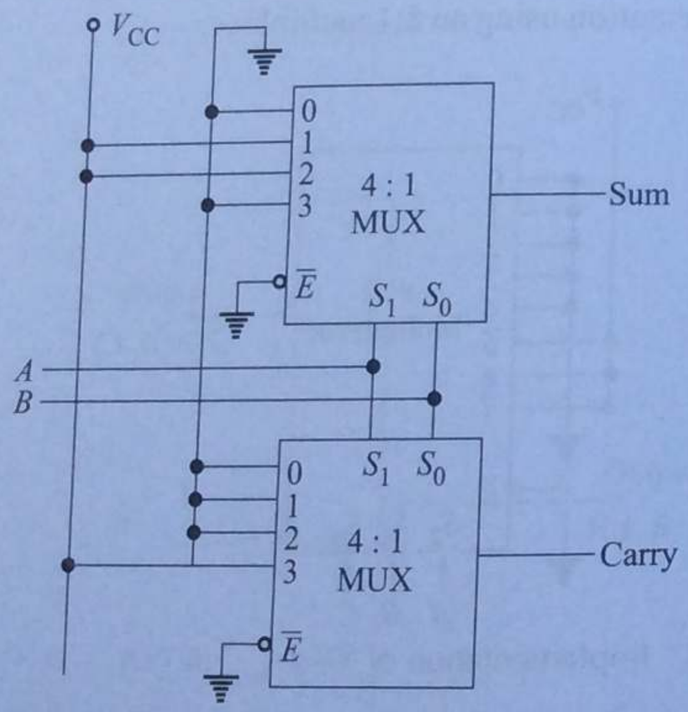


Fig. 4.59(a) Half-adder using multiplexer

(b) The half-subtractor is a combinational logic circuit that subtracts two bits and produces the difference and borrow as outputs. The truth table of half-subtractor is given in Table 4.23(b).

Table 4.23(b) Truth table of half-subtractor

Inputs		*Outputs*	
A	*B*	*D*	*C*
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

The outputs, difference and borrow, are the functions of inputs, A and B. The Boolean expressions can be written as

$D(A, B) = \Sigma m(1, 2)$

$C(A, B) = \Sigma m(1)$

These can be implemented using two 4:1 multiplexers as shown in Fig. 4.59(b).

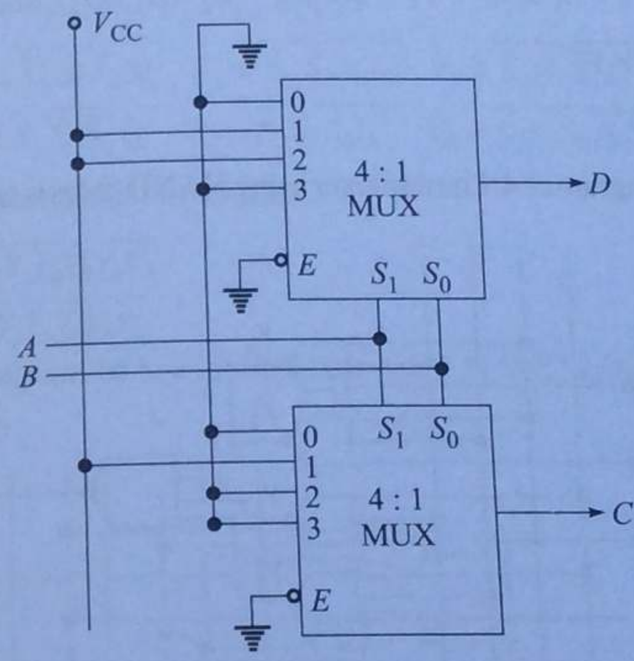


Fig. 4.59(b) Half-subtractor using multiplexer

Example 4.18 Design the following multiplexers using NAND gates only.

(a) 4:1 multiplexer

(b) 8:1 multiplexer

Solution

A multiplexer has multiple inputs and a single output. The selector selects one of the inputs at a time and sends it to the output line. For an n-input multiplexer, m select inputs are required, where $n = 2^m$.

(a) For a 4:1 multiplexer, $n = 4$ and $m = 2$

Let us assume I_0, I_1, I_2, I_3 are the inputs of the multiplexers and S_1 and S_0 are the select inputs. The truth table of 4:1 multiplexer is shown in Table 4.24(a).

Table 4.24(a) Truth table of 4:1 multiplexer

Select Inputs		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

The expression of output Y can be written as

$$Y = \overline{S}_1 \overline{S}_0 I_0 + \overline{S}_1 S_0 I_1 + S_1 \overline{S}_0 I_2 + S_1 S_2 I_3$$

$$\overline{Y} = \overline{\overline{S}_1 \overline{S}_0 I_0 + \overline{S}_1 S_0 I_1 + S_1 \overline{S}_0 I_2 + S_1 S_2 I_3}$$

$$= \overline{\overline{S}_1 \overline{S}_0 I_0} \cdot \overline{\overline{S}_1 S_0 I_1} \cdot \overline{S_1 \overline{S}_0 I_2} \cdot \overline{S_1 S_2 I_3}$$

$$= Y_0 \cdot Y_1 \cdot Y_2 \cdot Y_3$$

where $Y_0 = \overline{\overline{S}_1 \overline{S}_0 I_0}$, $Y_1 = \overline{\overline{S}_1 S_0 I_1}$, $Y_2 = \overline{S_1 \overline{S}_0 I_2}$, and $Y_3 = \overline{S_1 S_0 I_3}$

$$\overline{\overline{Y}} = \overline{Y_0 Y_1 Y_2 Y_3}$$

$$Y = \overline{Y_0 Y_1 Y_2 Y_3}$$

The logic diagram of 4:1 multiplexer using NAND gates is shown in Fig. 4.60(a).

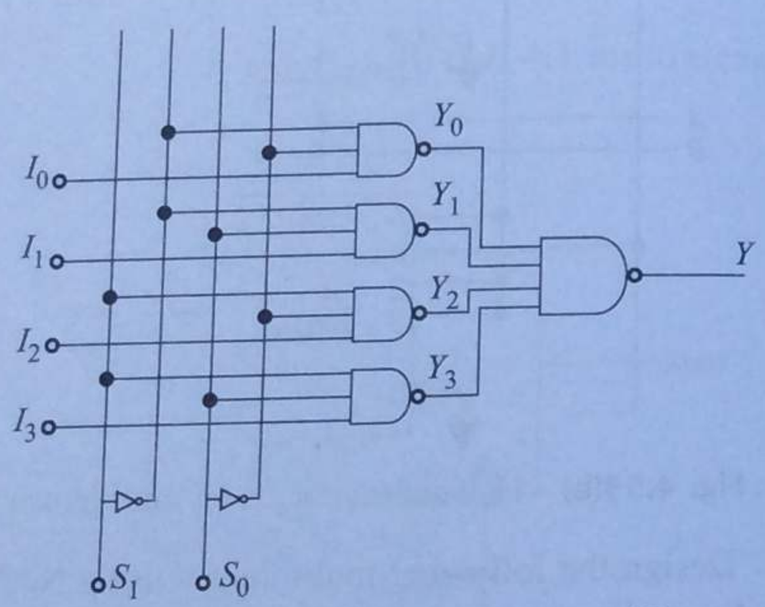


Fig. 4.60(a) 4:1 multiplexer using NAND gates

(b) For an 8:1 multiplexer, $n = 8$ and $m = 3$.

Let us assume $I_0, I_1, I_2, I_3, I_4, I_5, I_6$ and I_7 are the inputs of the multiplexers, and S_2, S_1 and S_0 are the select inputs. The truth table of 8:1 multiplexer is given in Table 4.24 (b).

The expression of output Y can be written as:

$$Y = \overline{S}_2 \overline{S}_1 \overline{S}_0 I_0 + \overline{S}_2 \overline{S}_1 S_0 I_1 + \overline{S}_2 S_1 \overline{S}_0 I_2 + \overline{S}_2 S_1 S_0 I_3 + S_2 \overline{S}_1 \overline{S}_0 I_4 + S_2 \overline{S}_1 S_0 I_5 + S_2 S_1 \overline{S}_0 I_6 + S_2 S_1 S_0 I_7$$

$$\overline{Y} = \overline{\overline{S}_2 \overline{S}_1 \overline{S}_0 I_0 + \overline{S}_2 \overline{S}_1 S_0 I_1 + \overline{S}_2 S_1 \overline{S}_0 I_2 + \overline{S}_2 S_1 S_0 I_3 + S_2 \overline{S}_1 \overline{S}_0 I_4 + S_2 \overline{S}_1 S_0 I_5 + S_2 S_1 \overline{S}_0 I_6 + S_2 S_1 S_0 I_7}$$

Table 4.24(b) Truth table of 8:1 multiplexer

Select Inputs			Output
S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

Using DeMorgan's theorem,

$$\overline{Y} = \overline{(\overline{S}_2 \overline{S}_1 \overline{S}_0 I_0)} \cdot \overline{(\overline{S}_2 \overline{S}_1 S_0 I_1)} \cdot \overline{(\overline{S}_2 S_1 \overline{S}_0 I_2)} \cdot \overline{(\overline{S}_2 S_1 S_0 I_3)} \cdot \overline{(S_2 \overline{S}_1 \overline{S}_0 I_4)} \cdot \overline{(S_2 \overline{S}_1 S_0 I_5)} \cdot \overline{(S_2 S_1 \overline{S}_0 I_6)} \cdot \overline{(S_2 S_1 S_0 I_7)}$$

$$= Y_0 \cdot Y_1 \cdot Y_2 \cdot Y_3 \cdot Y_4 \cdot Y_5 \cdot Y_6 \cdot Y_7$$

where $Y_0 = \overline{\overline{S}_2 \overline{S}_1 \overline{S}_0 I_0}$ $Y_1 = \overline{\overline{S}_2 \overline{S}_1 S_0 I_1}$ $Y_2 = \overline{\overline{S}_2 S_1 \overline{S}_0 I_2}$ $Y_3 = \overline{\overline{S}_2 S_1 S_0 I_3}$

$Y_4 = \overline{S_2 \overline{S}_1 \overline{S}_0 I_4}$ $Y_5 = \overline{S_2 \overline{S}_1 S_0 I_5}$ $Y_6 = \overline{S_2 S_1 \overline{S}_0 I_6}$ $Y_7 = \overline{S_2 S_1 S_0 I_7}$

Taking complement of both sides,

$$\overline{\overline{Y}} = \overline{Y_0 Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 Y_7}$$

$$Y = \overline{Y_0 Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 Y_7}$$

The logic diagram of 8:1 multiplexer using NAND gates only is shown in Fig. 4.60(b).

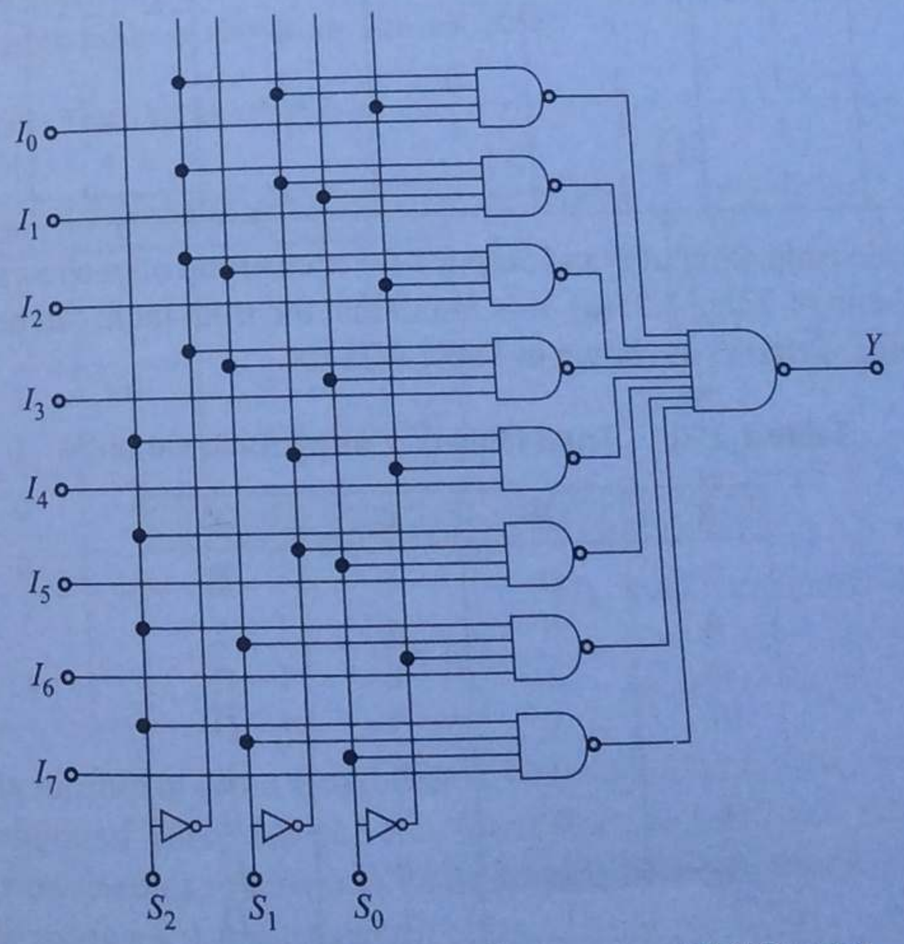


Fig. 4.60(b) 8:1 multiplexer using NAND gates

Example 4.19 Implement the following expression using a single 8:1 multiplexer.

$$Y(A, B, C, D) = \Sigma m(0, 2, 3, 6, 8, 9, 12, 14)$$

Solution

Given Boolean expression is

$$Y(A, B, C, D) = \Sigma m(0, 2, 3, 6, 8, 9, 12, 14)$$

It is a four-variable Boolean expression in terms of minterms. A 16:1 multiplexer is required to implement this expression. But it is also possible to implement the above expression using an 8 :1 multiplexer. The above expression can be written in the form of truth table as shown in Table 4.25 (a).

Table 4.25(a) Truth table of $Y = \Sigma m(0, 2, 3, 6, 8, 9, 12, 14)$

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

The relationship of input D and output Y for each group of two rows is given in the last column of Table 4.25(a). It is found that the truth table can be written in terms of three variables as shown in Table 4.25 (b).

Table 4.25(b) Truth table of Y using three variables

A	B	C	Y
0	0	0	$\overline{D}$
0	0	1	1
0	1	0	0
0	1	1	$\overline{D}$
1	0	0	1
1	0	1	0
1	1	0	$\overline{D}$
1	1	1	$\overline{D}$

The implementation of the given expression using an 8:1 multiplexer is shown in Fig. 4.61.

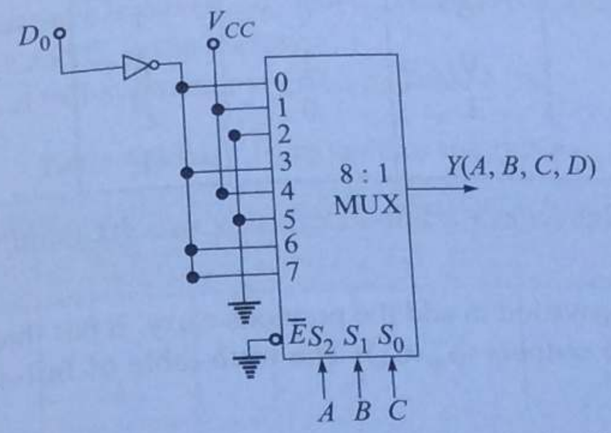


Fig. 4.61 Implementation of $Y = \Sigma m(0, 2, 3, 6, 8, 9, 12, 14)$ using 8:1 multiplexer

Example 4.20 Implement the following expression using a single 4:1 multiplexer.

$$Y(A, B, C) = \Sigma m(0, 1, 4, 6, 7)$$

Solution

Given Boolean expression is

$$Y(A, B, C) = \Sigma m(0, 1, 4, 6, 7)$$

It is a three-variable Boolean expression in terms of minterms. An 8:1 multiplexer is required to implement this expression. But it is also possible to implement the above expression using a 4:1 multiplexer. The above expression can be written in the form of truth table as shown in Table 4.26(a).

Table 4.26(a) Truth table of $Y(A, B, C) = \Sigma m(0, 1, 4, 6, 7)$

A	B	C	Y	
0	0	0	1	
0	0	1	1	$Y = 1$
0	1	0	0	
0	1	1	0	$Y = 0$
1	0	0	1	
1	0	1	0	$Y = \overline{C}$
1	1	0	1	
1	1	1	1	$Y = 1$

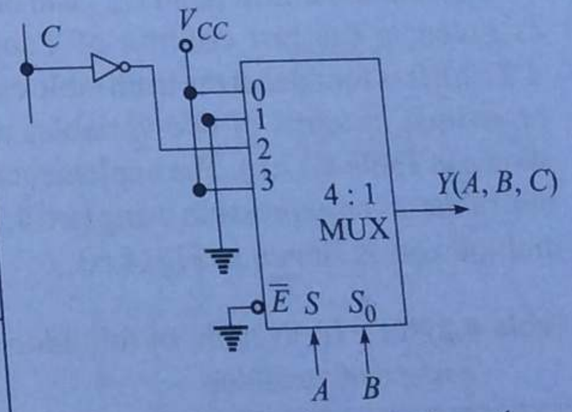


Fig. 4.62 Implementation of $Y(A, B, C) = \Sigma m(0, 1, 4, 6, 7)$ using 4:1 multiplexer

The relationship of input C and output Y for each group of two rows is given in the last column of Table 4.26(a). It is found that the truth table can be written in terms of two variables as shown in Table 4.26(b). The implementation of the given expression using a 4:1 multiplexer is shown in Fig. 4.62.

Table 4.26(b) Truth table of Y using three variables

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

Example 4.21 Implement a full-adder using two 4:1 multiplexers.

Solution

A full-adder has provision to add the previous carry. It has three inputs (A_n, B_n and C_{n-1}) and two outputs (S_n, C_n). The truth table of full-adder is given in Table 4.27 (a).

Table 4.27(a) Truth table of full-adder

Inputs			*Outputs*			
			S_n		C_n	
A_n	B_n	C_{n-1}				
0	0	0	0	C_{n-1}	0	0
0	0	1	1		0	
0	1	0	1	$\overline{C}_{n-1}$	0	C_{n-1}
0	1	1	0		1	
1	0	0	1	$\overline{C}_{n-1}$	0	C_{n-1}
1	0	1	0		1	
1	1	0	0	C_{n-1}	1	1
1	1	1	1		1	

The relationship of input C_{n-1} and outputs S_n and C_n for each group of two rows is given in the last column of Table 4.27(a). It is found that the truth table can be written in terms of two variables as shown in Table 4.27(b). The implementation of the given expression using two 4:1 multiplexers is shown in Fig. 4.63.

Table 4.27(b) Truth table of full-adder using two variables

A_n	B_n	S_n	C_n
0	0	C_{n-1}	0
0	1	$\overline{C}_{n-1}$	C_{n-1}
1	0	$\overline{C}_{n-1}$	C_{n-1}
1	1	C_{n-1}	1

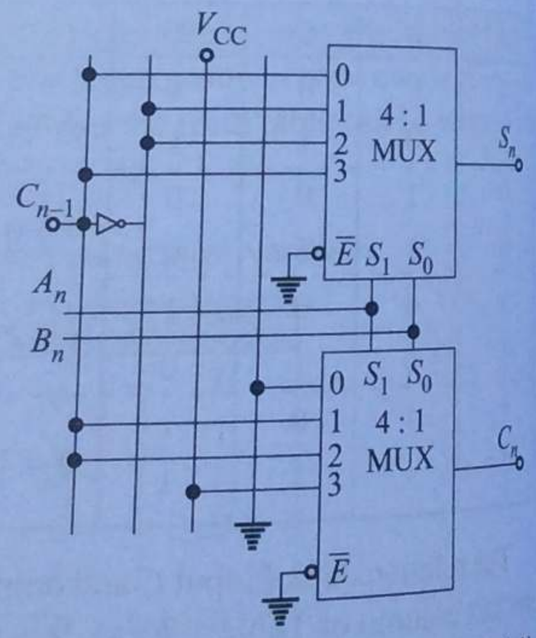


Fig. 4.63 Implementation of full-adder using 4:1 multiplexer

Example 4.21 Implement a full-subtractor using two 4:1 multiplexers.

Solution

A full-subtractor has three inputs (A_n, B_n, and C_{n-1}) and two outputs (D_n, C_n), where D_n is the difference and C_n is the borrow.

The truth table of full-subtractor is given in Table 4.28.

Table 4.28(a) Truth table of subtractor

Inputs			*Outputs*			
			D_n		C_n	
A_n	B_n	C_{n-1}				
0	0	0	0	C_{n-1}	0	C_{n-1}
0	0	1	1		1	
0	1	0	1	$\overline{C}_{n-1}$	1	1
0	1	1	0		1	
1	0	0	1	$\overline{C}_{n-1}$	0	0
1	0	1	0		0	
1	1	0	0	C_{n-1}	0	C_{n-1}
1	1	1	1		1	

The relationship of input C_{n-1} and outputs D_n and C_n for each group of two rows is given in Table 4.28(a). It is found that the truth table can be written in terms of two variables as shown in Table 4.28(b). Figure 4.64 shows the implementation of a full-subtractor using two 4:1 multiplexers.

Table 4.28(b) Truth table of subtractor using two variables

A_n	B_n	S_n	C_n
0	0	C_{n-1}	C_{n-1}
0	1	$\overline{C}_{n-1}$	1
1	0	$\overline{C}_{n-1}$	0
1	1	C_{n-1}	C_{n-1}

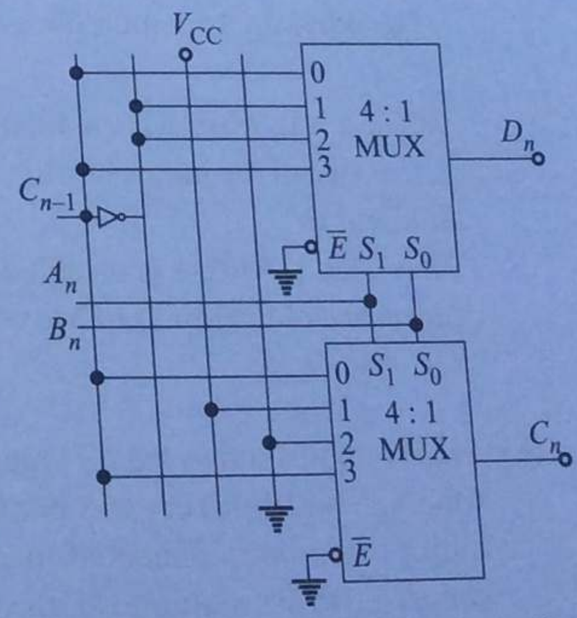


Fig. 4.64 Implementation of full-subtractor using 4:1 multiplexer

Example 4.23 Design a 16:1 multiplexer using

(a) 8:1 multiplexer and OR gate

(b) 4:1 multiplexer

Solution

(a) Figure 4.65 shows the 16:1 multiplexer using two 8:1 multiplexers and an OR gate. Here, the 8:1 multiplexers are connected in series. The enable input of multiplexer is used as the select input A.

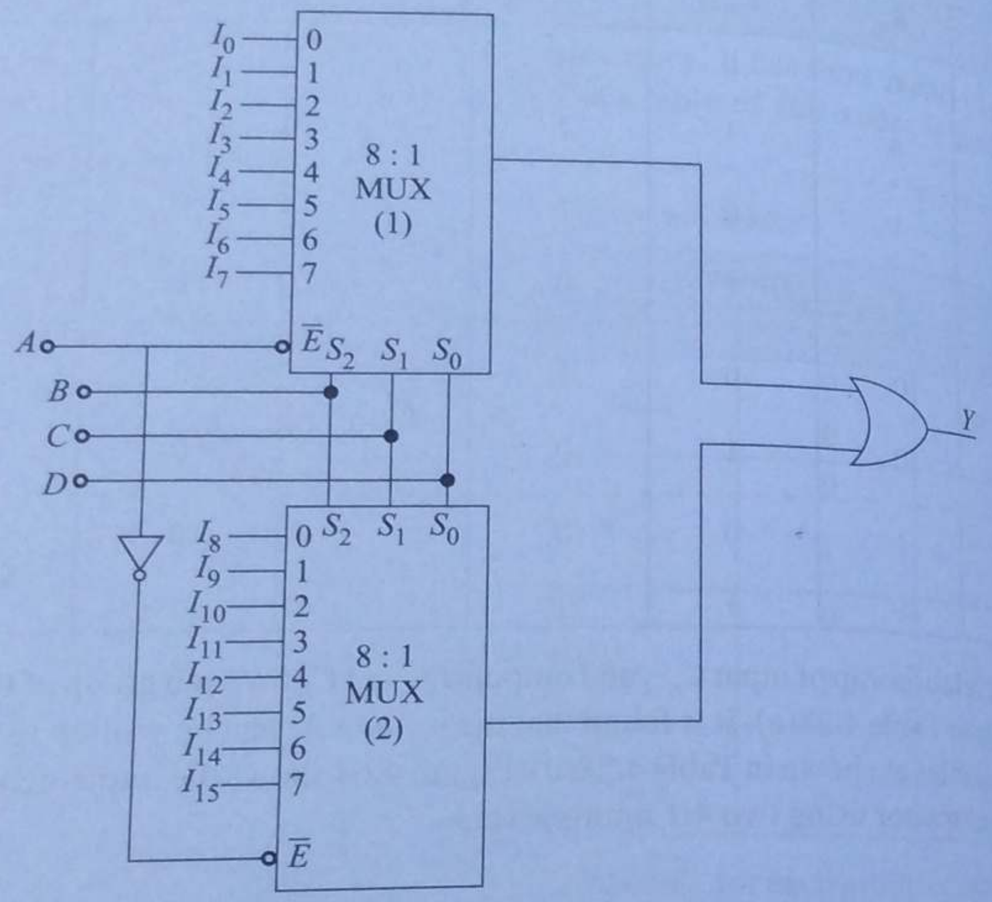


Fig. 4.65(a) 16:1 multiplexer using 8:1 multiplexer and OR gate

When $A = 0$, Mux-1 is enabled and Mux-2 is disabled. The output of OR gate is one of the inputs of Mux-1 according to the code at the select inputs B, C and D.

When $A = 1$, Mux-1 is disabled and Mux-2 is enabled. The output of OR gate is one of the inputs of Mux-2 according to the code at the select inputs B, C, and D.

(b) Figure 4.65(b) shows the 16:1 multiplexer using two 4:1 multiplexers. Here, four 4:1 multiplexers are connected in series. The enable inputs of multiplexers are connected to ground. The output of multiplexers are connected to 4:1 multiplexer; the select inputs of this multiplexers are used as the select inputs A and B.

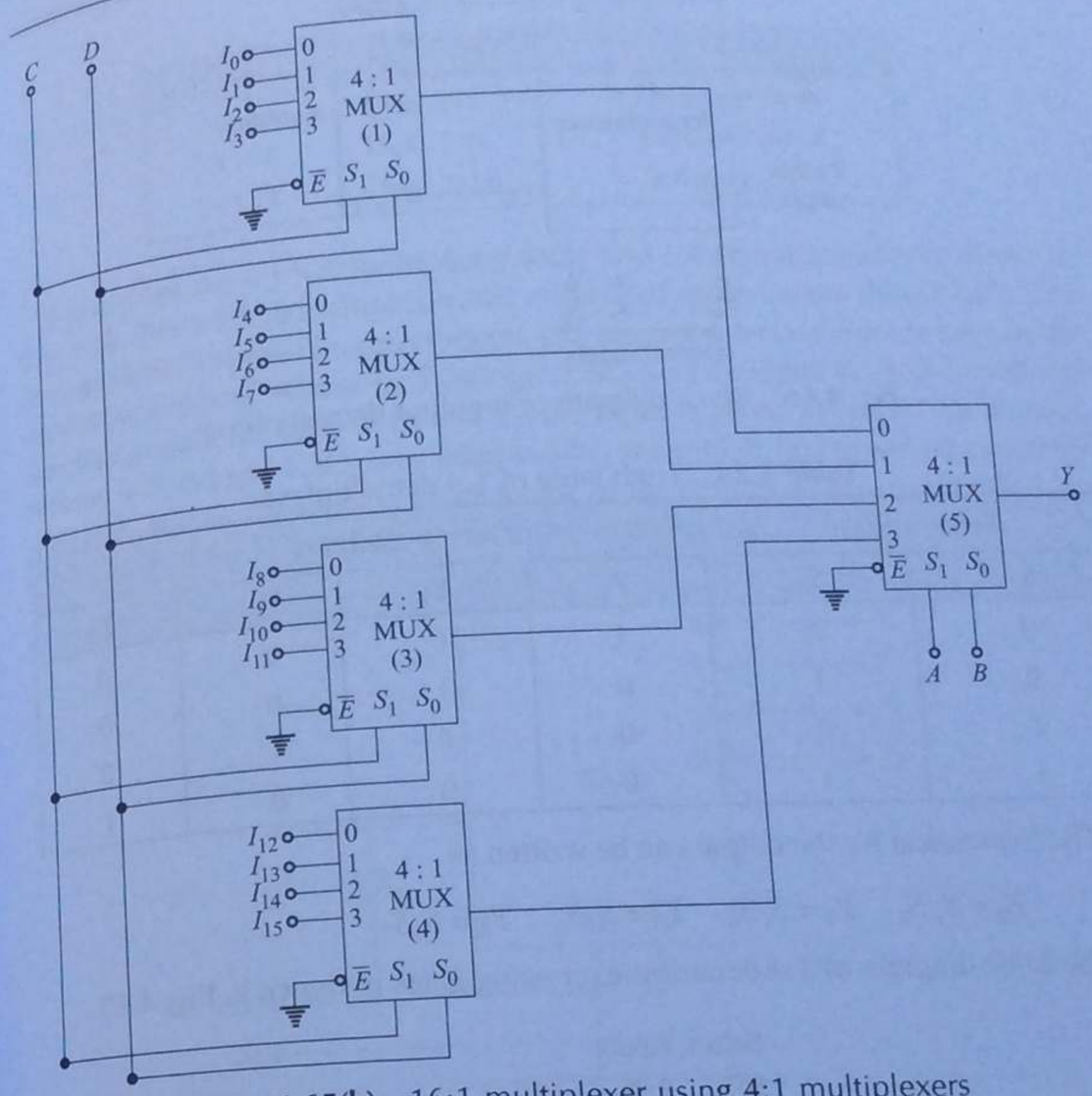


Fig. 4.65(b) 16:1 multiplexer using 4:1 multiplexers

4.13 DEMULTIPLEXER

The demultiplexer is a combinational logic circuit having a single input and many outputs. It performs the reverse operation of a multiplexer. It accepts a single input and sends it to one of the output lines, according to the select inputs. For an n-output demultiplexer, the number of select lines is m, where $n = 2^m$. The block diagram of an n-output demultiplexer is shown in Fig. 4.66.

When binary inputs are applied at the select input lines and input lines are connected to logic 1 level, then the demultiplexer works as a binary-to-decimal converter.

Let us consider a 1:4 demultiplexer as an example. For a 4-output demultiplexer, the number of select inputs is $2(2^2 = 4)$. The truth table of 1:4 demultiplexer is given in Table 4.29.

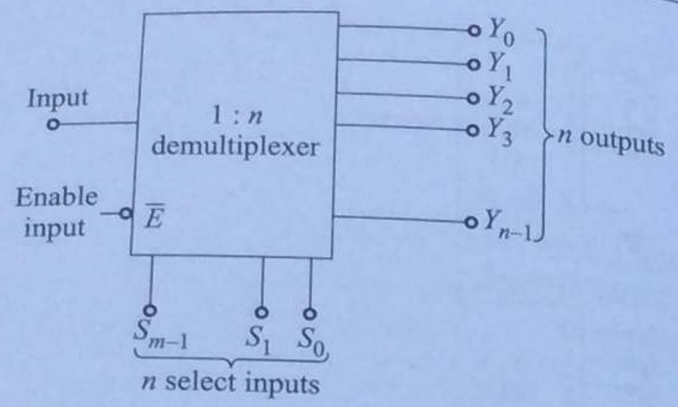


Fig. 4.66 Block diagram of *n*-output demultiplexer

Table 4.29 Truth table of 1:4 demultiplexer

Select inputs		*Outputs*			
S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

The expression for the output can be written as

$$Y_0 = \overline{S}_1 \overline{S}_0 \quad Y_1 = \overline{S}_1 S_0 \quad Y_2 = S_1 \overline{S}_0 \quad Y_3 = S_1 S_0$$

The logic diagram of 1:4 demultiplexer using gates is shown in Fig. 4.67.

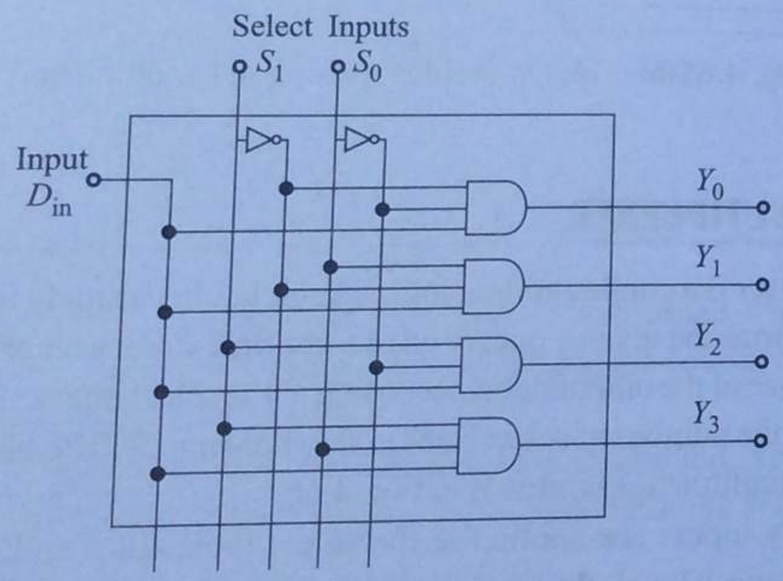


Fig. 4.67 Logic diagram of 1:4 demultiplexer using gates

4.13.1 Demultiplexer Tree

The 1:4, 1:8, and 1:16 demultiplexers are available in the form of ICs as given in Table 4.30.

The largest available ICs of demultiplexers are 1:16. Therefore, to implement a demultiplexer with more than 16 outputs, there should be provision for expansion. The demultiplexer has an enable (strobe) input which helps in its expansion.

Table 4.30 Demultiplexers

IC No.	*Description*	*Output*
74139	Dual 1:4	Inverted input
74138	Dual 1:8	Inverted input
74154	Dual 1:16	Same as input

Let us consider a 1:8 demultiplexer using two 1:4 demultiplexers as shown in Fig. 4.68. For a 1:8 demultiplexer, the number of select inputs should be 3. The demultiplexer 1:4 has two select inputs and hence the enable input is used as the third select input S_2. Whenever S_2 is logic 0, IC_1 is enabled and IC_2 is disabled; and depending upon the code at S_1 and S_0, the input is given to one of the outputs. Whenever S_2 is logic 1, IC_2 is enabled and IC_1 is disabled; and depending upon the code at S_1 and S_0, the input is given to one of the outputs as given in Table 4.31. The bubble on the output line shows that it is active low.

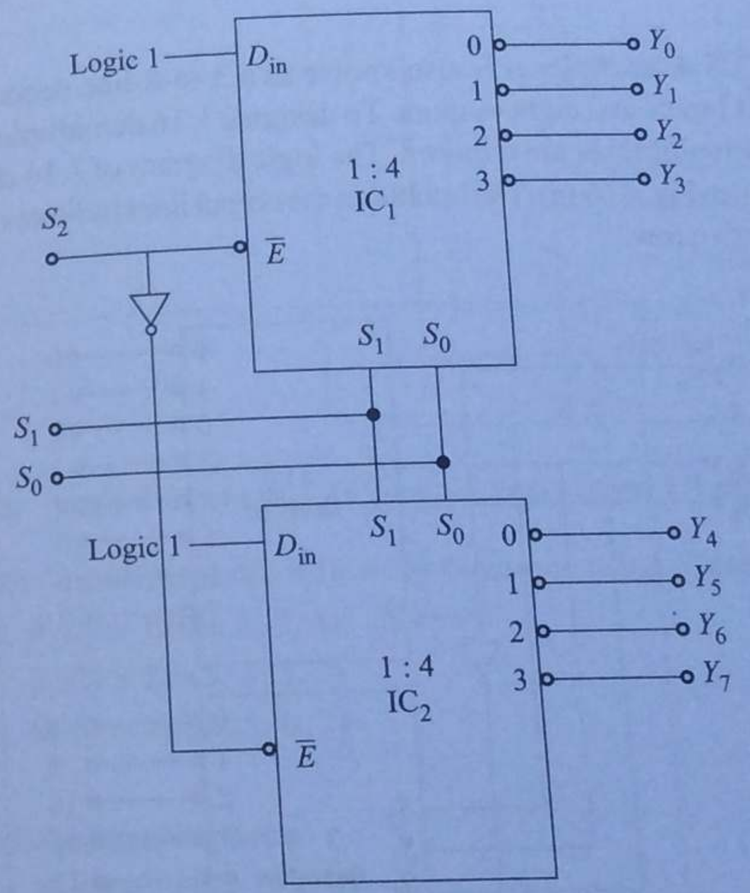


Fig. 4.68 1:8 demultiplexer using 1:4 demultiplex

Table 4.31 Truth table of 1:8 demultiplexer using two 1:4 demultiplexers

Select inputs			*Enable*	*Disable*	*Outputs*							
S_2	S_1	S_0			Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	IC_1	IC_2	1	1	1	1	1	1	1	0
0	0	1	IC_1	IC_2	1	1	1	1	1	1	0	1
0	1	0	IC_1	IC_2	1	1	1	1	1	0	1	1

Contd.

Contd.

Select inputs			Enable	Disable	Outputs							
S_2	S_1	S_0			Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	1	1	IC_1	IC_2	1	1	1	1	0	1	1	1
1	0	0	IC_2	IC_1	1	1	1	0	1	1	1	1
1	0	1	IC_2	IC_1	1	1	0	1	1	1	1	1
1	1	0	IC_2	IC_1	1	0	1	1	1	1	1	1
1	1	1	IC_2	IC_1	0	1	1	1	1	1	1	1

Example 4.24 Design a 1:16 demultiplexer using

(a) 1:8 demultiplexer

(b) 1:4 demultiplexer

Solution

(a) The 1:8 demultiplexer is also known as a 3-to-8-line decoder. It has three select inputs and eight outputs. To design a 1:16 demultiplexer, two ICs of 1:8 demultiplexer are required. The logic diagram of 1:16 demultiplexer is shown in Fig. 4.69 (a). The bubble on the output lines indicates that the outputs are active low.

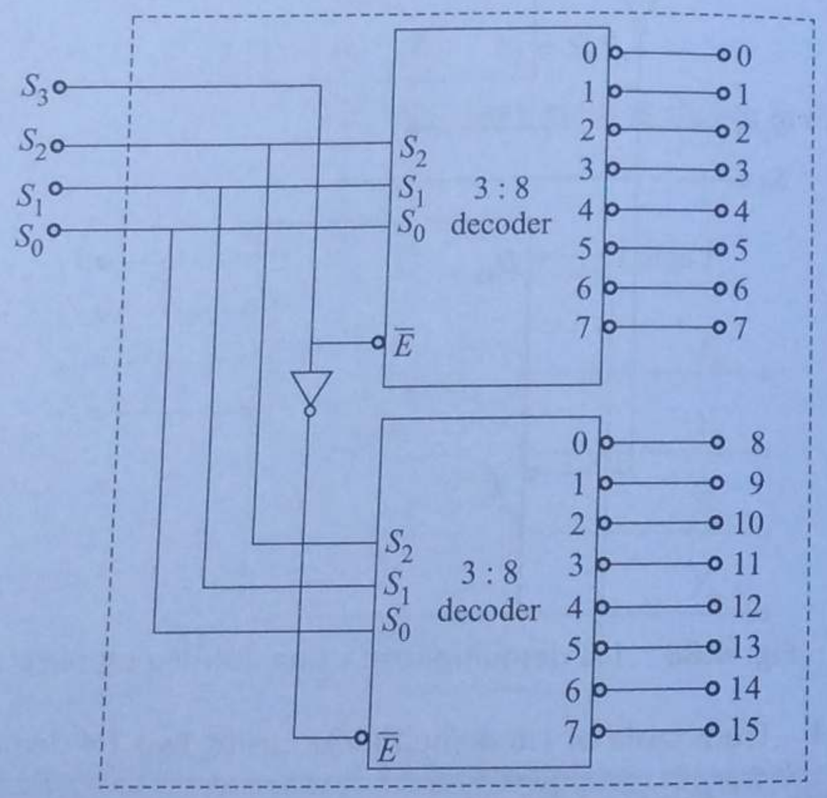


Fig. 4.69(a) Logic diagram of 1:16 demultiplexer using 1:8 demultiplexer

(b) The 4-to-16-line decoder can be designed using 2-to-4-line decoders as shown in Fig. 4.69(b).

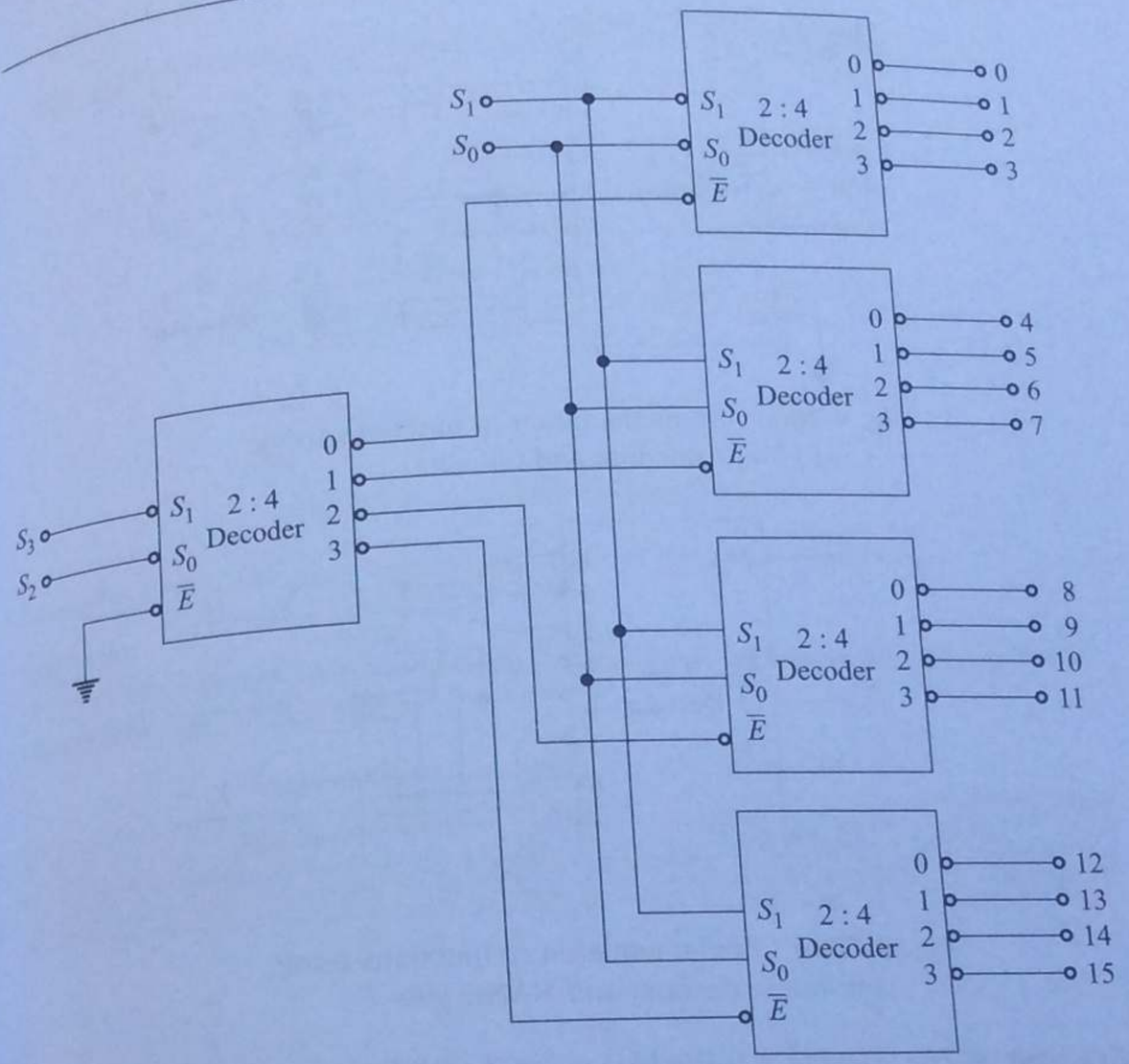


Fig. 4.69(b) Logic diagram of 1:16 demultiplexer using 1:4 demultiplexer

Example 4.25 Implement the following functions using 3-to-8-line decoder.

$$Y_0(A, B, C) = \Sigma m(0, 1, 2, 4)$$
$$Y_1(A, B, C) = \Sigma m(1, 3, 5, 7)$$
$$Y_2(A, B, C) = \Sigma m(4, 5, 6, 7)$$

Solution

Given Boolean functions are

$$Y_0(A, B, C) = \Sigma m(0, 1, 2, 4)$$
$$Y_1(A, B, C) = \Sigma m(1, 3, 5, 7)$$
$$Y_2(A, B, C) = \Sigma m(4, 5, 6, 7)$$

This multi-output combinational circuit can be implemented using 3-to-8-line decoder. The expressions of Y_0, Y_1, and Y_2 are in the form of minterms. The OR gates are used at the outputs to OR the defined minterms as shown in Fig. 4.70 (a).

The above circuit diagram is correct under the condition that the outputs of the decoder are active high. But usually, decoder outputs are active low, therefore the outputs are inverted and given to the OR gate. We have seen that inverted input OR gates are replaced by NAND gates. Figure 4.70(b) shows the logic diagram to implement the functions defined by the previous expression with NAND gates.

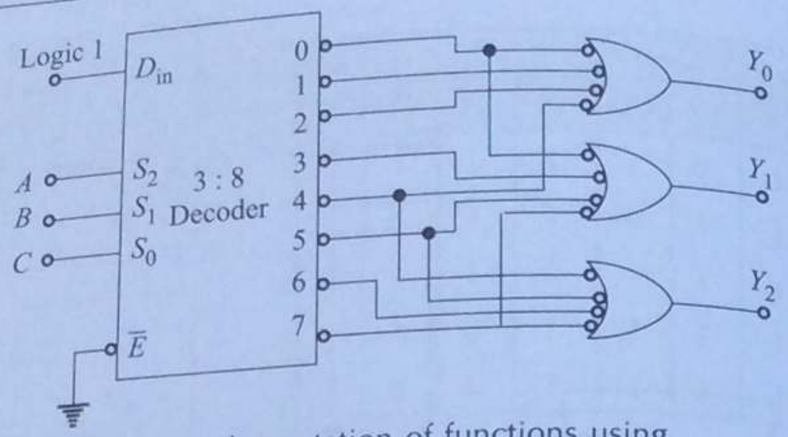


Fig. 4.70(a) Implementation of functions using 3-to-8-line decoder and OR gate

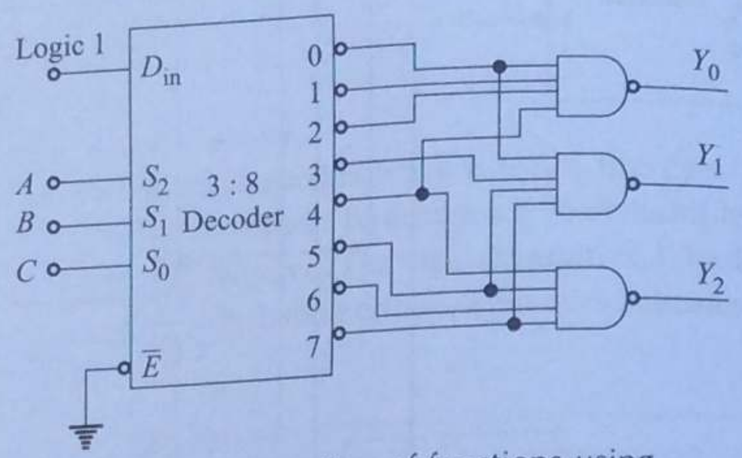


Fig. 4.70(b) Implementation of functions using 3-to-8-line decoder and NAND gate

Example 4.26 Design a full-adder using 3-to-8-line decoder.

Solution

The full-adder has three inputs and two outputs. The truth table of full-adder is given in Table 4.32. When the inputs are 001, 010, 100, 111, then $S_n = 1$; and when the inputs are 011, 101, 110, and 111, then $C_n = 1$. Therefore the expressions for S_n and C_n can be written as

$$S_n(A_n, B_n, C_{n-1}) = \Sigma m(1, 2, 4, 7)$$

$$C_n(A_n, B_n, C_{n-1}) = \Sigma m(3, 5, 6, 7)$$

Table 4.32 Truth table of full-adder

Inputs			*Outputs*	
A_n	B_n	C_{n-1}	S_n	C_n
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

The implementation of full-adder using decoder is shown in Fig. 4.71.

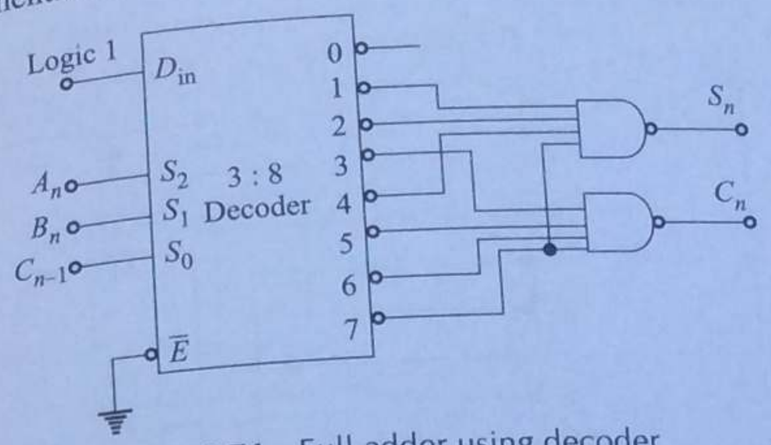


Fig. 4.71 Full-adder using decoder

Example 4.26 Design a 2-bit comparator using suitable decoder.

Solution

A 2-bit binary comparator has four inputs and three outputs. The truth table of a comparator is given in Table 4.33.

Table 4.33 Truth table of 2-bit comparator

Inputs				*Outputs*		
A_1	A_0	B_1	B_0	$A > B$	$A = B$	$A < B$
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

The expression for output can be written as

$$A > B(A_0, A_0, B_1, B_0) = \Sigma m(4, 8, 9, 12, 13, 14)$$

$$A = B(A_1, A_0, B_1, B_0) = \Sigma m(0, 5, 10, 15)$$

$$A < B(A_1, A_0, B_1, B_0) = \Sigma m(1, 2, 3, 6, 7, 11)$$

The implementation of a 2-bit comparator using a 4:16 decoder is shown in Fig. 4.72.

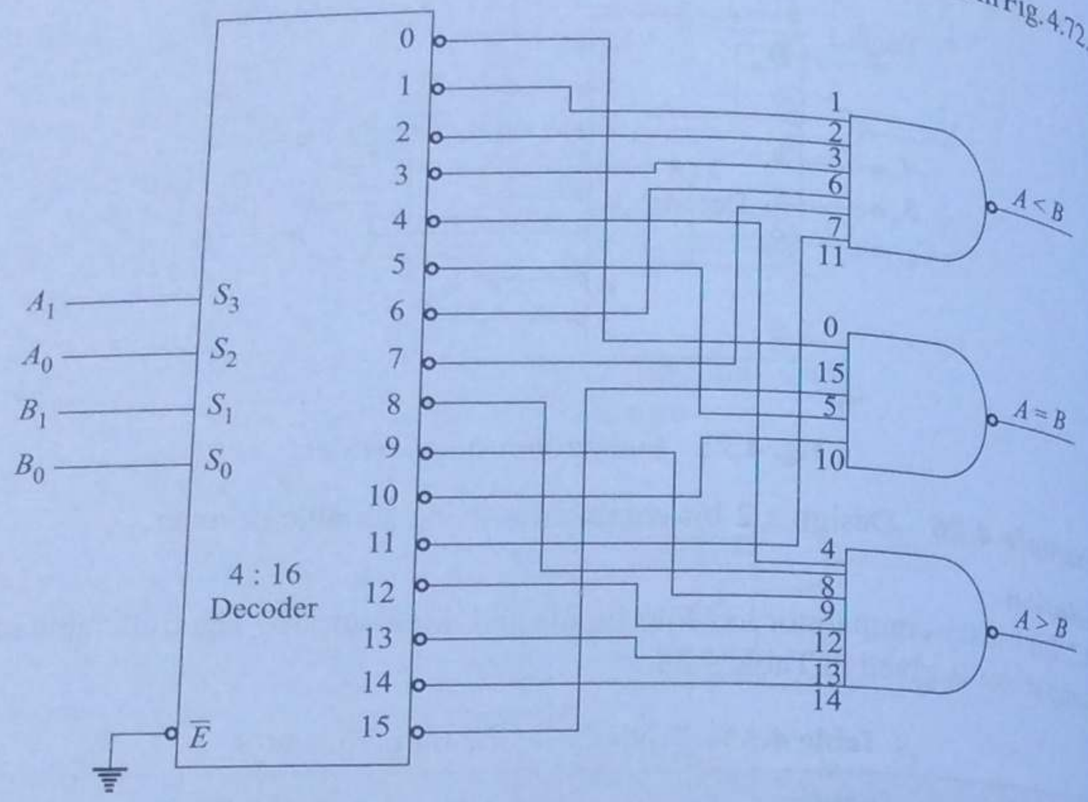


Fig. 4.72 2-bit comparator using 4:16 decoder

4.14 CODE CONVERTERS

Code is the symbolic representation of information in a particular format. The information may include numbers, alphabets, and symbols, which men and machine can recognize. The basic property of machine code is its binary nature. The binary representations of hexadecimal, octal, decimal are known as hexadecimal code, octal code, and decimal code, respectively. The codes are used to store and transmit the data efficiently. There is a wide variety of binary codes used in digital systems. Examples of binary codes are:

1. Binary Coded Decimal (BCD)
2. Excess-3 code
3. Gray code

In most of the applications, it is necessary to convert the code from one form to another. This section explains the various code converters and their design.

4.14.1 Binary-to-BCD Converter

BCD is a 4-bit binary representation of decimal number. The 4-bit binary code is converted into 5-bit BCD. The truth table of binary-to-BCD converter is given in Table 4.34.

Table 4.34 4-bit Binary and its equivalent BCD

Binary				BCD code				
A	B	C	D	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0	1
0	1	1	0	0	0	1	1	0
0	1	1	1	0	0	1	1	1
0	1	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	1	0	0	0	0
1	0	1	1	1	0	0	0	1
1	0	0	0	1	0	0	1	0
1	1	0	1	1	0	0	1	1
1	1	1	0	1	0	1	0	0
1	1	1	1	1	0	1	0	1

The K-map for BCD B_4, B_3, B_2, B_1 and B_0 with inputs A, B, C, D are:

K-map for B_4

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

$B_4 = AB + AC$

K-map for B_1

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

$B_1 = \overline{A}C + AB\overline{C}$

K-map for B_2

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

$B_2 = \overline{A}B + BC$

K-map for B_3

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

$B_3 = A\overline{B}\,\overline{C}$

K-map for B_0

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

$B_0 = D$

The logic diagram of binary-to-BCD code converter is shown in Fig. 4.73.

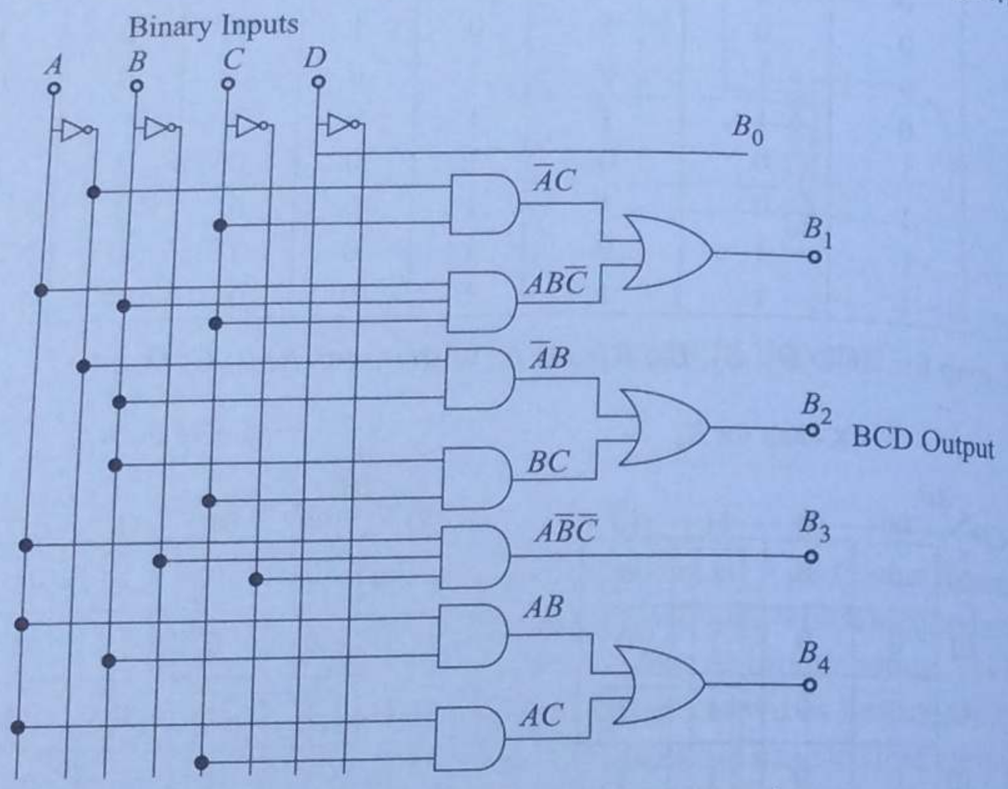


Fig. 4.73 Logic diagram of binary-to-BCD code converter

IC 74185

It is a 6-bit binary-to-BCD converter. The block diagram of binary-to-BCD converter is shown in Fig. 4.74.

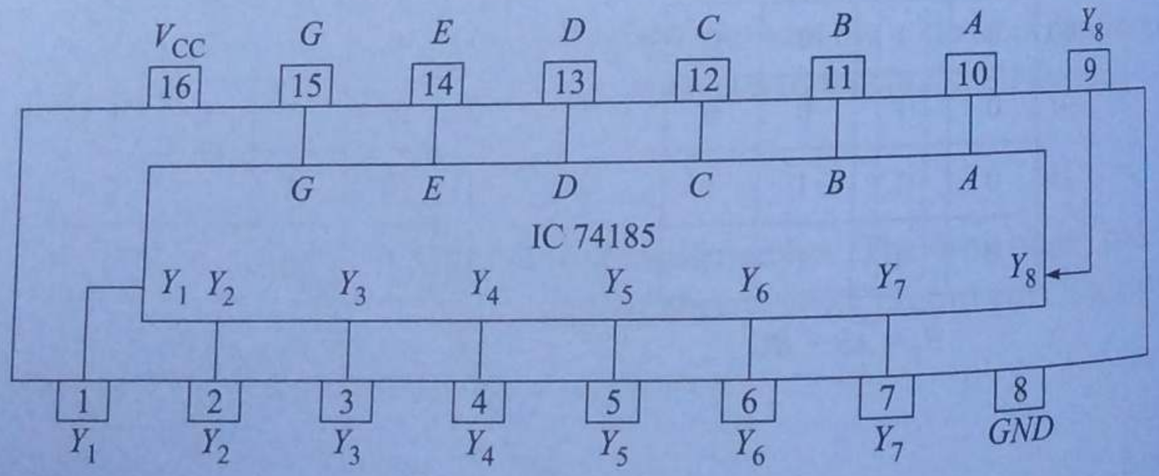


Fig. 4.74 Binary to BCD converter (IC 74185A)

The truth table of IC 74185 binary-to-BCD converter is given in Table 4.35.

Table 4.35 Truth table of IC 74185A

Binary words	Inputs						Outputs					
	E	*D*	*C*	*B*	*A*	*G*	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1
0 – 1	0	0	0	0	0	0	0	0	0	0	0	0
2 – 3	0	0	0	0	1	0	0	0	0	0	0	1
4 – 5	0	0	0	1	0	0	0	0	0	0	1	0
6 – 7	0	0	0	1	1	0	0	0	0	0	1	1
8 – 9	0	0	1	0	0	0	0	0	0	1	0	0
10 – 11	0	0	1	0	1	0	0	0	1	0	0	0
12 – 13	0	0	1	1	0	0	0	0	1	0	0	1
14 – 15	0	0	1	1	1	0	0	0	1	0	1	0
16 – 17	0	1	0	0	0	0	0	0	1	0	1	1
18 – 19	0	1	0	0	1	0	0	0	1	1	0	0
20 – 21	0	1	0	1	0	0	0	1	0	0	0	0
22 – 23	0	1	0	1	1	0	0	1	0	0	0	1
24 – 25	0	1	1	0	0	0	0	1	0	0	1	0
26 – 27	0	1	1	0	1	0	0	1	0	0	1	1
28 – 29	0	1	1	1	0	0	0	1	0	1	0	0
30 – 31	0	1	1	1	1	0	0	1	1	0	0	0
32 – 33	1	0	0	0	0	0	0	1	1	0	0	1
34 – 35	1	0	0	0	1	0	0	1	1	0	1	0
36 – 37	1	0	0	1	0	0	0	1	1	0	1	1
38 – 39	1	0	0	1	1	0	0	0	1	1	0	0
40 – 41	1	0	1	0	0	0	1	0	0	0	0	0
42 – 43	1	0	1	0	1	0	1	0	0	0	0	1
44 – 45	1	0	1	1	0	0	1	0	0	0	1	0
46 – 47	1	0	1	1	1	0	1	0	0	0	1	1
48 – 49	1	1	0	0	0	0	1	0	0	1	0	0
50 – 51	1	1	0	0	1	0	1	0	1	0	0	0
52 – 53	1	1	0	1	0	0	1	0	1	0	0	1
54 – 55	1	1	0	1	1	0	1	0	1	0	1	0
56 – 57	1	1	1	0	0	0	1	0	1	0	1	1
58 – 59	1	1	1	0	1	0	1	0	1	1	0	0
60 – 61	1	1	1	1	0	0	1	1	0	0	0	0
62 – 63	1	1	1	1	1	0	1	1	0	0	0	1
ALL	X	X	X	X	X	1	1	1	1	1	1	1

4.14.2 BCD-to-Binary Converter

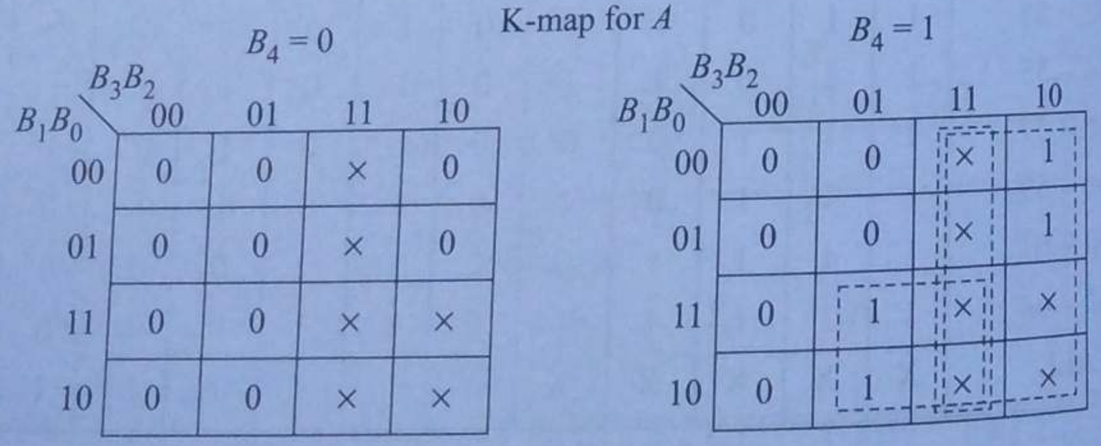
BCD is a 4-bit binary representation of decimal numbers. The 4-bit BCD code is converted into 4-bit binary code, and the 5-bit BCD code is converted into 5-bit binary code. The truth table of 5-bit BCD-to-binary converter is given in Table 4.36.

Table 4.36 5-bit BCD and its equivalent Binary

BCD Inputs					*Outputs*				
B_4	B_3	B_2	B_1	B_0	*A*	*B*	*C*	*D*	*E*
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	0	0	1	0
0	0	0	1	1	0	0	0	1	1
0	0	1	0	0	0	0	1	0	0
0	0	1	0	1	0	0	1	0	1
0	0	1	1	0	0	0	1	1	0
0	0	1	1	1	0	0	1	1	1
0	1	0	0	0	0	1	0	0	0
0	1	0	0	1	0	1	0	0	1
1	0	0	0	0	0	1	0	1	0
1	0	0	0	1	0	1	0	1	1
1	0	0	1	0	0	1	1	0	0
1	0	0	1	1	0	1	1	0	1
1	0	1	0	0	0	1	1	1	0
1	0	1	0	1	0	1	1	1	1
1	0	1	1	0	1	0	0	0	0
1	0	1	1	1	1	0	0	0	1
1	1	0	0	0	1	0	0	1	0
1	1	0	0	1	1	0	0	1	1

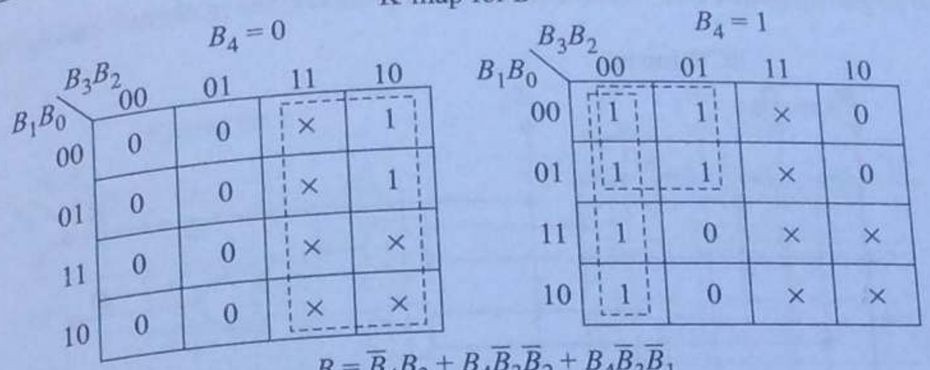
The K-map for binary *A*, *B*, *C*, *D*, and *E* with BCD inputs B_4, B_3, B_2, B_1, and B_0 are:

K-map for *A*



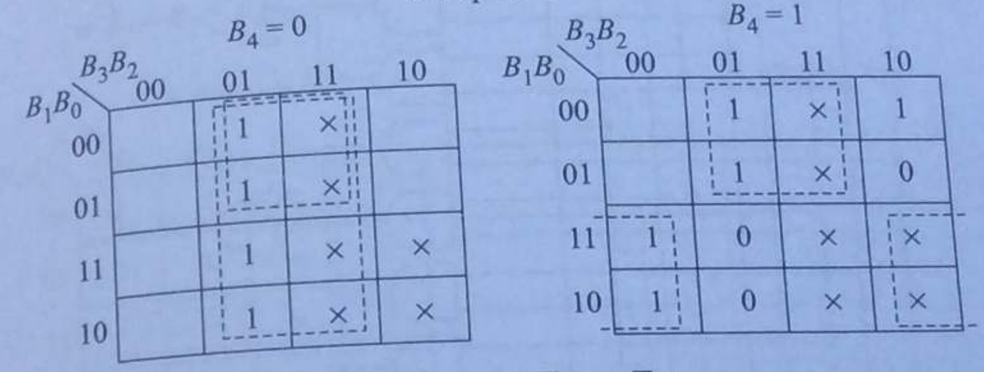
$$A = B_4B_2B_1 + B_4B_3$$

K-map for *B*



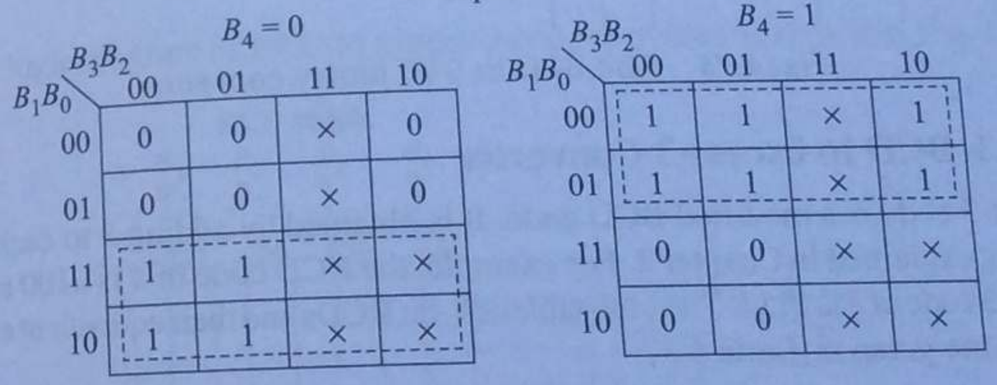
$$B = \overline{B}_4B_3 + B_4\overline{B}_3\overline{B}_2 + B_4\overline{B}_3\overline{B}_1$$

K-map for *C*



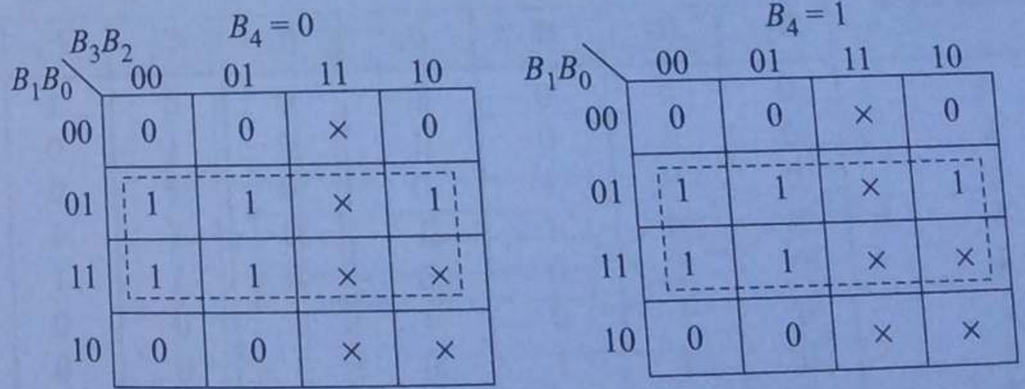
$$C = \overline{B}_4B_2 + B_2\overline{B}_1 + B_4\overline{B}_2B_1$$

K-map for *D*



$$D = \overline{B}_4B_1 + B_4\overline{B}_1$$

K-map for *E*



$$E = B_0$$

The logic diagram of 5-bit BCD-to-binary converter is shown in Fig. 4.75.

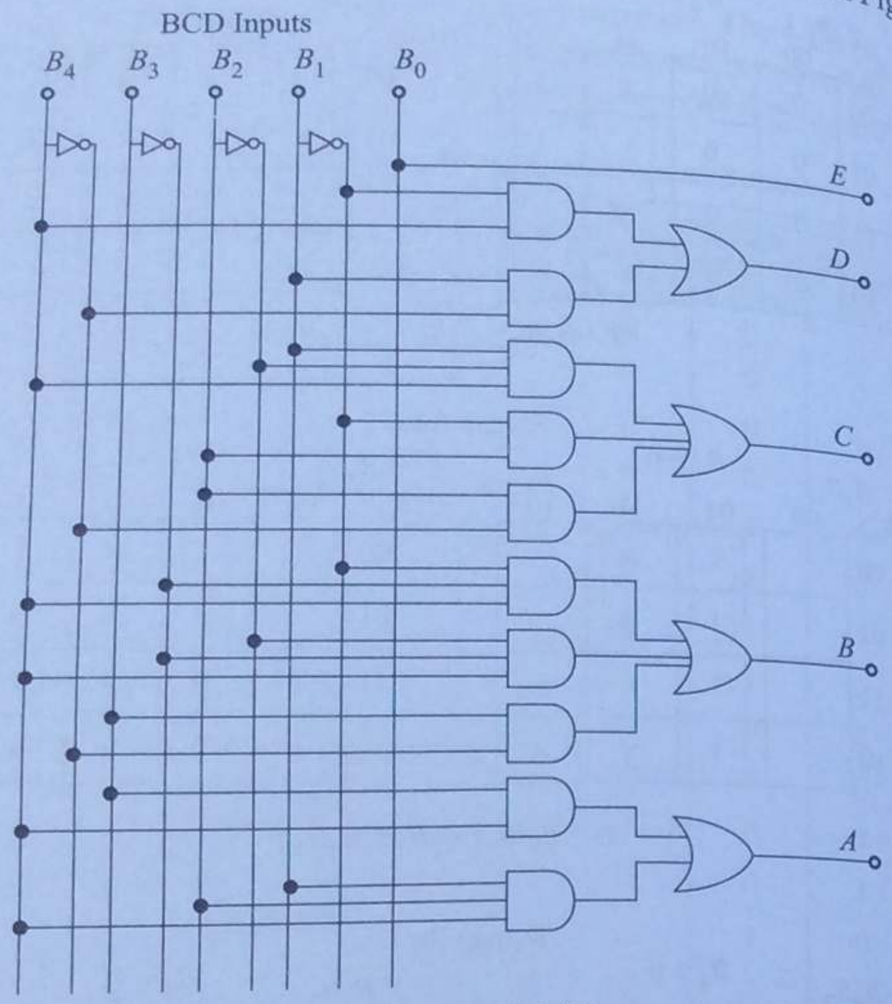


Fig. 4.75 5-bit BCD to 5-bit binary converter

4.14.3 BCD to Excess-3 Converter

Excess-3 code is a modified BCD code. It is obtained by adding 3 to each BCD code, as explained in Chapter 2. For example, the BCD code of 4 is 0100 and the excess-3 code of 4 is 0111. The truth table of 4-bit BCDs and their equivalent excess-3 codes are given in Table 4.37.

Table 4.37 4-bit BCD and its equivalent excess-3 codes

Decimal Digit	*BCD Code*				*Excess-3 code*			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

The K-maps for excess-3 codes E_3, E_2, E_1, and E_0 with BCD inputs B_3, B_2, B_1, and B_0 are:

K-map for E_3

B_1B_0 \ B_3B_2	00	01	11	10
00	0	0	×	1
01	0	1	×	1
11	0	1	×	×
10	0	1	×	×

$E_3 = B_3 + B_2B_0 + B_2B_1$

K-map for E_2

B_1B_0 \ B_3B_2	00	01	11	10
00	0	1	×	0
01	1	0	×	1
11	1	0	×	×
10	1	0	×	×

$E_2 = \overline{B}_2B_0 + \overline{B}_2B_1 + B_2\overline{B}_1\overline{B}_0$

K-map for E_1

B_1B_0 \ B_3B_2	00	01	11	10
00	1	1	×	1
01	0	0	×	0
11	1	1	×	×
10	0	0	×	×

$E_1 = \overline{B}_1\overline{B}_0 + B_1B_0$

K-map for E_0

B_1B_0 \ B_3B_2	00	01	11	10
00	1	1	×	1
01	0	0	×	0
11	0	0	×	×
10	1	1	×	×

$E_0 = \overline{B}_0$

The logic diagram of BCD to excess-3 code converter is shown in Fig. 4.76.

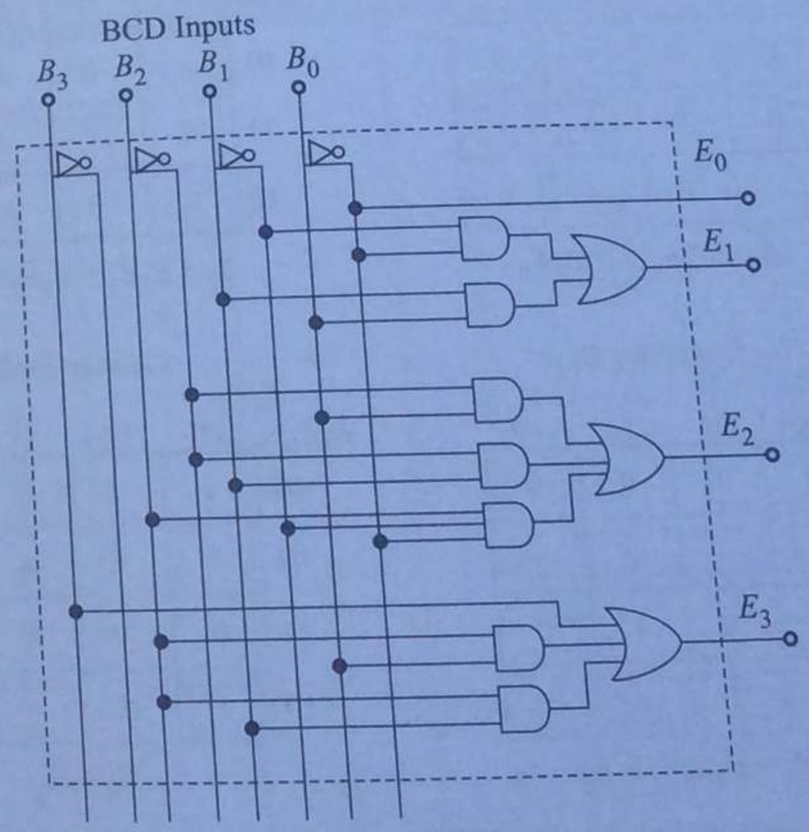


Fig. 4.76 4-bit BCD to excess-3 code converter

4.14.4 Excess-3 to BCD Converter

The truth table of excess-3 code and its equivalent BCD is given in Table 4.38.

Table 4.38 Excess-3 code and its equivalent BCD

Decimal Digit	Excess-3 code				BCD code			
	E_3	E_2	E_1	E_0	B_3	B_2	B_1	B_0
0	0	0	1	1	0	0	0	0
1	0	1	0	0	0	0	0	1
2	0	1	0	1	0	0	1	0
3	0	1	1	0	0	0	1	1
4	0	1	1	1	0	1	0	0
5	1	0	0	0	0	1	0	1
6	1	0	0	1	0	1	1	0
7	1	0	1	0	0	1	1	1
8	1	0	1	1	1	0	0	0
9	1	1	0	0	1	0	0	1

The K-map for BCD codes B_3, B_2, B_1, and B_0 with excess-3 inputs E_3, E_2, E_1, and E_0 are:

K-map for B_3

E_1E_0 \ E_3E_2	00	01	11	10
00	×	0	1	0
01	×	0	×	0
11	0	0	×	1
10	×	0	×	0

$B_3 = E_3E_2 + E_3E_1E_0$

K-map for B_2

E_1E_0 \ E_3E_2	00	01	11	10
00	×	0	0	1
01	×	0	×	1
11	0	1	×	0
10	×	0	×	1

$B_2 = \overline{E}_2\overline{E}_1 + \overline{E}_2\overline{E}_0 + E_2E_1E_0$

K-map for B_1

E_1E_0 \ E_3E_2	00	01	11	10
00	×	0	0	0
01	×	1	×	1
11	0	0	×	0
10	×	1	×	1

$B_1 = \overline{E}_1E_0 + E_1\overline{E}_0$

K-map for B_0

E_1E_0 \ E_3E_2	00	01	11	10
00	1	1	×	1
01	0	0	×	0
11	0	0	×	×
10	1	1	×	×

$B_0 = \overline{E}_0$

The logic diagram of excess-3 to BCD code converter is shown in Fig. 4.77.

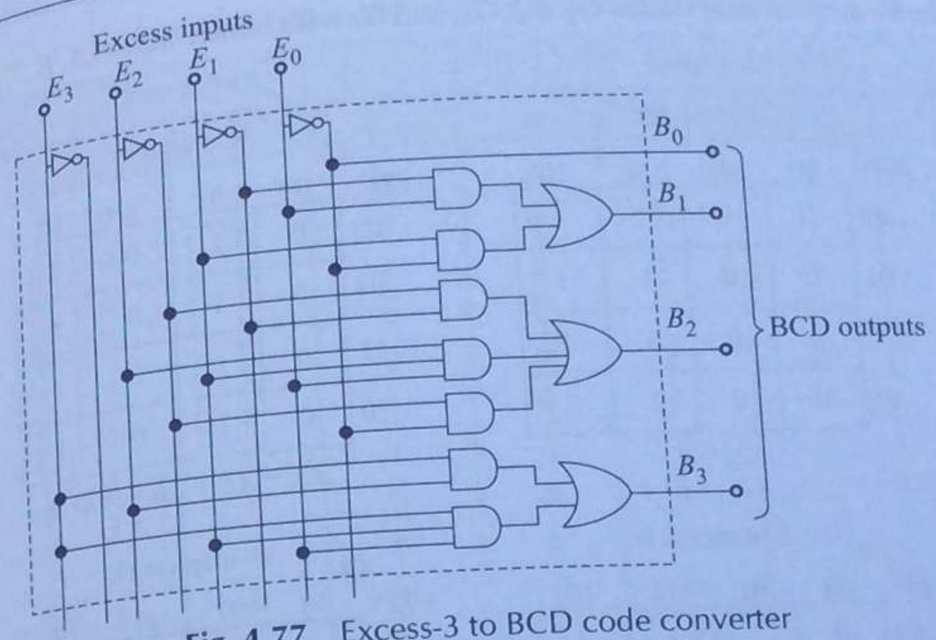


Fig. 4.77 Excess-3 to BCD code converter

4.14.5 Binary-to-Gray Code Converter

The gray code is widely used in digital systems because it has the advantage that only one bit in the numerical representation changes between successive numbers. The truth table of binary code and its equivalent gray code is given in Table 4.39.

Table 4.39 4-bit Binary and its equivalent Gary code

Binary inputs				Gray outputs			
A	B	C	D	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

The K-map for gray codes G_3, G_2, G_1, and G_0 with binary inputs A, B, C, and D are:

K-map for G_3

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

$G_3 = A$

K-map for G_2

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

$G_2 = \overline{A}B + A\overline{B} = A \oplus B$

K-map for G_1

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

$G_1 = B\overline{C} + \overline{B}C = B \oplus C$

K-map for G_0

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

$G_0 = \overline{C}D + C\overline{D} = C \oplus D$

The logic diagram of binary-to-gray code converter is shown in Fig. 4.78.

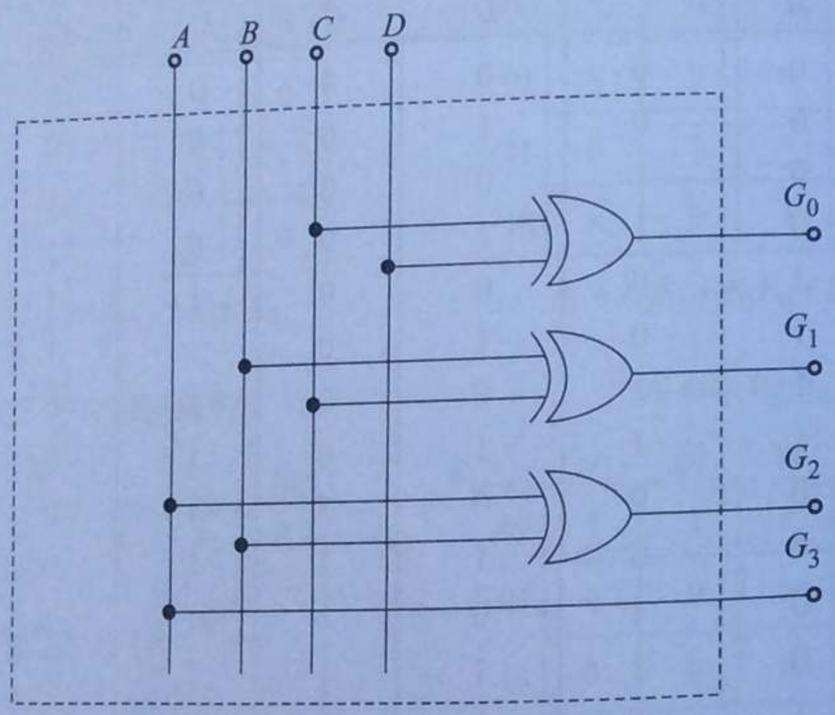


Fig. 4.78 Binary-to-gray code converter

4.14.6 Gray-to-Binary Code Converter

The truth table of gray code and its equivalent binary code converter is given in Table 4.40.

Table 4.40 Gray code and its equivalent binary code

Gray inputs				*Binary outputs*			
G_3	G_2	G_1	G_0	A	B	C	D
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	1	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	1	1	1	1	1	0
1	0	0	1	1	1	1	1
1	0	0	0				

The K-map for binary codes A, B, C, and D with gray code inputs G_3, G_2, G_1, and G_0 are:

K-map for A

G_1G_0 \ G_3G_2	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

$A = G_3$

K-map for B

G_1G_0 \ G_3G_2	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$B = \overline{G}_3G_2 + \overline{G}_2G_3 = G_2 \oplus G_3$

K-map for C

G_1G_0 \ G_3G_2	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	1	0	1	0
10	1	0	1	0

$C = G_1 \oplus B$

K-map for D

G_1G_0 \ G_3G_2	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

$D = G_0 \oplus C$

$$C = \overline{G}_3 G_2 \overline{G}_1 + G_3 \overline{G}_2 \overline{G}_1 1 + \overline{G}_3 \overline{G}_2 G_1 + G_3 G_2 G_1$$
$$= \overline{G}_1 (G_3 \oplus G_2) + G_1 (G_3 \cdot G_2)$$
$$= \overline{G}_1 (G_3 \oplus G_2) + G_1 (\overline{G_3 \oplus G_2})$$
$$= (G_3 \oplus G_2 \oplus G_1)$$
$$= (G_1 \oplus B)$$
$$D = \overline{G}_3 G_2 \overline{G}_1 \overline{G}_0 + G_3 \overline{G}_2 \overline{G}_1 \overline{G}_0 + \overline{G}_3 \overline{G}_2 \overline{G}_1 G_0 + \overline{G}_3 \overline{G}_2 G_1 \overline{G}_0 + G_3 G_2 \overline{G}_1 G_0$$
$$+ \overline{G}_3 G_2 G_1 G_0 + G_3 \overline{G}_2 G_1 G_0 + G_3 G_2 G_1 G_0$$
$$= \overline{G}_1 \overline{G}_0 (G_3 \oplus G_2) + \overline{G}_1 G_0 (G_3 \oplus G_2) + G_1 G_0 (G_3 \oplus G_2) + G_1 \overline{G}_0 (G_3 \times G_2)$$
$$= (G_3 \oplus G_2)(\overline{G}_1 \overline{G}_0 + G_1 G_0) + (G_3 \times G_2)(\overline{G}_1 G_0 + G_1 \overline{G}_0)$$
$$= (G_3 \oplus G_2)(G_1 \times G_0) + (G_3 \times G_2)(G_0 \oplus G_1)$$
$$= (G_3 \oplus G_2)\,(\overline{G_1 \oplus G_0}) + (\overline{G_3 \oplus G_2})\,(G_0 \oplus G_1)$$
$$= G_3 \oplus G_2 \oplus G_1 \oplus G_0$$
$$= (G_0 \oplus C)$$

The logic diagram of gray-to-binary code converter is shown in Fig. 4.79.

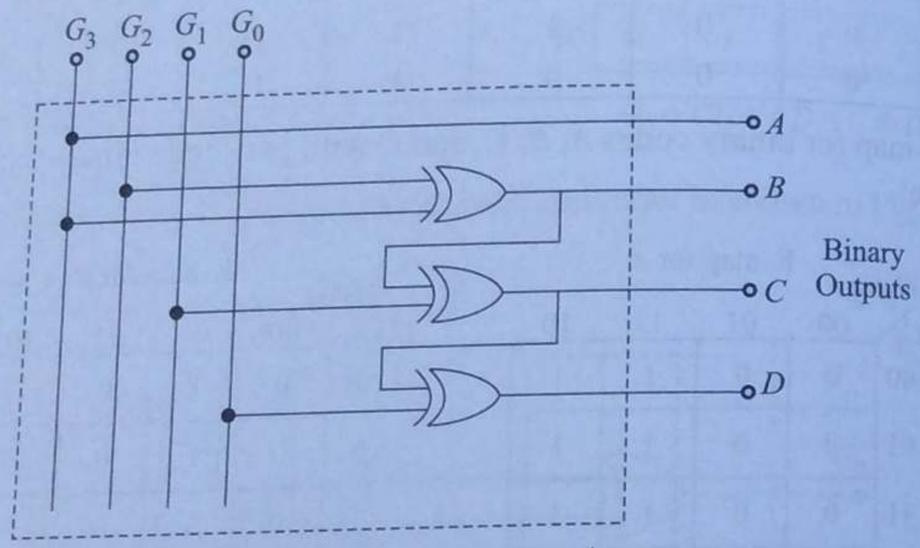


Fig. 4.79 Gray-to-binary code converter

4.14.7 BCD TO SEVEN-SEGMENT CODE CONVETER

In most of the applications, seven segments are used to display the result. The binary code of a decimal digit cannot display directly over the seven segments. There is a need to convert the binary representation of decimal digit (BCD) into seven segment code. The seven segment display includes the eight segments, which are referred as *a*, *b*, *c*, *d*, *e*, *f*, *g*, and *h*. The segment will be ON or OFF according to the input data. The seven segments are of two types: (i) common anode and (ii) common cathode.

Common anode

In a common anode type seven segment, the anode of light emitting diodes (segment) are connected together to *Vcc*. The segment is ON when the input is 0 and it is OFF when the input is 1. BCD codes and their equivalent seven-segment codes for common anode type is given in Table 4.41

Table 4.41 BCD and its equivalent seven-segment code for common anode

Decimal digit	BCD				Seven segment code							
	B_3	B_2	B_1	B_0	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*
0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	1	1	0	0	1	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0	1
3	0	0	1	1	0	0	0	0	1	1	0	1
4	0	1	0	0	1	0	0	1	1	0	0	1
5	0	1	0	1	0	1	0	0	1	0	0	1
6	0	1	1	0	0	1	0	0	0	0	0	1
7	0	1	1	1	0	0	0	1	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0	1
9	1	0	0	1	0	0	0	0	1	0	0	1

The K-maps for the segment of display a, b, c, d, e, f, g, and h with BCD inputs B_3, B_2, B_1, and B_0 are:

K-map for a

B_1B_0 \ B_3B_2	00	01	11	10
00	0	1	×	0
01	1	0	×	0
11	0	0	×	×
10	0	0	×	×

$a = B_3\overline{B}_1\overline{B}_0 + \overline{B}_3\overline{B}_2\overline{B}_1B_0$

K-map for b

B_1B_0 \ B_3B_2	00	01	11	10
00	0	0	×	0
01	0	1	×	0
11	0	0	×	×
10	0	1	×	×

$b = B_2\overline{B}_1B_0 + B_2B_1\overline{B}_0$

K-map for c

B_1B_0 \ B_3B_2	00	01	11	10
00	0	0	×	0
01	0	0	×	0
11	0	0	×	×
10	1	0	×	×

$c = \overline{B}_2\overline{B}_0B_1$

K-map for d

B_1B_0 \ B_3B_2	00	01	11	10
00	0	1	×	0
01	1	0	×	0
11	0	1	×	×
10	0	0	×	×

$d = B_2\overline{B}_1\overline{B}_0 + B_2B_1B_0 + \overline{B}_3\overline{B}_2\overline{B}_1B_0$

K-map for e

B_1B_0 \ B_3B_2	00	01	11	10
00	0	1	×	0
01	1	1	×	1
11	1	1	×	×
10	0	0	×	×

$e = B_0 + B_2\overline{B}_1$

K-map for f

B_1B_0 \ B_3B_2	00	01	11	10
00	0	0	×	0
01	1	0	×	0
11	1	1	×	×
10	1	0	×	×

$f = B_0B_1 + \overline{B}_2B_1 + \overline{B}_3\overline{B}_2B_0$

K-map for g

B_1B_0 \ B_3B_2	00	01	11	10
00	1	0	×	0
01	1	0	×	0
11	0	1	×	×
10	0	0	×	×

$g = \overline{B}_3\overline{B}_2\overline{B}_1 + B_2B_1B_0$

K-map for h

B_1B_0 \ B_3B_2	00	01	11	10
00	1	1	×	1
01	1	1	×	1
11	1	1	×	×
10	1	1	×	×

$h = 1$

The logic diagram of BCD to seven-segment code converter for common anode type is shown in Fig. 4.80.

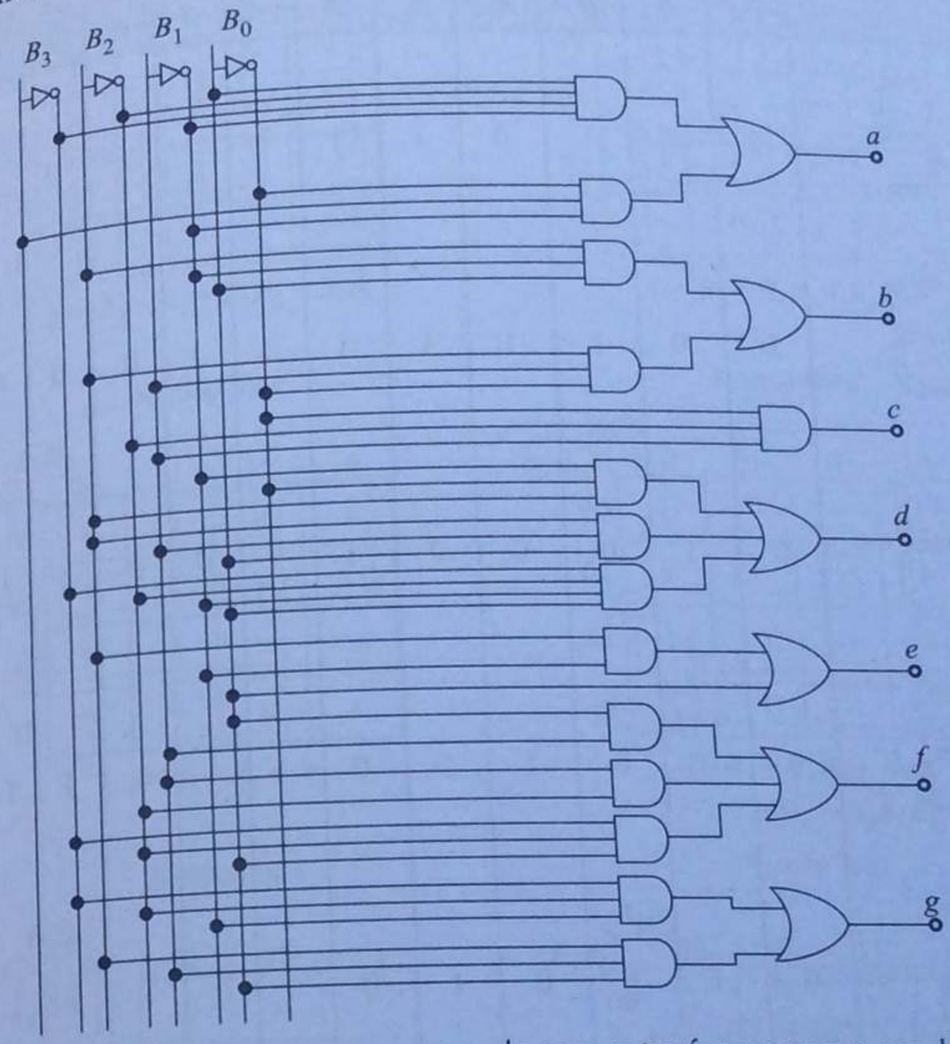


Fig. 4.80 BCD to seven-segment code converter for common anode type

Common cathode

In a common cathode type seven segment, the cathode of light emitting diodes (segment) are connected together to ground. The segment is ON when the input is 1 and it is OFF when the input is 0. BCD codes and their equivalent seven-segment code for common cathode type is given in Table 4.42

Table 4.42 BCD and its equivalent seven-segment code for common cathode

Decimal digit	*BCD code*				*Seven-segment code*							
	B_3	B_2	B_1	B_0	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*
	0	0	0	0	1	1	1	1	1	1	0	0
	0	0	0	1	0	1	1	0	0	0	0	0

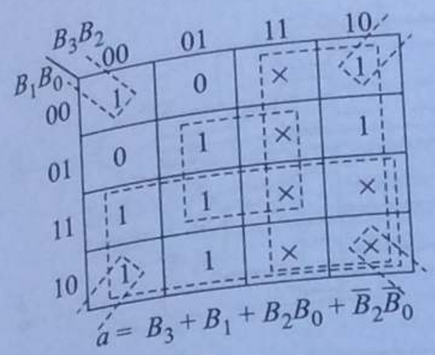
Contd.

Contd.

Decimal digit	BCD code				Seven-segment code							
	B_3	B_2	B_1	B_0	a	b	c	d	e	f	g	h
2	0	0	1	0	1	1	0	1	1	0	1	0
3	0	0	1	1	1	1	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0	0	1	1	0
5	0	1	0	1	1	0	1	1	0	1	1	0
6	0	1	1	0	1	0	1	1	1	1	1	0
7	0	1	1	1	1	1	1	0	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1	0
9	1	0	0	1	1	1	1	1	0	1	1	0

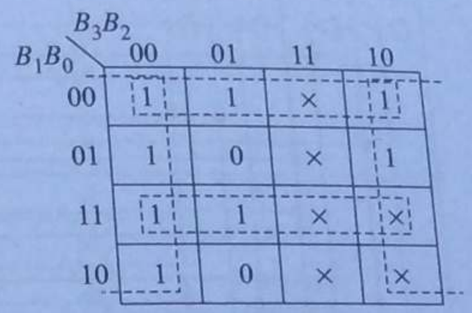
The K-maps for the segment of display a, b, c, d, e, f, g, and h with BCD inputs B_3, B_2, B_1, and B_0 are:

K-map for a



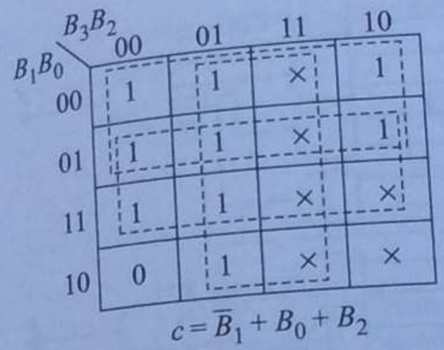
$a = B_3 + B_1 + B_2B_0 + \overline{B}_2\overline{B}_0$

K-map for b



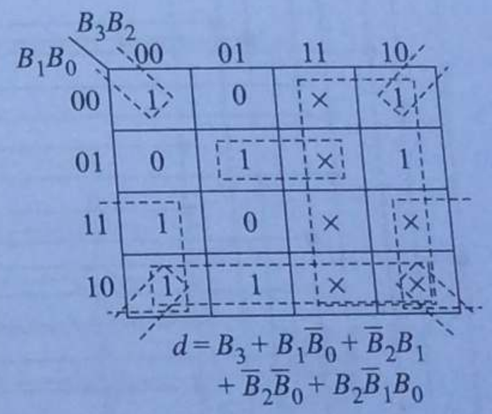
$b = \overline{B}_2 + \overline{B}_1\overline{B}_0 + B_1B_0$

K-map for c



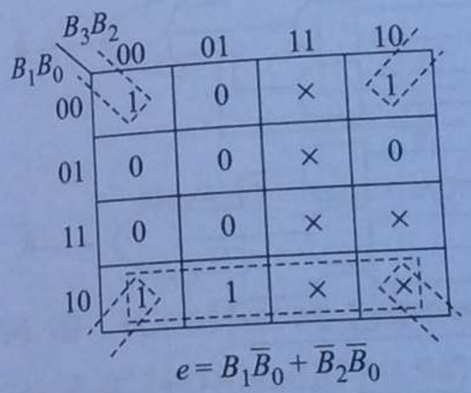
$c = \overline{B}_1 + B_0 + B_2$

K-map for d



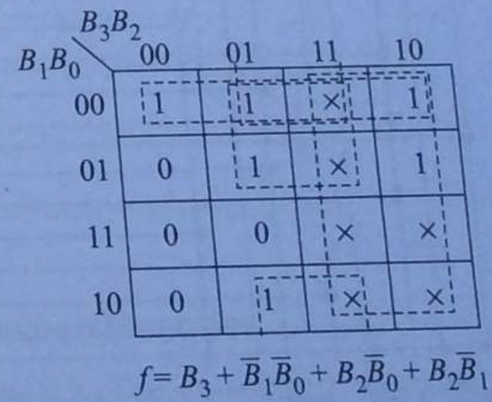
$d = B_3 + B_1\overline{B}_0 + \overline{B}_2B_1 + \overline{B}_2\overline{B}_0 + B_2\overline{B}_1B_0$

K-map for e



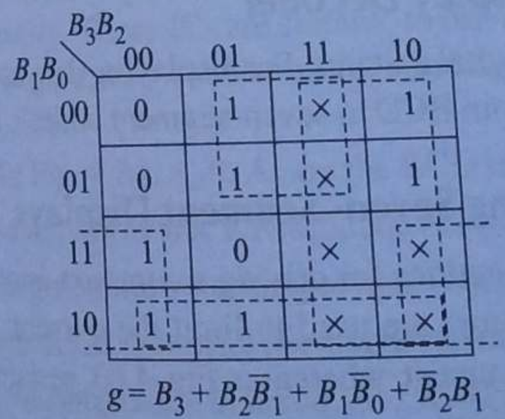
$e = B_1\overline{B}_0 + \overline{B}_2\overline{B}_0$

K-map for f



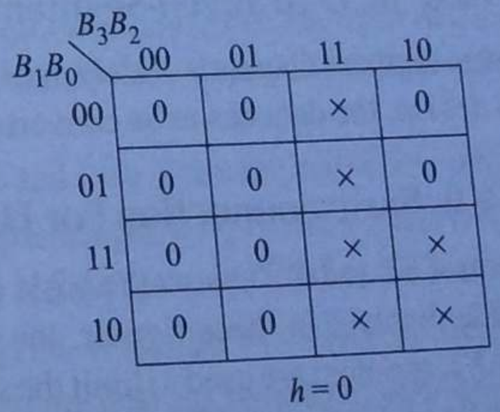
$f = B_3 + \overline{B}_1\overline{B}_0 + B_2\overline{B}_0 + B_2\overline{B}_1$

K-map for g



$g = B_3 + B_2\overline{B}_1 + B_1\overline{B}_0 + \overline{B}_2B_1$

K-map for h



$h = 0$

The logic diagram of BCD to seven-segment code converter for common cathode type is shown in Fig. 4.81.

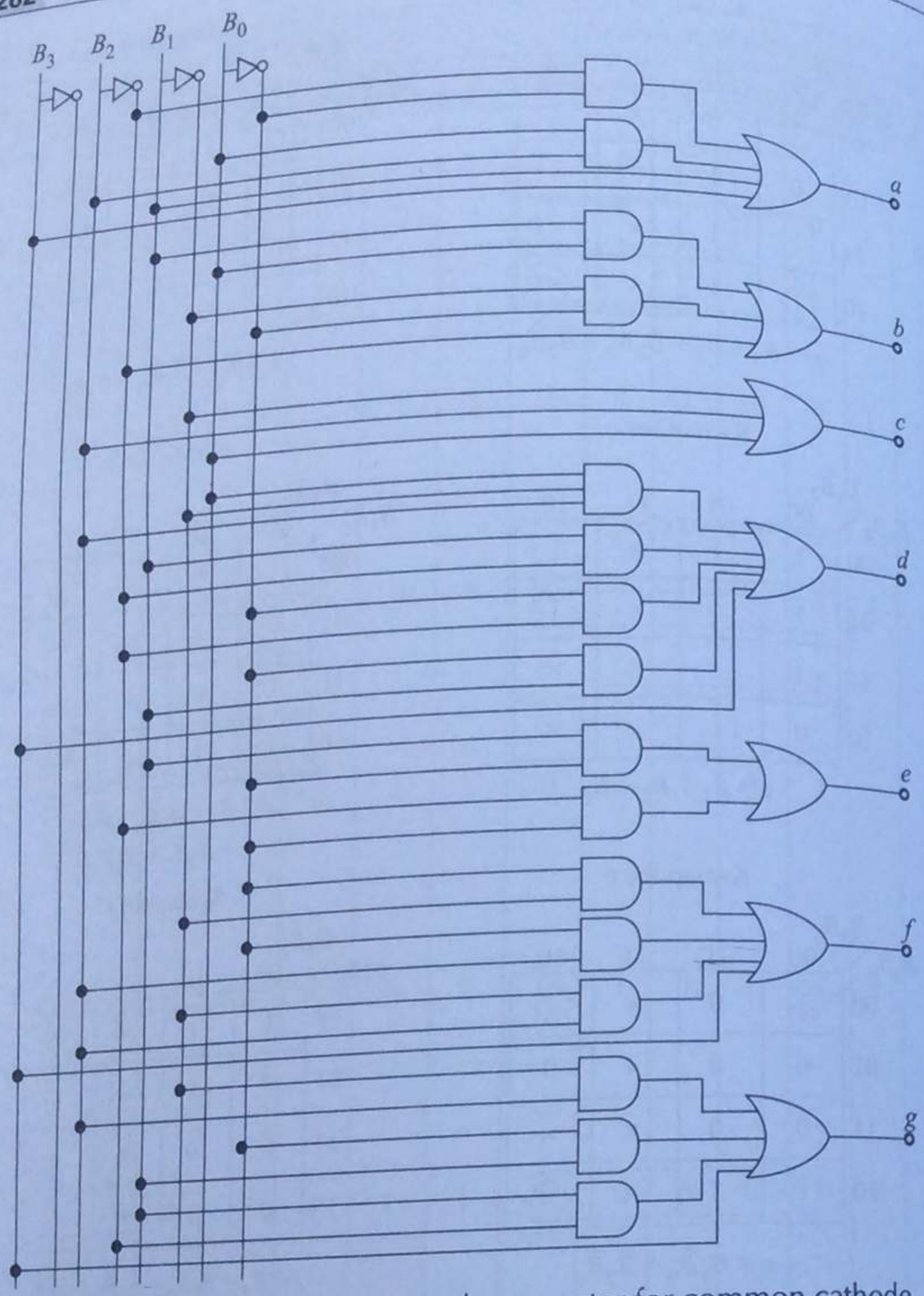


Fig. 4.81 BCD to seven-segment code converter for common cathode

4.14.8 BCD to Seven-Segment Display Decoder

Seven-segment display is widely used in digital systems. For displaying data using this device, the data has to be converted from BCD to seven-segment code.

4.14.9 Basic connection For Driving Seven–Segment Displays

Figures 4.82 and 4.83 show the basic connections for driving a common anode/ cathode display. In these figures, the resistors are used to limit the current. In Fig. 4.82, resistors are used to limit the sink current, whereas in Fig. 4.83, resistors are used to limit the source current of the driver.

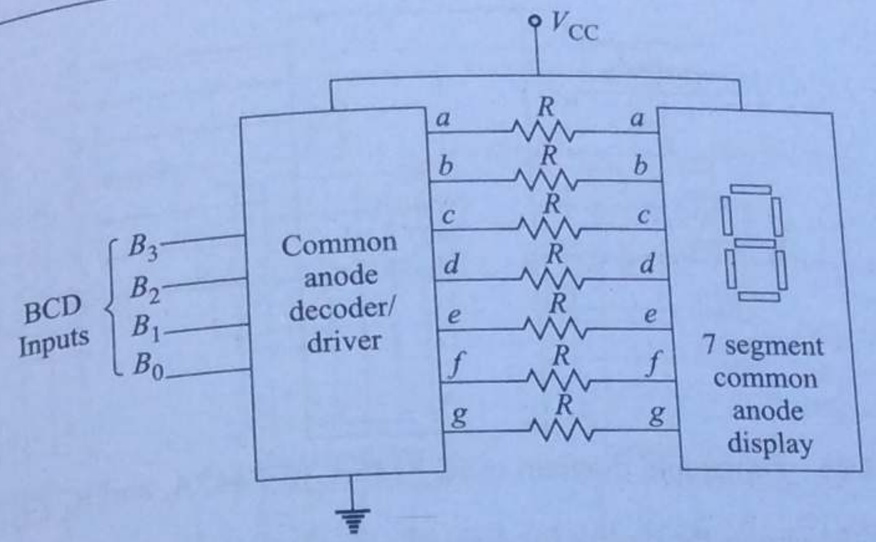


Fig. 4.82 Logic diagram-common anode dsiplay

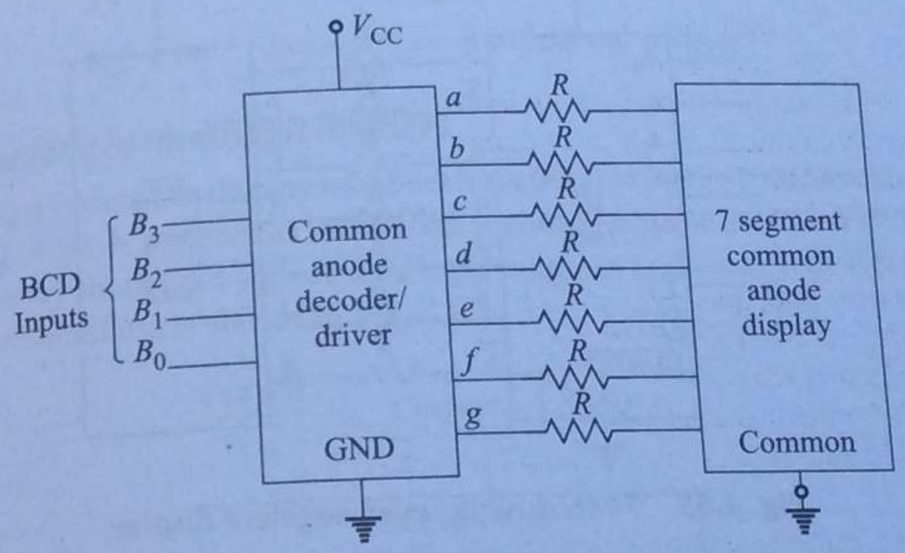


Fig. 4.83 Logic diagram—common cathode display

4.14.10 ICs of Seven-Segment Driver/Decoder

IC 7446A, IC 7447A, and IC 74LS47 are the ICs of seven-segment driver/decoder. They accept 4-bit BCD input data and provide open collector outputs to drive the individual segments directly. Each segment has 40 mA maximum sink current capacity. These ICs are suitable to drive common anode seven-segment display.

Figure 4.84 shows the functional diagram of IC 7446A, IC 7447A, and IC 74LS47.

In Fig. 4.84, A_0 to A_3 are the BCD inputs and $\overline{a}$ to $\overline{g}$ are the active low outputs of seven-segments. $\overline{\text{LT}}$ is the active low lamp test input; it is provided to test whether all segments are working or not. When $\overline{\text{LT}}$ is held low with $\overline{\text{RBO}}$ open or at logic high, the IC drives all the display terminals ON. When the $\overline{\text{BI / RBO}}$ is pulled low, all the outputs are blanked. $\overline{\text{BI / RBO}}$ along with RBI can be used to provide *ripple blanking*.

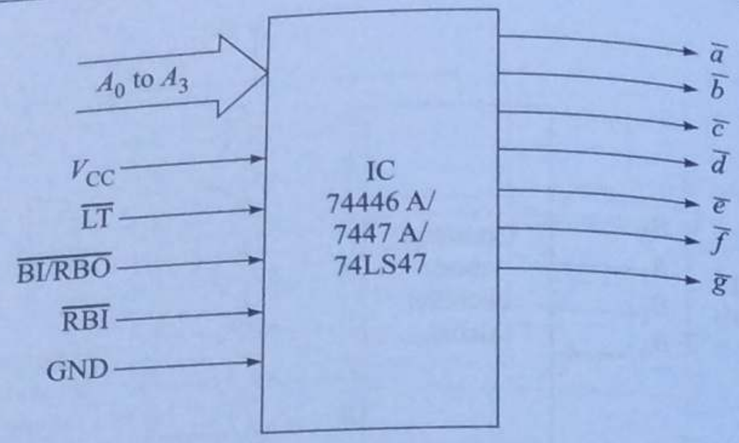


Fig.4.84 Functional diagram of IC 7446A, IC 7447A, and IC 74LS47

Figure 4.85 shows the circuit for driving a single seven-segment display using 7446.

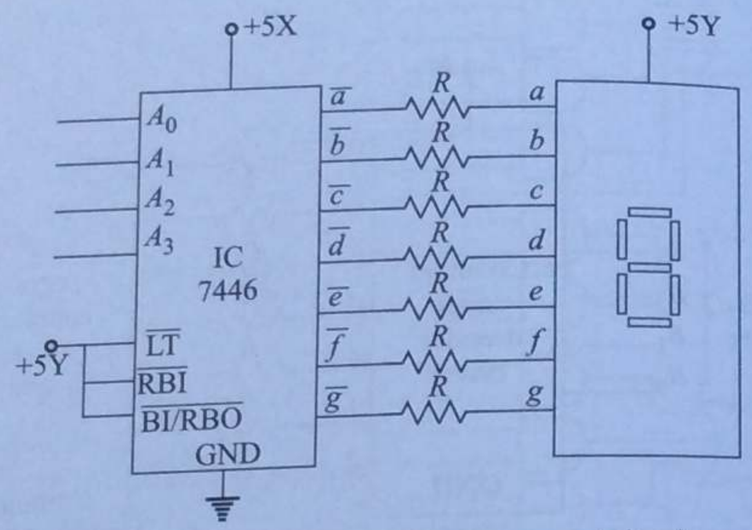


Fig. 4.85 7446 driving seven-segment display

The resistor R is used to limit the current. The value of resistors can be calculated as

$$V_{CC} - \text{drop across LED} - I \times R = 0$$

The drop across the LED segment is approximately 1.5 V and the current I is within the range of 5 to 30 mA. Let us assume current $I = 20$ mA.

$$5 - 1.5\text{ V} - 20 \times 10^{-3} \times R = 0$$

$$\therefore \qquad R = \frac{5 - 1.5}{20 \times 10^{3}} = 116.67\ \Omega$$

It displays a single digit at a time. But in most of the applications, there is a need to display more than one digit. Different techniques are used to drive multidigit displays, such as:

(a) Cascaded non-multiplexed displays,
(b) Ripple blanking in multidigit displays, and
(c) Multiplexed displays.

Cascaded non-multiplexed displays

To display more than one digit, more than one seven-segment displays are required. Each display has separate decoder IC, as shown in Fig. 4.84. The control logic is used to drive the decimal point.

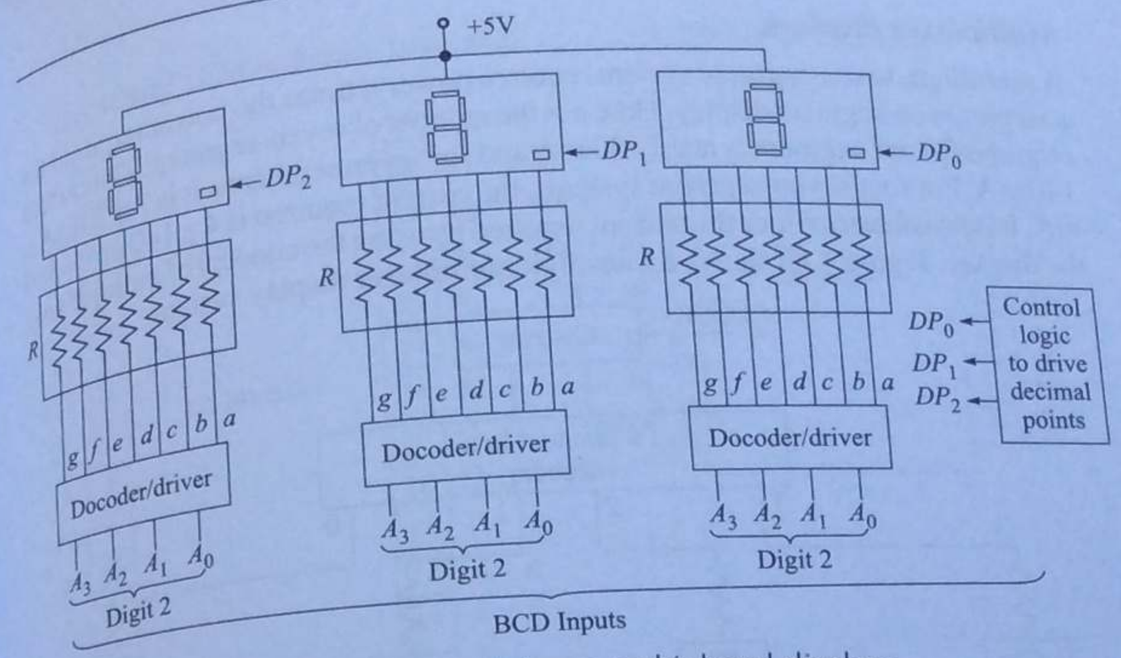


Fig. 4.86 Cascaded non-multiplexed displays

Ripple blanking in multidigit displays

The four-digit system can display numbers from 0000 to 9999. If we have to display 405, in Fig. 4.84, it appears as 0405. But it should appear as 405. For this, the leading display should not appear. The logic diagram to display four bits with the leading zero blanking is shown in Fig. 4.87.

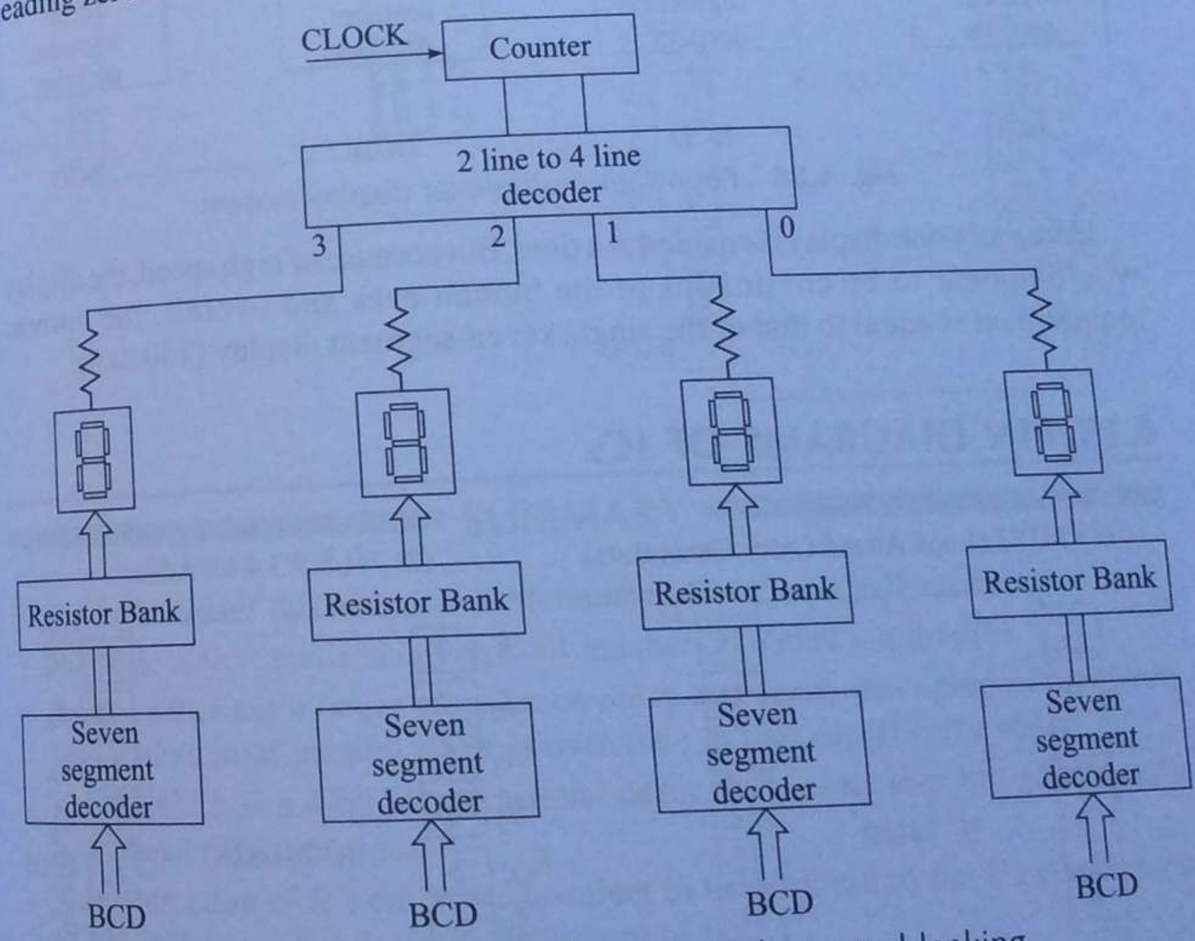


Fig. 4.87 Display four bits with leading zero blanking

Multiplexer displays

A multidigit, seven-segment system requires power n times the power required for a single seven-segment display. Here n is the number of seven-segment. The current required for one segment is nearly 20 mA and for seven segments, it is 7×20 mA = 140 mA. For four seven-segment systems, the current required is 4×140 mA = 560 mA. It is possible to reduce the current required by using the concept of multiplexing the display. Figure 4.88 shows a four-digit multiplexer display system.

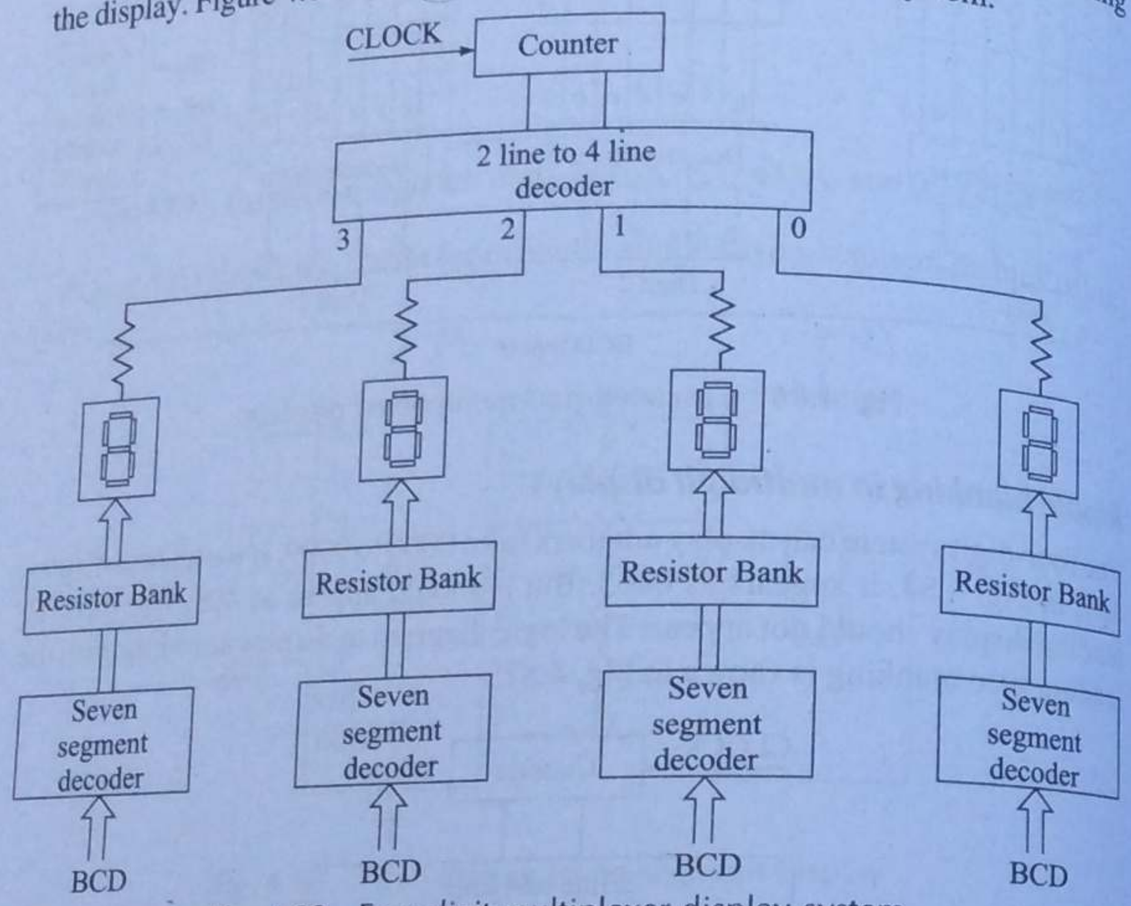
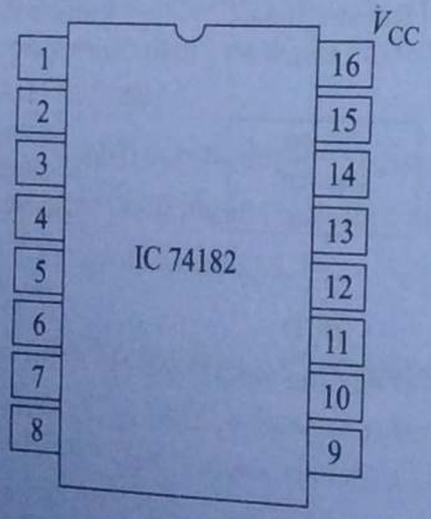


Fig. 4.88 Four-digit multiplexer display system

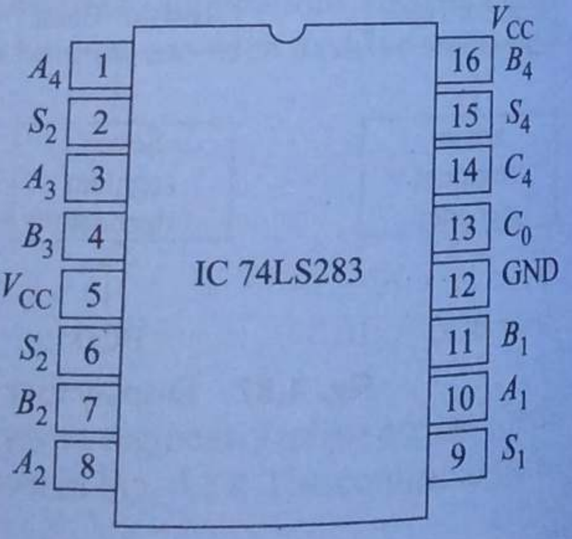
Here, only one display is enabled at a time. But because of high speed, the display would appear to be continuous to the human eyes and overall, the current requirement is equal to that of the single seven-segment display (140 mA).

4.15 PIN DIAGRAMS OF ICs

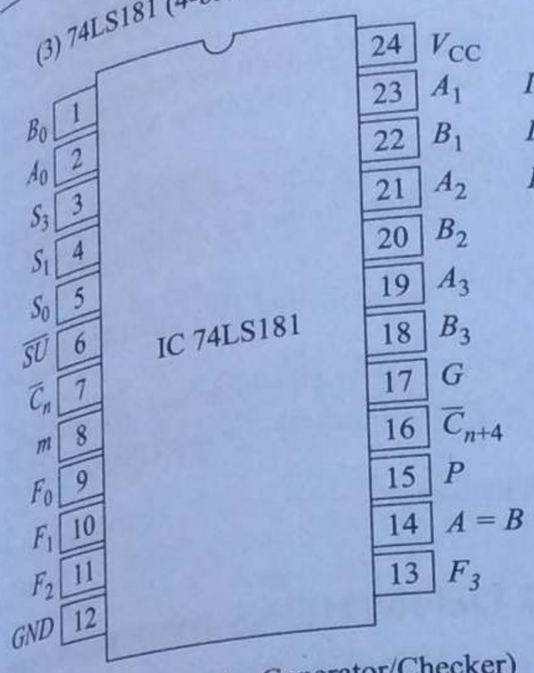
(1) 74182 (Look Ahead Carry Generators)



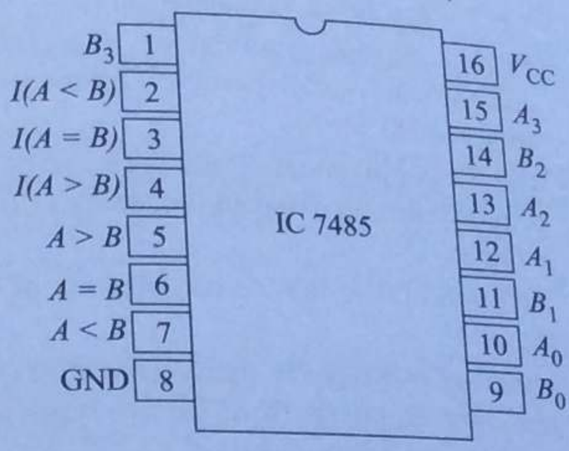
(2) 74LS283 4-bit Adder



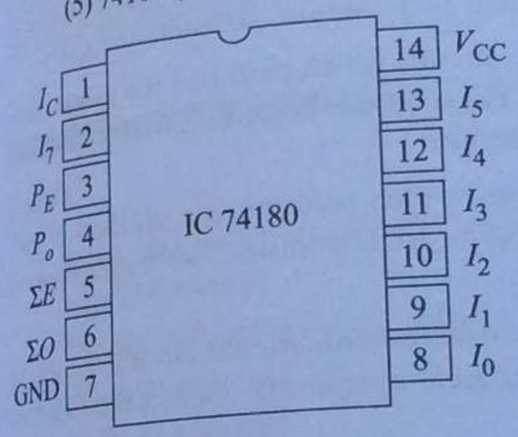
(3) 74LS181 (4-bit Arithmetic Logic unit)



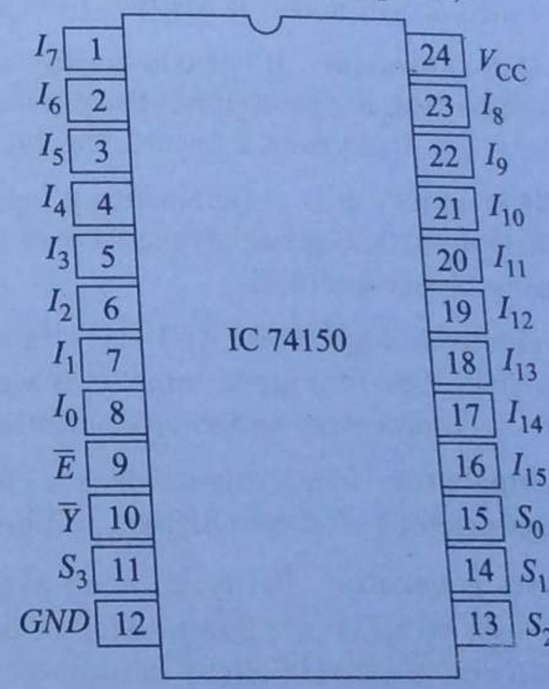
(4) 7485 (4-bit comparator)



(5) 74180 (Parity Generator/Checker)



(6) Multiplexer ICs:
(a) 74150 (16: 1 multiplexer):



SUMMARY

- The logic circuit that adds two 1-bit numbers is called a half-adder.
- The logic circuit that adds three 1-bit numbers is called a full-adder.
- Parallel adder has a larger propagation delay and hence, low speed of operation. This limitation of parallel adder is overcome in look-ahead carry adder.
- IC 74LS83: It is a 4-bit binary parallel adder. It contains four full-adders and a look-ahead carry generator.
- The subtraction of B from A is equivalent to the addition of the 1's complement of B to A, i.e. $A - B = A +$ 1's complement of B.
- In 1's complement subtraction, if carry is 1, then the result is positive, and the correct result is obtained by adding the end-around carry. If the carry is 0, then the result is negative and it is in 1's complement form.

- The subtraction of B from A is equivalent to addition of 2's complement of B to A, i.e. $A - B = A +$ 2's complement of B.
- In 2's complement subtraction, if a carry is generated, then the result is positive, and the carry is discarded. Otherwise, the result is negative and it is in 2's complement form.
- The 9's complement of any digit can be obtained by subtracting the digit from 9, or it can also be obtained by adding $(10)_{10}$ in binary to the 1's complement of that number.
- Addition of 1 to the 9's complement of the digit gives the 10's complement of a digit.
- There are two types of parity generator: (i) even parity generator and (ii) odd parity generator. Similarly there are two types of parity checkers: (i) even parity checker and (ii) odd parity checker.
- IC 74180 is a 9-input parity generator/checker.

KEY TERMS AND DEFINITIONS

Half-adder The logic circuit that performs the addition of two 1-bit numbers is called as half-adder. It has two 1-bit inputs and two outputs, viz. sum and carry.

Half-subtractor It is a combinational logic circuit, which performs the subtraction of two 1-bit numbers. It has the provision to consider the borrow, which could have been generated from a lower order bit subtraction.

BCD-adder It is a combinational logic circuit, which performs the addition of two BCD digits and produces the sum in BCD form. BCD addition can be performed using binary addition.

Arithmetic logic unit (ALU) It is a widely used combinational circuit, which is capable of performing arithmetic as well as logical operations. This is the heart of any microprocessor and microcontroller.

Comparator The comparator is a combinational logic circuit. It compares the magnitude of two n-bit numbers and provides the relative result as the output.

Parity generator Parity bit is an extra bit included with binary information to detect the errors during transmission of binary information. In digital communication, an extra bit is added in binary message such that the total number of 1s in the message can be either odd or even according to the type of parity used. The parity generator is a combinational logic circuit that generates the parity bit(s).

Even parity generator It is a combinational logic circuit that generates the parity bit such that the number of 1s in the message becomes even. The parity generator checks the input's binary information and generates the parity bit 0/1 such that after the addition of parity bit, the total number of 1s in the message becomes even.

Odd parity generator It is a combinational logic circuit that generates the parity bit such that number of 1s in the message becomes odd. The parity generator checks the input's binary information and generates the parity bit 0/1 such that after the addition of parity bit, the total number of 1s in the message becomes odd.

Parity checker At the receiving end, a combinational logical circuit is used to check the parity of the received information. It determines whether the error is included in the message or not. The combinational logic circuit used at the receiver to check the parity of the received information is known as the parity checker.

Even parity checker It is a combinational logical circuit; it has n-bit inputs message and a parity error as the output. The circuit checks the parity of inputs and provides the output 0/1. For an even parity checker, if the parity of input message is even, then the output is 0, otherwise it is 1. When the output is 1, it shows there is an error in the message.

Odd parity checker It is a combinational logical circuit; it has n-bit inputs message and a parity error as the output. The circuit checks the parity of inputs and provides the output 0/1. For an odd parity checker, if the parity of input message is odd, then the output is 0, otherwise it is 1. When the output is 1, it shows there is an error in the message.

Multiplexer It is a combinational logic circuit having multiple inputs and a single output. Selectors select one input at a time and send it to the output line. For an n-input multiplexer, m select inputs are required, where $n = 2^m$.

Demultiplexer It is a combinational logic circuit having a single input and many outputs. It performs the reverse operation of a multiplexer. It accepts a single input and sends it to one of the output lines, according to the select inputs. For an n-output demultiplexer, the number of select lines is m, where $n = 2^m$.

Excess-3 to BCD code Converter Excess-3 code is a modified BCD code. It is obtained by adding 3 to each BCD code.

EXERCISE

Part-I

1. Draw the block diagram of IC74181 and explain the function of all the important pins.
2. Design and implement a 2-bit comparator using suitable gates.
3. Write short notes on:
   (i) BCD addition (ii) Priority encoder
4. Design a full-subtractor circuit using AND-OR-INVERTER logic.
5. Briefly describe the following: (i) parallel adder and (ii) serial adder
6. Implement the following:
   (i) 4-bit prime number detector using 8:1 multiplexers
   (ii) 2-bit comparator with 4:16 line decoder IC (Generate only EQUALITY output)
7. Implement a 16:1 multiplexer using 4:1 multiplexer ICs.
8. Assume that 1011 input data pattern is loaded into a 4-bit ring counter. Sketch the resulting flip-flop Q output waveforms (Assume positive edge triggering).
9. Write short notes on:
   (i) Static bi-polar RAM cell
   (ii) Look-ahead carry generator
10. Write short notes on: (i) 4-bit binary adder, (ii) 4-bit binary subtractor.
11. Design a combinational circuit for an 8-segment common cathode display as shown in Fig. 4.89. It has 4-bit binary input and eight outputs as X_1 to X_8. It

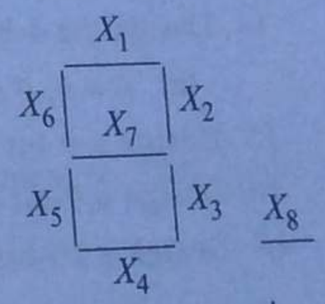


Fig. 4.89 Display

should display the equivalent of the 4-bit input binary number which is in 1's complement form. For example, 0001 => 1 1110 => –1.

12. Design a 1-bit BCD adder using 7483 IC.
13. Explain how an 8-bit ALU can be constructed from two 74181 ICs.
14. Implement the following function using 8:1 MUX:
    $f(A, B, C, D) = \Sigma m(2, 4, 5, 7, 10, 14)$
15. Design a full-adder using 74138 IC.
16. Write a short note on 2-bit magnitude comparator.
17. Design an odd parity generator for a 4-bit message code.
18. A combinational circuit has four inputs and one output. The output is equal to 1 when: (i) all the inputs are equal to 1, (ii) none of the inputs are equal to 1, and (iii) an odd number of inputs are equal to 1.
    (a) Obtain the truth table.
    (b) Find the simplified output function in sum of products form.
    (c) Find the simplified output function in product of sums form.
    (d) Draw the logic diagram for sum of products equation in (b).
19. Implement the following functions with an 8:1 MUX:
    (i) $F(w, x, y, z) = \Sigma m(0, 1, 3, 4, 5, 8, 9, 15)$
    (ii) $F(A, B, C) = \Sigma m(2, 4, 5, 7)$
20. Design a full-subtractor using 3:8 decoders.

Part-II

1. Design a 2-bit comparator using logic gates.
2. Design a combinational logic circuit with four input variables that will produce logic 1 output when the number of 0s in input is even.
3. Design a combinational logic circuit with three input variables that will produce logic 1 output when more than one input variables are logic 1.
4. Define half-adder and full-adder.
5. Design a half-adder using NAND gates only.
6. Design a full-adder using NAND gates only.
7. Define half-subtractor and design it using NAND gates only.
8. Define full-subtractor and design it using NAND gates only.
9. Draw and explain the block diagram of an n-bit parallel adder and explain its limitations.
10. Draw and explain the carry look-ahead adder.
11. Draw the logic diagram of a 4-bit subtractor using 7483.
12. Draw the logic diagram of a 4-bit ALU.
13. Design the 4-bit BCD adder using 7483.
14. Design the 4-bit adder using 74181. Show the working for the following cases:
    (a) $A = 6, B = 3$ (b) $A = 9, B = 8$
15. Design a 9-bit odd parity checker using 74180 and suitable gates.
16. Design a 9-bit even parity generator using 74180.
17. Design a 10-bit even parity generator using 74180.

18. Implement the following functions using a multiplexer:
    (a) $Y(A, B, C) = \Sigma m(1, 2, 5, 6)$ (b) $Y(A, B, C) = \Sigma m(1, 2, 5, 6)$
19. Design the following combinational logic circuits using a multiplexer:
    (a) Half-adder (b) Full-adder
20. Design an 8:1 multiplexer using NAND gates only.
21. Implement the following expression using a single 8:1 multiplexer:
    $Y(A, B, C, D) = \Sigma m(0, 1, 2, 5, 7, 8, 9, 14, 15)$
22. Implement a full-adder using two 4:1 multiplexers.
23. Implement a full-subtractor using two 4:1 multiplexers.
24. Design a 16:1 MUX using:
    (a) 3:1 MUX and OR gate (b) 8:1 and 2:1 MUX
25. Implement the following functions using 3-to-8-line decoder:
    (a) $Y_0(A, B, C) = \Sigma m(0, 1, 5, 6)$ (b) $Y_1(A, B, C) = \Sigma m(0, 4, 5, 7)$
    (c) $Y_2(A, B, C) = \Sigma m(2, 3, 4, 6)$
26. Design a BCD to Excess-3 code converter using truth table, K-maps, and logic circuits.
27. Design the following code converters:
    (a) Binary to BCD (b) BCD to binary
    (c) Binary to Excess-3 (d) Excess-3 to binary
    (e) BCD to Excess-3 (f) Excess-3 to BCD
    (g) Binary to gray (h) Gray to binary
28. Design a 4-bit gray-to-binary converter using truth table, K-maps, and logic circuits.

Multiple-Choice Questions

1. In a half-adder having two inputs A and B and two outputs (S and C are the sum and carry output bits, respectively), the Boolean expression for S and C in terms of A and B is .....
    (a) $S = \overline{A}B + A\overline{B}, C = AB$ (b) $S = AB + \overline{A}B, C = A + B$
    (c) $S = \overline{A}\,\overline{B} + AB, C = A + \overline{B}$ (c) $S = \overline{A} + \overline{A}\,\overline{B}, C = \overline{A} + \overline{B}$
2. For a binary half-subtractor having two inputs A and B, the correct set of logic expressions for the outputs D(= A minus B) and X(= borrow) are .....
    (a) $D = AB + \overline{A}B, X = \overline{A}B$ (b) $D = \overline{A}B + A\overline{B}, X = A\overline{B}$
    (c) $D = \overline{A}B + A\overline{B}, X = \overline{A}B$ (d) $D = AB + \overline{A}\,\overline{B}, X = A\overline{B}$
3. The logical expression $Y = A + \overline{A}B$ is equivalent to .....
    (a) $Y = AB$ (b) $Y = \overline{A}B$
    (c) $Y = A + \overline{B}$ (d) $Y = A + B$
4. The minimum number of NAND gates required to implement $A + A\overline{B} + A\overline{B}C$ is equal to .....
    (a) 0 (b) 1
    (c) 4 (d) 7

5. The dual of the Boolean theorem $A \cdot (B + C) = AB + AC$ is
   (a) $A \cdot (B + C) = AB + AC$ (b) $A \cdot (B + C) = (A + B)(A + C)$
   (c) $A \cdot BC = (A + B)(A + C)$ (d) None of these
6. A carry look-ahead adder is frequently used for addition, because it .....
   (a) is faster (b) is more accurate
   (c) uses fewer gates (d) costs less
7. A full-adder can be implemented with half-adders and OR gates. A 4-bit parallel full-adder without any initial carry requires .....
   (a) 8 half-adders and 4 OR gates (b) 8 half-adders and 3 OR gates
   (c) 7 half-adders and 4 OR gates (d) 7 half-adders and 4 OR gates
8. A combinational circuit is one in which the output depends on the .....
   (a) input combination at that time
   (b) the previous output and input combination
   (c) the previous input and input combination at that time
   (d) present output and the previous output
9. Which of the following circuit is a parallel-to-serial converter?
   (a) digital counter (b) decoder
   (c) demultiplexer (d) multiplexer
10. The code used to reduce the error due to ambiguity in reading of a binary encoder is .....
   (a) octal code (b) excess-3 code
   (c) gray code (d) BCD code
11. In 8421 BCD code, the decimal number 125 is written as .....
   (a) 1111101 (b) 000100100101
   (c) 7D (d) none of these
12. With four Boolean variables, how many Boolean expressions can be formed?
   (a) 16 (b) 256
   (c) 1024 (d) 64K

Answers

1. (a)	2. (c)	3. (d)	4. (a)
5. (d)	6. (a)	7. (a)	8. (a)
9. (d)	10. (c)	11. (b)	12. (a)

Chapter 5

SEQUENTIAL LOGIC CIRCUITS

Chapter Outline

- Flip-flop as 1-bit memory cell
- Different types of flip-flops
- Types of triggering used in flip-flops
- Applications of flip-flop as shift register
- Applications of shift register
- Asynchronous and synchronous counter
- Application of flip-flop as sequence generator and detector
- Design concept of sequence generator
- Study of synchronous circuit
- Moore and Mealy logic
- Analysis of clocked sequential circuit
- Design of clocked sequential circuit
- Study of state diagram and state table
- State reduction technique and state assign

5.1 INTRODUCTION

In the previous chapter, we have studied the basic concepts of combinational logic circuit and its design. Let us now learn another important digital system known as the *sequential circuit*. In a sequential circuit, the output is dependent upon the present inputs as well as the past inputs and outputs. The sequential circuits include the memory elements, which store the past inputs and outputs. The flip-flop is a 1-bit memory cell; it stores the 1-bit logical data (logic 0 or logic 1). The data available in memory can be used for further operation. This chapter includes the study of different types of flip-flops, such as

1. S-R flip-flop,
2. J-K flip-flop,
3. Master-Slave J-K flip-flop,
4. T flip-flop, and
5. D flip-flop.

A set of flip-flops is referred as a *shift register*, and it can store more than 1-bit data. This chapter also includes shift registers and their applications, such as

1. Parallel-to-serial converter,
2. Serial-to-parallel converter,
3. Ring and twisted-ring counter, and
4. Sequence generator and detector.

The sequential logic circuit is a digital circuit, in which the output is a function of the present inputs as well as the past inputs and outputs. It includes memory elements to store the past data. The flip-flop is a basic element of sequential logic circuits. Using flip-flops and combinational logic circuit, any sequential circuit can be designed. The basic block diagram of a sequential logic circuit is shown in Fig. 5.1.

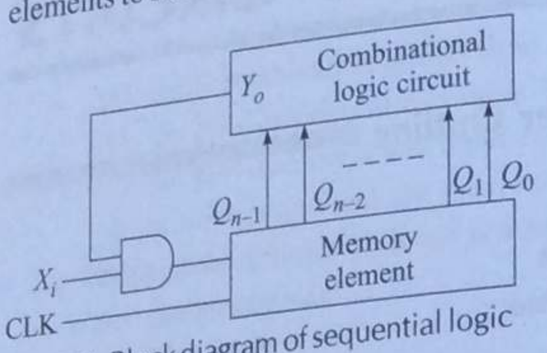


Fig. 5.1 Block diagram of sequential logic circuit

The output of a combinational logic circuit is a function of the previous output of memory elements, which may be given as an input to control the next output of the system. In some of the applications, the output is a function of present and past inputs only. In such case, the output of combinational logic circuit is not given to the input of memory elements. There are two types of sequential circuits: (a) *asynchronous circuit* and (b) *synchronous circuit*.

5.2 1-BIT MEMORY CELL

In digital systems, data is represented by two states; one is logic 0 state and another is logic 1 state. A 1-bit memory cell stores the data in terms of 0 or 1. This 1-bit memory cell is commonly referred as a flip-flop. The circuit having two stable states can be used as a flip-flop. The 1-bit memory cell can be obtained by using transistors, NAND gates, or NOR gates.

5.2.1 1-Bit Memory Cell Using Transistors

The circuit diagram of a 1-bit memory cell using transistors is shown in Fig. 5.2.

When the switch of power supply is ON, due to small difference in h_{fe} of transistors, one of the transistors takes more current and goes in saturation and the other goes in cut-off. Let us assume transistor T_1 goes in saturation and T_2 in cut-off, the output of T_1 is $V_{CE\,sat} = 0.2$ V, say logic 0 state and the output of T_2 is V_{CC} = +5 V, say logic 1 state. Both the states are stable. This circuit does not have a control input and the output of circuit cannot be changed.

The 1-bit memory cell with an external control input circuit is shown in Fig. 5.3.

Operation

1. When inputs are $S = 0$ and $R = 0$, then there is no change in state. The current state is same as the previous state.

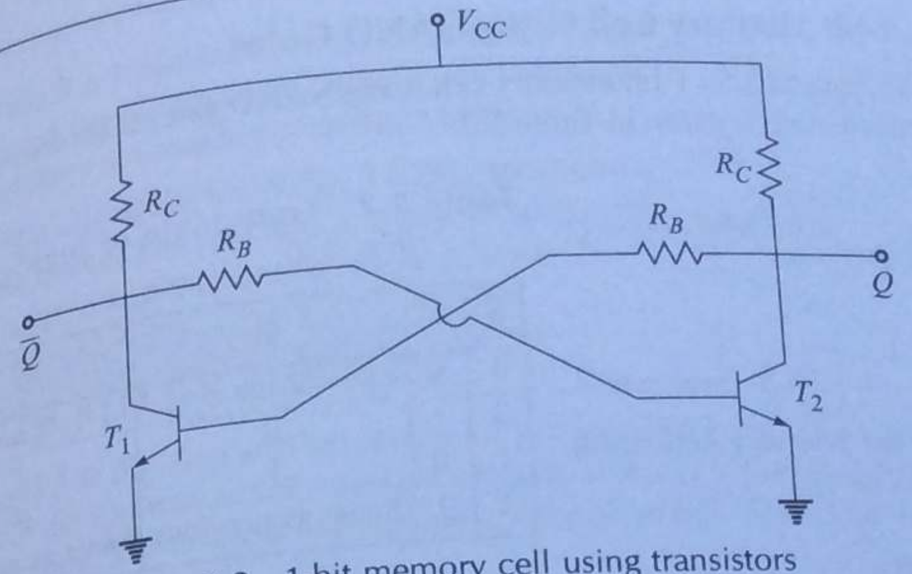


Fig. 5.2 1-bit memory cell using transistors

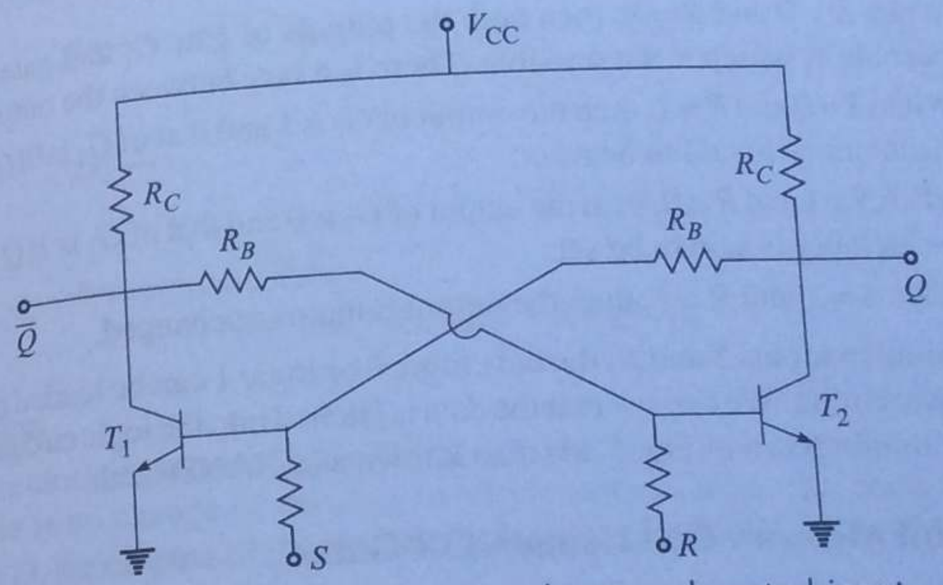


Fig. 5.3 1-bit memory cell with external control input

2. When inputs are $S = 1$ and $R = 0$, then T_1 goes in saturation and T_2 goes in cut-off. The output of T_1 is $V_{CE\,sat} = 0.2$ V, say logic 0 state and the output of T_2 is V_{CC}, say logic 1 state, and the flip-flop is said to be *set*.
3. When inputs are $S = 0$ and $R = 1$, then T_2 goes in saturation and T_1 goes in cut-off. The output of T_2 is $V_{CE\,sat} = 0.2$ V, say logic 0 state and the output of T_1 is V_{CC}, say logic 1 state, and the flip-flop is said to be *reset*.
4. When inputs are $S = 1$ and $R = 1$, then both the transistors try to go in saturation, which is not possible.

The truth table of the circuit is given in Table 5.1.

Table 5.1 Truth table of circuit (Fig. 5.3)

S	R	Q	$\overline{Q}$
0	0	Same as previous	Same as previous
0	1	0	1
1	0	1	0
1	1	Not possible	Not possible

5.2.2 1-bit Memory Cell Using NAND Gates

The logic diagram of a 1-bit memory cell using NAND gates is shown in Fig. 5.4 and the truth table is given in Table 5.2.

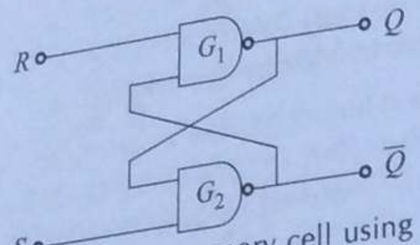


Fig. 5.4 1-bit memory cell using NAND gates

Table 5.2 Truth table of logic diagram (Fig. 5.4)

S	R	Q	$\overline{Q}$
0	0	Race	Race
0	1	0	1
1	0	1	0
1	1	Same as previous	Same as previous

Operation

1. When $S = 0$ and $R = 0$, then both the outputs of gate G_1 and gate G_2 try to become 1, which is not possible. There is a race between the outputs.
2. When $S = 0$ and $R = 1$, then the output of G_2 is 1 and that of G_1 is 0 ($Q = 0$) and the memory is said to be reset.
3. When $S = 1$ and $R = 0$, then the output of G_2 is 0 and that of G_1 is 1 ($Q = 1$) and the memory is said to be set.
4. When $S = 1$ and $R = 1$, then the output remains unchanged.

According to inputs S and R, the data logic 0 or logic 1 can be loaded or stored into the logic circuit. We can say that the data is latched into the logic circuit; hence, the logic circuit shown in Fig. 5.4 is also known as a NAND latch.

5.2.3 1-bit Memory Cell Using NOR Gates

The logic diagram of a 1-bit memory cell using NOR gate is shown in Fig. 5.5 (a) and the truth table is given in Table 5.3.

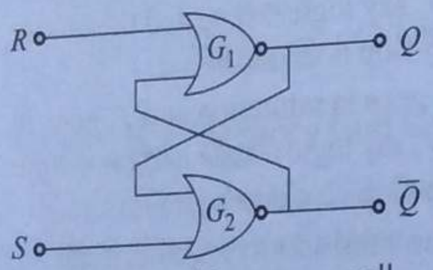


Fig. 5.5(a) 1-bit memory cell using NOR gates

Table 5.3 Truth table of logic diagram (Fig. 5.5(a))

S	R	Q	$\overline{Q}$
0	0	Same as previous	Same as previous
0	1	0	1
1	0	1	0
1	1	Race	Race

Operation

1. When $S = 0$ and $R = 0$, then the output remains unchanged.
2. When $S = 0$ and $R = 1$, then the output of G_1 is 0 ($Q = 0$) and that of G_2 is 1 and the memory is reset.
3. When $S = 1$ and $R = 0$, then the output of G_1 is 1 ($Q = 1$) and that of G_2 is 0 and the memory is set.

4. When $S = 1$ and $R = 1$, then the outputs of both the gates try to become 0, which is not possible, and there is a race between the outputs.

The logic circuit shown in Fig. 5.5(a) is also known as a NOR latche. The symbol of a simple latch with control inputs S and R is shown in Fig. 5.5 (b).

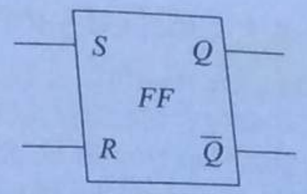


Fig. 5.5(b) Symbol of S-R flip-flop

5.3 CLOCKED S-R FLIP-FLOP

The simple S-R flip-flop will be set or reset any time by changing the inputs S and R. Most of the applications demand the memory cell to set or reset in synchronous with the clock. The clock is nothing but a train of pulses. The clocked S-R flip-flop is shown in Fig. 5.6.

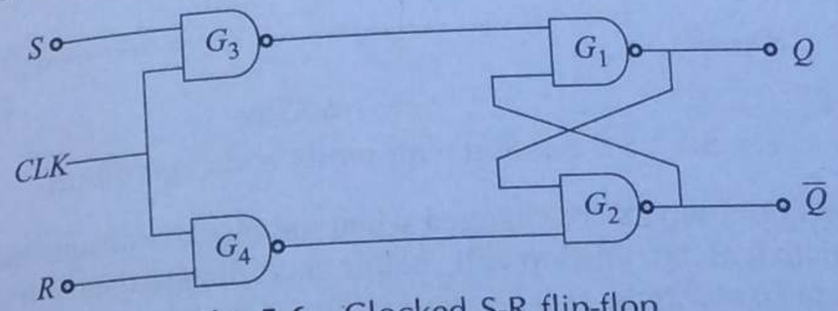


Fig. 5.6 Clocked S-R flip-flop

Operation

When the clock is absent (CLK = 0), the outputs of both the gates G_3 and G_4 are 1 and there is no change in the outputs of G_1 and G_2. When the clock is present (CLK = 1), the outputs of G_3 and G_4 are the function of the inputs S and R.

1. When $S = 0$ and $R = 0$, then the outputs of G_3 and G_4 are 1 ($R' = 1$ and $S' = 1$), the outputs of G_1 and G_2 are unchanged.
2. When $S = 0$ and $R = 1$, then the output of G_3 is 1 and that of G_4 is 0 ($R' = 1$ and $S' = 0$), the output of G_1 is 0 and that of G_2 is 1. The flip-flop is reset.
3. When $S = 1$ and $R = 0$, then the output of G_3 is 0 and that of G_4 is 1 ($R' = 0$ and $S' = 1$), the output of G_1 is 1 and that of G_2 is 0. The flip-flop is set.
4. When $S = 1$ and $R = 1$, then the outputs of G_3 and G_4 are 0 ($R' = 0$ and $S' = 0$), and the outputs of G_1 and G_2 try to become 1, which is not possible. There is a race and the output is undefined and it is known as *forbidden state*.

The truth table of S-R flip-flop with clock is given in Table 5.4. This state is represented as '?' in Table 5.4, where S_n and R_n are the present inputs, Q_n is the present output, and Q_{n+1} is the output after the clock. The clocked S-R flip-flop is simply referred as S-R flip-flop.

Table 5.4 Truth table of S-R flip-flop

Input		*Output*
S_n	R_n	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	?

5.3.1 Clocked S-R Flip-Flop With Preset and Clear Inputs

When the flip-flop is switched ON, the output state of the flip-flop is uncertain, i.e. output Q is either set or reset. In some of the applications, the initial output of the flip-flop should be fixed, it may be 0 or 1. There is a need to use additional inputs to set or reset the initial output. Preset and clear are the additional asynchronous inputs used to set the initial condition of the flip-flop.

The logic diagram of an S-R flip-flop with preset and clear inputs is shown in Fig 5.7.

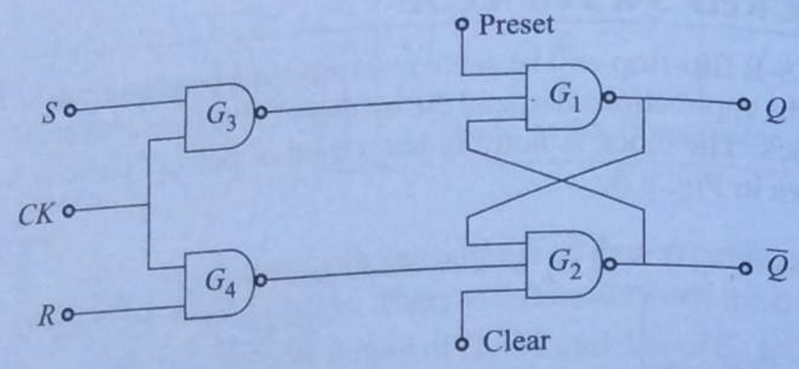


Fig. 5.7 S-R flip-flop with preset and clear inputs

The set and reset inputs are changed when the clock is absent and when it is present, the output of the flip-flop will change according to the truth table. Hence these inputs are known as the *synchronous inputs* of flip-flop. The preset and clear inputs can change at any time and at the same instant, the output of the flip-flop is changed. It is independent of the clock input, and hence these inputs are known as *asynchronous inputs*.

1. When preset = 0 and clear = 1, then the output of G_1 is 1 and that of G_2 is 0, which is independent of inputs S and R, and the flip-flop is set.
2. When preset = 1 and clear = 0, then the output of G_2 is 1 which makes the output of G_1 is 0 (i.e. $Q = 0$), which is independent of inputs S and R, and the flip-flop is reset.
3. When preset = 1 and clear = 1, then outputs of G_1 and G_2 depend on other inputs. For normal operation, preset and clear are connected to logic 1.
4. When preset = 0 and clear = 0, then outputs of G_1 and G_2 try to become 1. Here, the uncertain state occurs and hence, preset = 0 and clear = 0 is not used.

The symbol of S-R flip-flop with preset and clear inputs is shown in Fig. 5.8 and the truth table is given in Table 5.5.

Table 5.5 Truth table of S-R flip-flop with preset and clear inputs

Input				*Output*
S_n	R_n	*Pr*	*Cr*	Q_{n+1}
X	X	0	1	1
X	X	1	0	0
0	0	1	1	Q_n
0	1	1	1	0
1	0	1	1	1
1	1	1	1	?

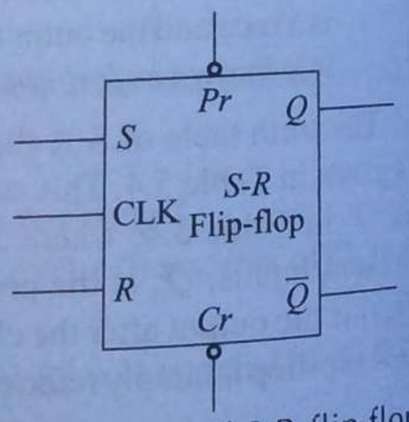


Fig. 5.8 Symbol of S-R flip-flop with preset and clear inputs

Bubbles on preset and clear inputs show that these inputs are active low.

In S-R flip-flop (using NAND gate), when $S = 1, R = 1$, the outputs Q and $\overline{Q}$ are same, logically it is not possible. Such state is referred as forbidden state. This is one of the important limitations of S-R flip-flop.

5.4 J-K FLIP-FLOP

The limitation of S-R flip-flop is overcome in J-K flip-flop. The logic diagram of J-K flip-flop is shown in Fig. 5.9.

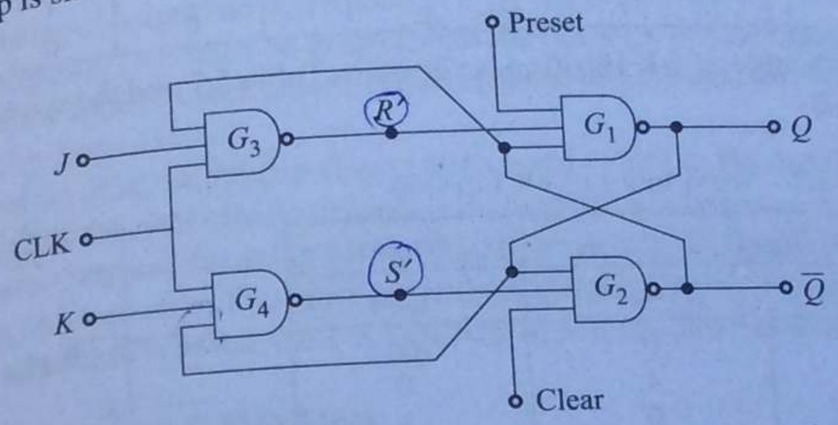


Fig. 5.9 Logic diagram of J-K flip-flop

The outputs of the flip-flop are connected to the inputs.

Operation

When the clock is absent (CLK = 0), the outputs of G_3 and G_4 are 1 and there is no change in the outputs of G_1 and G_2. When the clock is present (CLK = 1), the outputs of G_3 and G_4 are the function of inputs J and K.

1. When $J = 0$ and $K = 0$, then the outputs of G_3 and G_4 are 1, independent of other inputs, i.e. $R' = 1$ and $S' = 1$, and the outputs of G_1 and G_2 are unchanged.
2. When $J = 0$ and $K = 1$, then the output of G_3 is 1 and that of G_4 is 0, i.e. $R' = 1$ and $S' = 0$, and the output $Q = 0$ and $\overline{Q} = 1$, and the flip-flop is said to be reset.
3. When $J = 1$ and $K = 0$, then the output of G_3 is 0 and that of G_4 is 1, i.e. $R' = 0$ and $S' = 1$, and the output $Q = 1$ and $\overline{Q} = 0$, and the flip-flop is said to be set.
4. When $J = 1$ and $K = 1$, then the outputs of G_3 and G_4 are the functions of other inputs.
   (i) When $Q = 0$ and $\overline{Q} = 1$, then the output of $G_3 = 0$ and that of $G_4 = 1$, i.e. $R' = 0$ and $S' = 1$, and the output $Q = 1$ and $\overline{Q} = 0$.
   (ii) When $Q = 1$ and $\overline{Q} = 0$, then the output of $G_3 = 1$ and that of $G_4 = 0$, i.e. $R' = 1$ and $S' = 0$. Hence the output $Q = 0$ and $\overline{Q} = 1$.

In short, when $J = 1$ and $K = 1$, the present output is the complement of previous output. The operation summary of J-K flip flop is given in Table 5.6.

Table 5.6 Truth table of J-K flip-flop

Inputs		*Previous output*		*Input of latch*		*Output*	
J	*K*	Q_n	$\overline{Q}_n$	*R′*	*S′*	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	0	1	1	1	0	1
0	0	1	0	1	1	1	0
0	1	X	X	1	0	0	1
0	1	X	X	0	1	1	0
1	0	0	1	0	1	1	0
1	1	1	0	1	0	0	1
1	1						

The truth table of J-K flip-flop is shown in Table 5.7 and the symbol is shown in Fig. 5.10.

Table 5.7 Truth table of J-K flip-flop

Input		*Output*
J_n	K_n	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

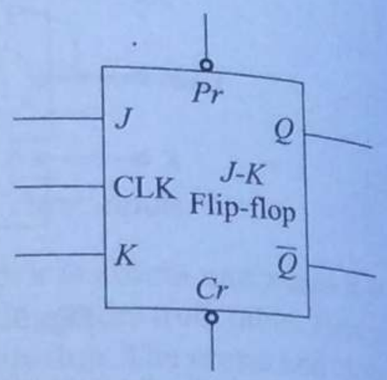


Fig. 5.10 The symbol of J-K flip-flop

Similar to the S-R flip-flop, the J-K flip-flop has preset and clear asynchronous inputs which sets or resets the flip-flop initially, as given in Table 5.8.

Table 5.8 Preset and Clear inputs set or reset the flip-flop

Input		*Output*
Pr	*Cr*	Q_n
0	0	Not used
0	1	Set
1	0	Reset
1	1	Normal operation

5.4.1 Race-around Condition

The truth table of J-K flip-flop shows that when $J_n = 1$ and $K_n = 1$, then $Q_{n+1} = \overline{Q}_n$. That is for $J_n = 1$ and $K_n = 1$, the J-K flip-flop acts as a *toggle switch*. Consider the input $J_n = 1$ and $K_n = 1$ and output $Q = 0$; after the propagation delay Δt of flip-flop (it is twice the delay of NAND gate), the output will change from 0 to 1. Since in a J-K flip-flop, output is connected to inputs, this output acts as input, and after the next delay Δt, the output will change from 1 to 0. This process is continued and at the end of the applied clock pulse, the output is uncertain. This situation is known as the *race-around condition*, as shown in Fig. 5.11 where T is the duration of pulse.

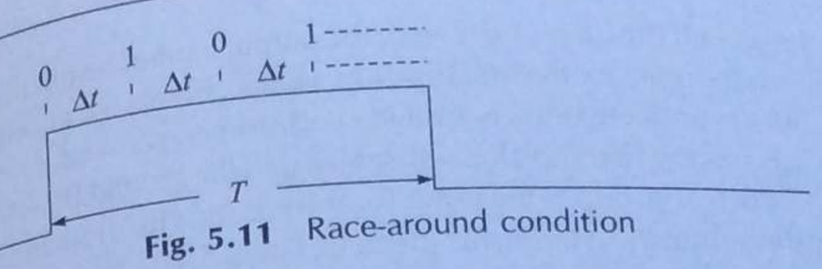


Fig. 5.11 Race-around condition

The race-around condition is a major problem in J-K flip-flop. It is not a problem for the flip-flop where the input does not change during the clock. But in case of J-K flip-flop, because of the feedback, the input changes during the clock pulse. This uncertainty of output can be avoided, if the delay of the flip-flop is increased. Delay of the flip-flop (Δt) must be greater than the duration of clock pulse (T) or the duration of the applied clock pulse (T) gets reduced and it becomes less than the delay of the flip-flop (Δt).

To increase the delay of the flip-flop is not a good solution. By increasing the delay, the speed of the system is decreased. It is also difficult to reduce the delay of the applied pulse beyond the delay of flip-flop because the propagation delay of flip-flop is in nanoseconds. The most practical solution to avoid the problem of race-around condition is to use the J-K flip-flop in Master-and-Slave mode.

5.4.2 Master-slave J-K Flip-Flop

The Master-Slave J-K flip-flop using S-R flip-flop is shown in Fig. 5.12.

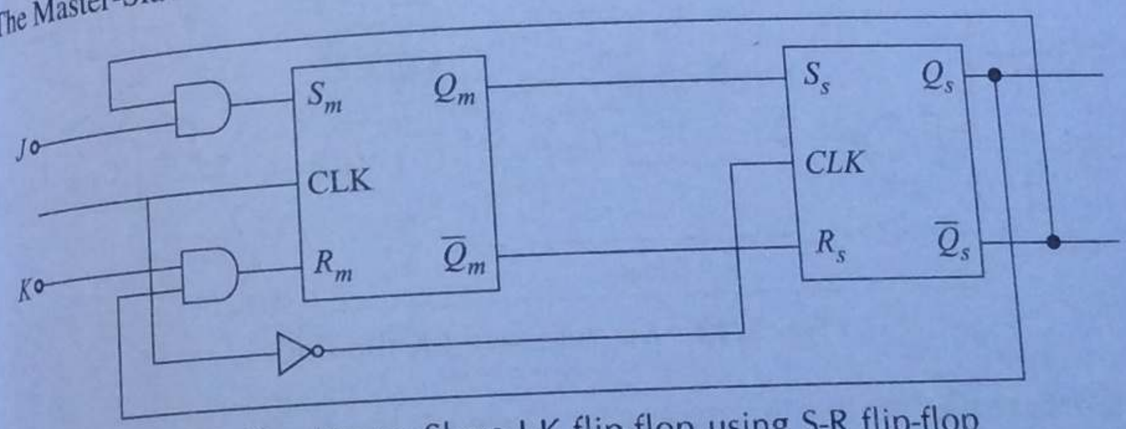


Fig. 5.12 Master-Slave J-K flip-flop using S-R flip-flop

The output of second S-R flip-flop is given to the input of the first S-R flip-flop through AND gate. $S_m = J_n\,\overline{Q}_n$ and $R_m = K_n Q_n$. The clock is directly applied to the first S-R flip-flop after the NOT is applied to the second S-R flip-flop. When CLK = 1, the first flip-flop is enabled and the second flip-flop is disabled. The output of first flip-flop changes as per the truth table of S-R flip-flop.

Table 5.9 Operation of first flip-flop

J_n	K_n	S_{m_n}	R_m	$Q_{m_{n+1}}$
0	0	0	0	Q_n
0	1	0	Q_n	0
1	0	$\overline{Q}_n$	0	1
1	1	$\overline{Q}_n$	Q_n	$\overline{Q}_n$

Since the second flip-flop is disabled, the output cannot be changed during the clock, which is the input for the first flip-flop. Hence, inputs are not changed during the clock and the problem of race-around condition is resolved.

When CLK = 0, the first flip-flop is disabled and the second flip-flop is enabled. The output of first flip-flop is the input for the second flip-flop and the output of second flip-flop follows the input as given in Table 5.10.

Table 5.10 Operation of second flip-flop

Q_{m_n}	$\overline{Q}_{m_n}$	S_{s_n}	R_{s_n}	$\overline{Q}_{n+1}$
0	1	0	1	0
1	0	1	0	1

Thus, the output of first flip-flop changes as per the truth table of S-R flip-flop during the positive cycle and it is given to the output of second flip-flop during the negative cycle. Hence, the first flip-flop is known as Master and the second is known as Slave. The logic diagram of Master-Slave J-K flip-flop is shown in Fig. 5.13.

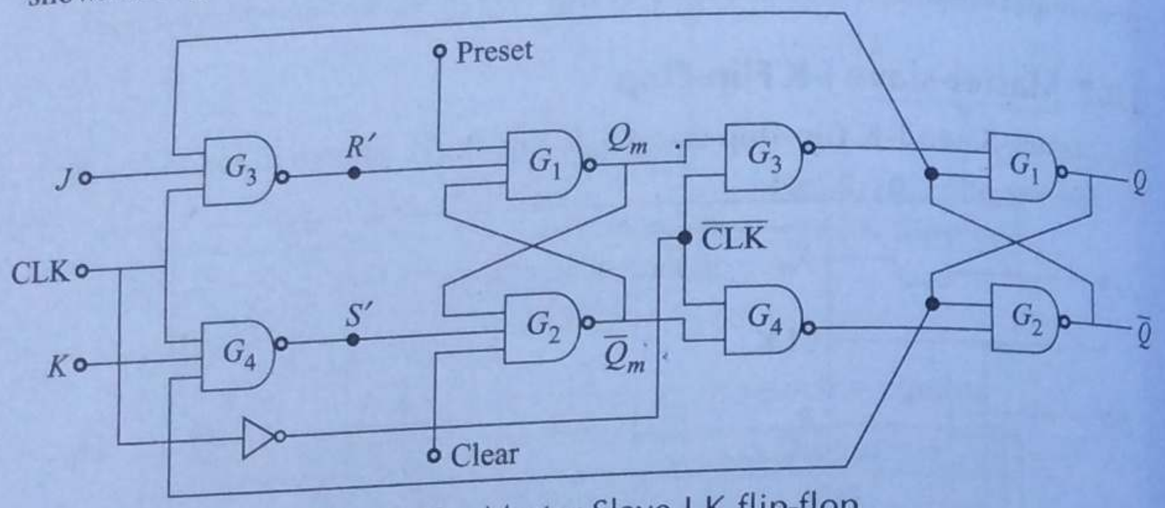


Fig. 5.13 Master-Slave J-K flip-flop

Similar to the J-K flip-flop, the Master-Slave J-K flip-flop has two asynchronous inputs, preset and clear. These inputs are set to the initial condition of Master-Slave J-K flip-flop. The symbol of Master-Slave J-K flip-flop is shown in Fig. 5.14 and the truth table is given in Table 5.11.

Table 5.11 Truth table of master-slave J-K flip-flop

Input		*Output*
J_n	K_n	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

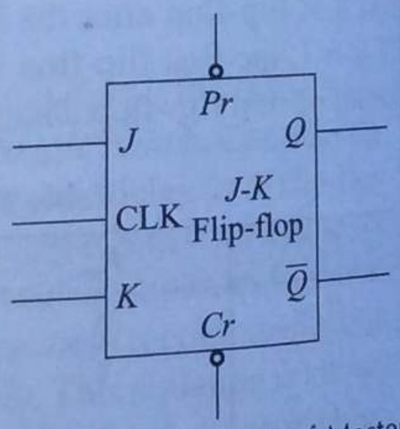


Fig. 5.14 Symbol of Master-Slave J-K flip-flop

5.5 D FLIP-FLOP

The second and third rows of the truth table of J-K flip-flop show that $Q_{n+1} = J_n$, under the condition $K_n = \overline{J}_n$. When K_n is complement of J_n, then the output is same as J; it is delay by the amount of clock. Such a flip-flop is known as delay flip-flop or D flip-flop. The D flip-flop is obtained from the J-K flip-flop by connecting a NOT gate between the J and K input of the J-K flip-flop, as shown in Fig. 5.15.

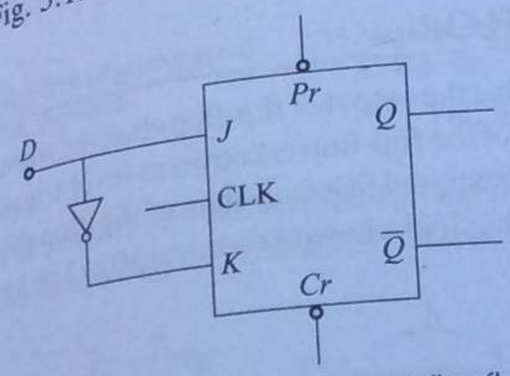


Fig. 5.15 D flip-flop using J-K flip-flop

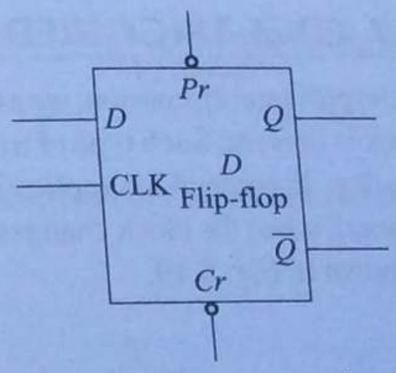


Fig. 5.16 Symbol of D flip-flop

Figure 5.16 shows the symbol of D flip-flop and the truth table is given in Table 5.12.

Table 5.12 Truth table of D flip-flop

Input D_n	*Output* Q_{n+1}
0	0
1	1

5.6 T FLIP-FLOP

The T flip-flop has a single input, the output of T flip-flop is same to the previous output or complement of the previous output. The truth table of J-K flip-flop shows that $Q_{n+1} = Q_n$ or $Q_{n+1} = \overline{Q}_n$, under the condition $J_n = K_n$. Hence, a T flip-flop is obtained from J-K flip-flop, by shorting J and K inputs of the J-K flip-flop. The T flip-flop using J-K flip-flop is shown in Fig. 5.17.

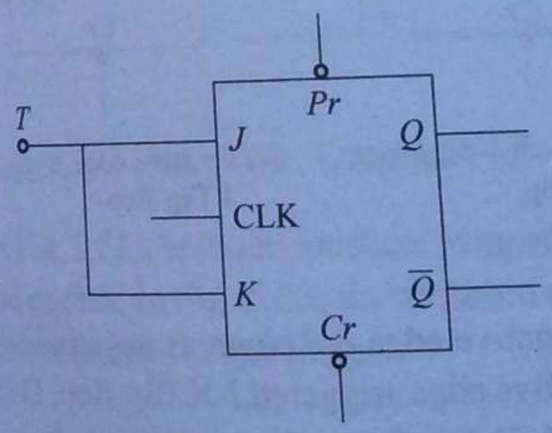


Fig. 5.17 T flip-flop using J-K flip-flop

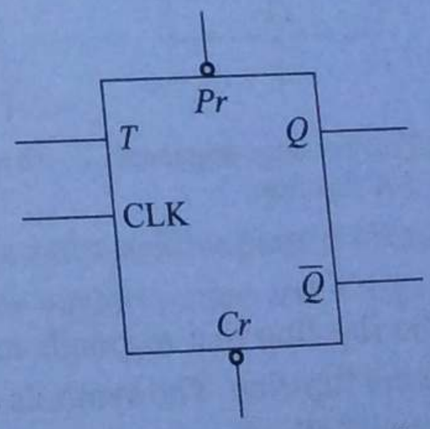


Fig. 5.18 Symbol of T flip-flop

Fig. 5.18 shows the symbol of T flip-flop and the truth table is given in Table 5.13.

Table 5.13 Truth table of T flip-flop

Input T	*Output Q_{n+1}*
0	Q_n
1	$\overline{Q_n}$

5.7 EDGE-TRIGGERED FLIP-FLOP

In the previous discussion, we assumed that the output of flip-flop changes when the clock is present. Such type of triggering of the flip-flop is known as level-triggered flip-flop. In some of the applications, it is required that the output of flip-flop should respond, when the clock changes state. The clock changes state from 0 to 1 or 1 to 0, as shown in Fig. 5.19.

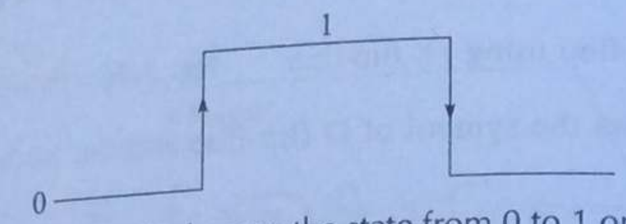
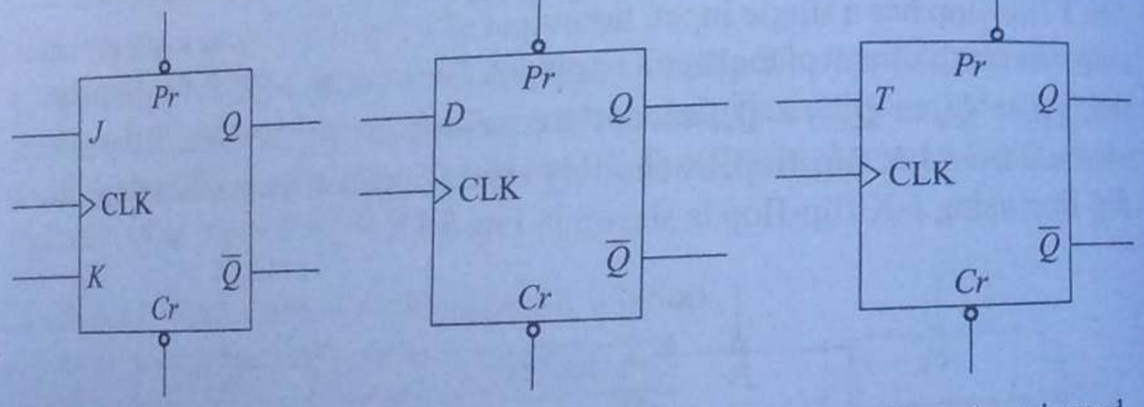


Fig. 5.19 Clock changes the state from 0 to 1 or 1 to 0

The change of state from 0 to 1 is known as *positive edge* and the change of state from 1 to 0 is known as *negative edge*.

The flip-flop that responds for the positive edge is known as positive edge-triggered flip-flop. The symbols of positive edge-triggered J-K flip-flop, D flip-flop, and T flip-flop are shown in Figs. 5.20 (a), (b), and (c), respectively.



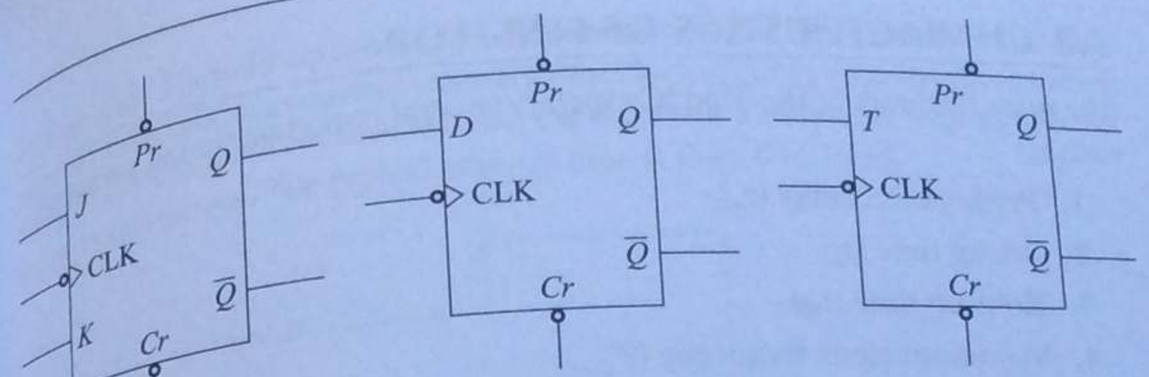
(a) Positive edge-triggered J-K flip-flop

(b) Positive edge-triggered D flip-flop

(c) Positive edge-triggered T flip-flop

Fig. 5.20

The flip-flop that responds to the negative edge is known as negative edge-triggered flip-flop. The symbols of negative edge-triggered J-K flip-flop, D flip-flop, and T flip-flop are shown in Figs. 5.21 (a), (b), and (c), respectively.



(a) Negative edge-triggered J-K flip-flop

(b) Negative edge-triggered D flip-flop

(c) Negative edge-triggered T flip-flop

Fig. 5.21

The circuits that convert the clock pulse into positive edge and negative edge are shown in Figs. 5.22 and 5.23, respectively.

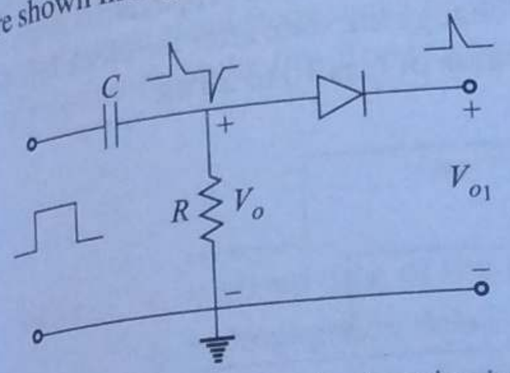


Fig. 5.22 Converts the clock pulse into positive edge

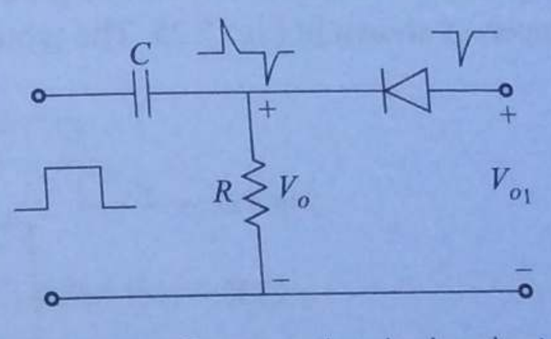


Fig. 5.23 Converts the clock pulse into negative edge

The R-C circuit acts as a differentiator. When the pulse is applied as an input to the differentiator, the positive going pulse and the negative going pulse with small duration are available at the output, as shown in Fig. 5.24. The duration of pulse is a function of resistor and capacitor values.

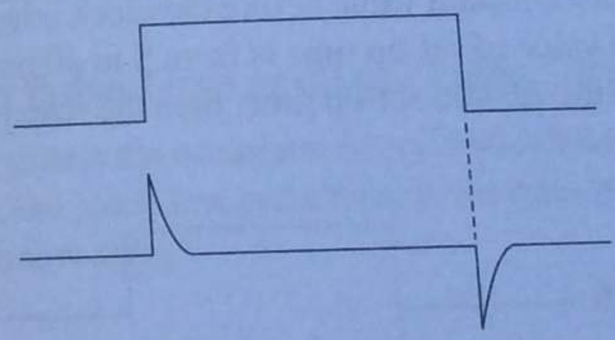


Fig. 5.24 Output of differentiator for pulse input

In Fig. 5.22, the diode conducts for positive edge and a positive pulse is obtained at the output. In Fig. 5.23, the diode conducts for negative edge and a negative pulse is obtained at the output.

5.8 CHARACTERISTICS OF FLIP-FLOP

The manufacturers of flip-flop ICs specify several important timing parameters, such as:

1. Propagation delay (t_p)
2. Set-up time (t_s)
3. Hold-up time (t_H)
4. Maximum clock frequency (F_{max})
5. Asynchronous active pulse width

5.8.1 Propagation Delay

We have defined the propagation delay for logic families. The same definition is applicable for flip-flops. However, there is one exception—the output of flip-flop starts to change when the clock hits at the input. Hence, the propagation delay of flip-flop is the amount of time required to change the state after the clock hits the input, as shown in Fig. 5.25. The typical value of t_p is 10 to 20 ns.

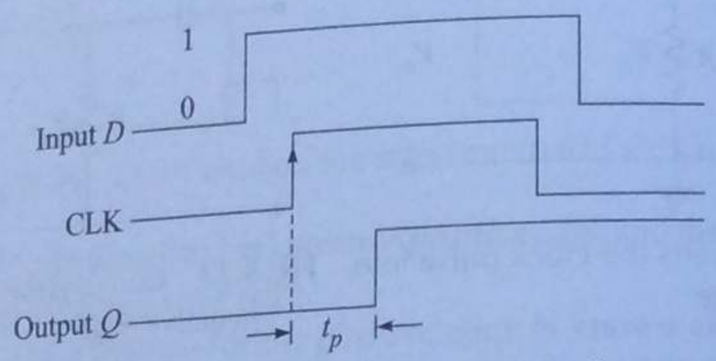


Fig. 5.25 Propagation delay of flip-flop

5.8.2 Set-up Time

The set-up time is defined as the minimum amount of time for which the input must be maintained at a constant value before the clock edge arrives, as shown in Fig 5.26. The typical value of set-up time is from 5 to 40 ns. If the set-up time of the signal is less than the desired set-up time, then the response of flip-flop is not reliable.

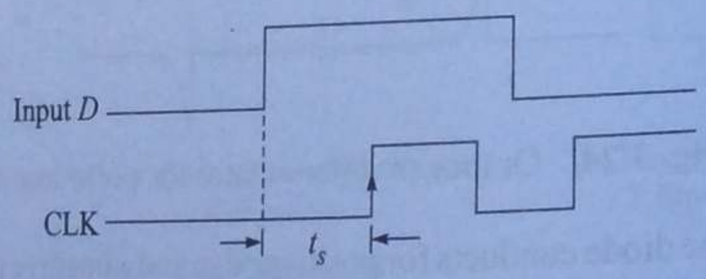


Fig. 5.26 Set-up time

5.8.3 Hold-up Time

In positive edge-triggered flip-flops, the output changes with the rising edge of the clock (0 to 1). It does not mean that after the rising edge of clock, the input signal can be changed immediately. The input signal should be held at least for the

holding time. The hold-up time is the minimum amount of time for which the input must be maintained at a constant value after the clock edge has been struck, as shown in Fig. 5.27. The main cause of holding the input is the switching time of internal resistors and diodes. The typical hold-up time is from 0 to 10 ns.

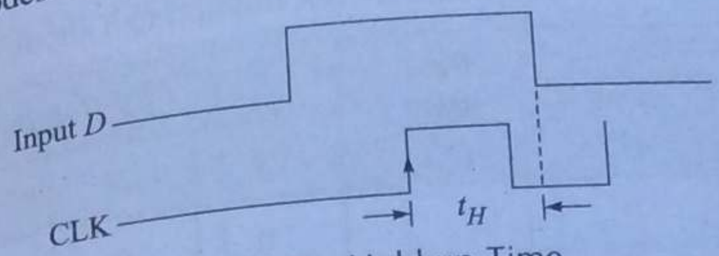


Fig. 5.27 Hold-up Time

5.8.4 Maximum Clock Frequency

The selection of clock frequency is a function of propagation delay and set-up time. The maximum clock frequency that can be applied as a CLK input for proper and stable operation of the flip-flop is given as

$$f_{max} = \frac{1}{t_s + t_p + t_{np}}$$

where t_s = set-up time of flip-flop
t_p = propagation delay time of flip-flop
t_{np} = propagation of next stage decoder

5.8.5 Asynchronous Active Pulse Width

Preset and clear are active low asynchronous inputs. These inputs are set to the initial condition of flip-flop by applying a negative pulse to one of the inputs. Asynchronous active pulse width is the minimum duration of negative pulse, which is required to set the initial conditions of flip-flop.

5.8.6 Clock High Pulse Time and Low Pulse Time

The clock high pulse time is the minimum time for which the clock must remain in high state. Similarly, the clock low pulse time is the minimum time for which the clock must remain in low state.

5.9 FLIP-FLOP CONVERSIONS

It is possible to convert one flip-flop into another with some additional circuit. The flip-flop conversion includes the following steps:

Step 1: Write the excitation table.

Step 2: Simplify the excitation table using K-map.

Step 3: Draw the desired logic diagram.

5.9.1 S-R Flip-Flop to T Flip-Flop

Step 1: The excitation table for S-R flip-flop to T flip-flop conversion is given in Table 5.14.

Table 5.14 Excitation table for S-R flip-flop to T flip-flop conversion

Input *T*	*Present state* Q_n	*Next state* Q_{n+1}	*Flip-flop inputs*			
			S	*R*	*S*	*R*
0	0	0	0	0	0	X
			0	1		
0	1	1	0	0	X	0
			1	0		
1	0	1	1	0	1	0
1	1	0	0	1	0	1

Step 2: The excitation table is simplified using K-map.

K-map for S

Q_n \ T	0	1
0	0	(1)
1	×	0

$S = T\overline{Q}_n$

K-map for R

Q_n \ T	0	1
0	×	0
1	0	(1)

$R = TQ_n$

Step 3: The logic diagram of T flip-flop using S-R flip-flop is shown in Fig. 5.28.

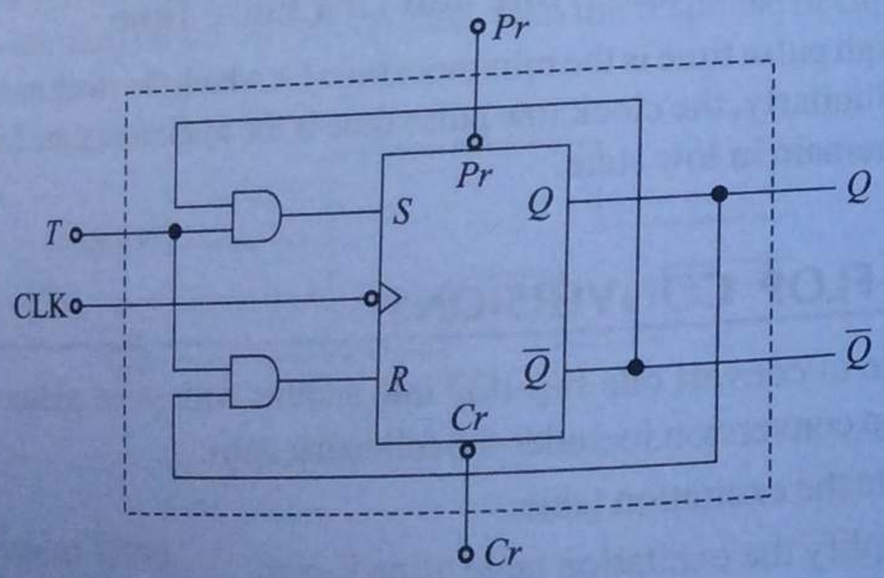


Fig. 5.28 T flip-flop using S-R flip-flop

5.9.2 S-R Flip-Flop to D Flip-Flop

Step 1: The excitation table for S-R flip-flop to D flip-flop conversion is given in Table 5.15.

Table 5.15 Excitation table for S-R flip-flop to D flip-flop conversion

Input *D*	*Present state* Q_n	*Next state* Q_{n+1}	*Flip-flop inputs*			
			S	*R*	*S*	*R*
0	0	0	0	0	0	X
			0	1		
0	1	0	0	1	0	1
1	0	1	1	0	1	0
1	1	1	0	0	X	0
			1	0		

Step 2: The excitation table is simplified using K-map.

K-map for S

Q_n \ D	0	1
0	0	(1)
1	0	×

$S = D$

K-map for R

Q_n \ D	0	1
0	×	0
1	(1)	0

$R = \overline{D}$

Step 3: The logic diagram of D flip-flop using S-R flip-flop is shown in Fig. 5.29.

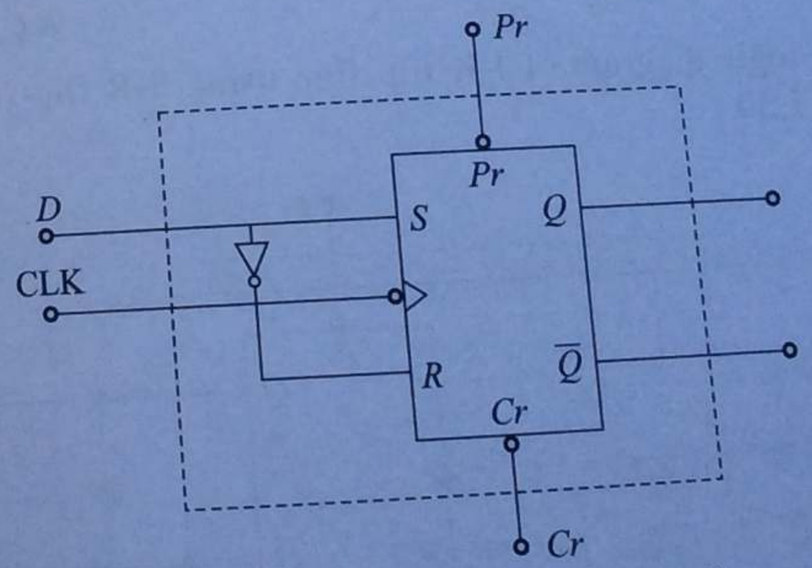


Fig. 5.29 D flip-flop using S-R flip-flop

5.9.3 S-R Flip-Flop to J-K Flip-Flop

Step 1: The excitation table for S-R flip-flop to J-K flip-flop conversion is given in Table 5.16.

Table 5.16 Excitation table for S-R flip-flop to J-K flip-flop conversion

Input		*Present state*	*Next state*	*Flip-flop inputs*			
J	*K*	Q_n	Q_{n+1}	*S*	*R*	*S*	*R*
0	0	0	0	0	0	0	X
				0	1		
0	0	1	1	0	0	X	0
				1	0		
0	1	0	0	0	0	0	X
				0	1		
0	1	1	0	0	1	0	1
1	0	0	1	1	0	1	0
1	0	1	1	0	0	X	0
				1	0		
1	1	0	1	1	0	1	0
1	1	1	0	0	1	0	1

Step 2: The excitation table is simplified using K-map.

K-map for *S*

Q_n \ JK	00	01	11	10
0	0	0	1	1
1	×	0	0	×

$S = J\overline{Q}_n$

K-map for *R*

Q_n \ JK	00	01	11	10
0	×	×	0	0
1	0	1	1	0

$R = K Q_n$

Step 3: The logic diagram of J-K flip-flop using S-R flip-flop is shown in Fig. 5.30.

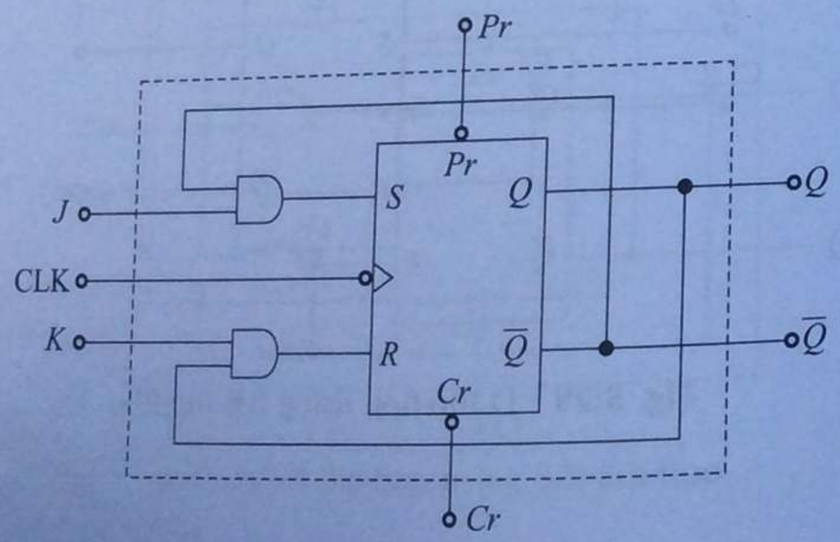


Fig. 5.30 J-K flip-flop using S-R flip-flop

5.9.4 T Flip-Flop to D Flip-Flop

Step 1: The excitation table for T flip-flop to D flip-flop conversion is given in Table 5.17.

Table 5.17 Excitation table for T flip-flop to D flip-flop conversion

Input *D*	*Present state* Q_n	*Next state* Q_{n+1}	*Flip-flop input* *T*
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Step 2: The excitation table is simplified using K-map.

K-map for *T*

Q_n \ D	0	1
0	0	(1)
1	(1)	0

$T = \overline{D}\,Q_n + D\overline{Q}_n$
$= D \oplus Q_n$

Step 3: The logic diagram of D flip-flop using T flip-flop is shown in Fig. 5.31.

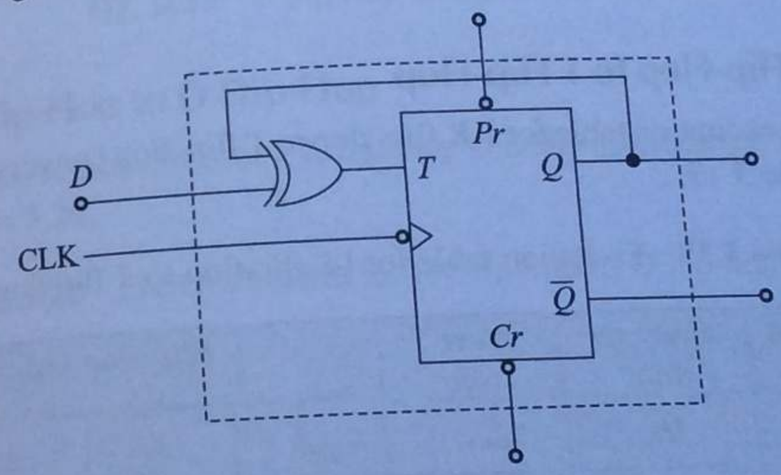


Fig. 5.31 D flip-flop using T flip-flop

5.9.5 D Flip-Flop to T Flip-Flop

Step 1: The excitation table for D flip-flop to T flip-flop conversion is given in Table 5.18.

Table 5.18 Excitation table for D flip-flop to T flip-flop conversion

Input *T*	*Present state* Q_n	*Next state* Q_{n+1}	*Flip-flop input* *D*
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Step 2: The excitation table is simplified using K-map.

K-map for D

Q_n \ T	0	1
0	0	(1)
1	(1)	0

$$D = \overline{T}\,Q_n + T\overline{Q}_n$$
$$= T \oplus Q_n$$

Step 3: The logic diagram of T flip-flop using D flip-flop is shown in Fig. 5.32.

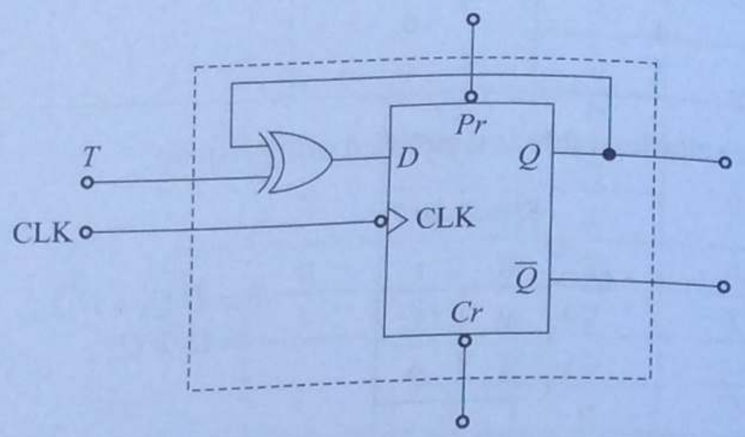


Fig. 5.32 T flip-flop using D flip-flop

5.9.6 J-K Flip-Flop to T Flip-Flop

Step 1: The excitation table for J-K flip-flop to T flip-flop conversion is given in Table 5.19.

Table 5.19 Excitation table for J-K flip-flop to T flip-flop conversion

Input T	*Present state* Q_n	*Next state* Q_{n+1}	*Flip-flop inputs* J	K	J	K
0	0	0	0	0	0	X
			0	1		
0	1	1	0	0	X	0
			1	0		
1	0	1	1	0	1	X
			1	1		
1	1	0	0	1	X	1
			1	1		

Step 2: The excitation table is simplified using K-map.

K-map for J

Q_n \ T	0	1
0	0	×
1	×	×

$J = T$

K-map for K

Q_n \ T	0	1
0	×	×
1	0	1

$K = T$

Step 3: The logic diagram of T flip-flop using J-K flip-flop is shown in Fig. 5.33.

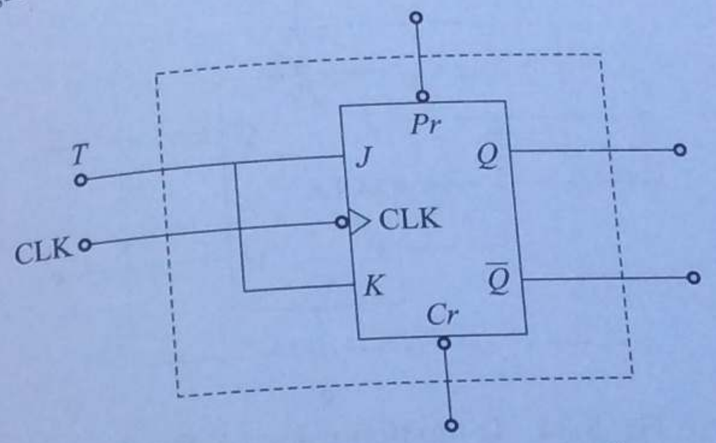


Fig. 5.33 T flip-flop using J-K flip-flop

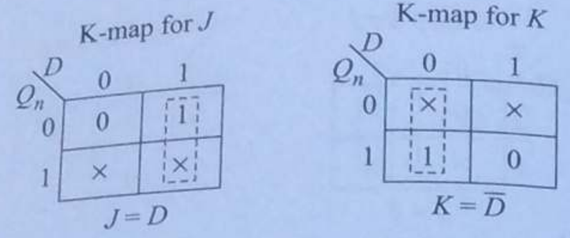
5.9.7 J-K Flip-Flop to D Flip-Flop

Step 1: The excitation table for J-K flip-flop to D flip-flop conversion is given in Table 5.20.

Table 5.20 Excitation table for J-K flip-flop to D flip-flop conversion

Input D	*Present state* Q_n	*Next state* Q_{n+1}	*Flip-flop inputs* J	K	J	K
0	0	0	0	0	0	X
			0	1		
0	1	0	0	1	X	1
			1	1		
1	0	1	1	0	1	X
			1	1		
1	1	1	0	0	X	0
			1	0		

Step 2: The excitation table is simplified using K-map.



Step 3 : The logic diagram of D flip-flop using J-K flip-flop is shown in Fig. 5.34.

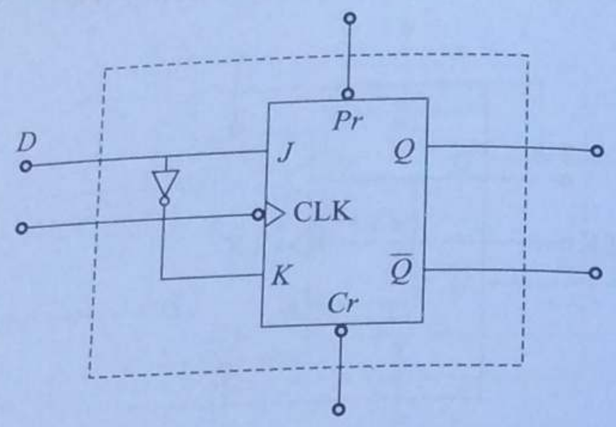


Fig. 5.34 D flip-flop using J-K flip-flop

Example 5.1 Draw the output waveform of J-K flip-flop for the input shown in Fig. 5.35

(i) if the flip-flop is positive edge-triggered

(ii) if the flip-flop is negative edge-triggered

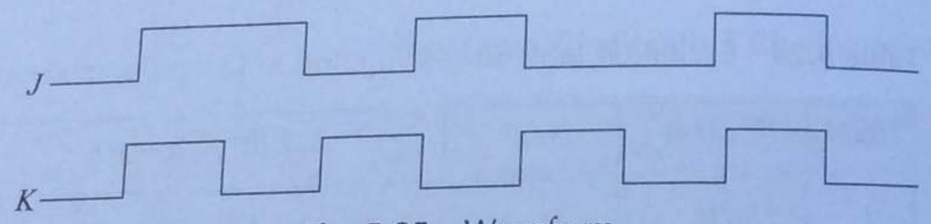


Fig. 5.35 Waveform

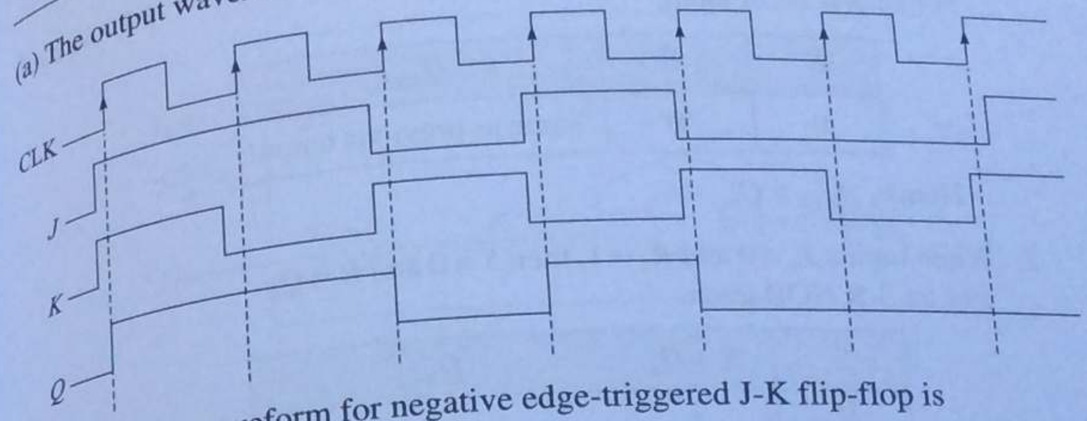
Solution

The truth table of J-K flip-flop is given in Table 5.21.

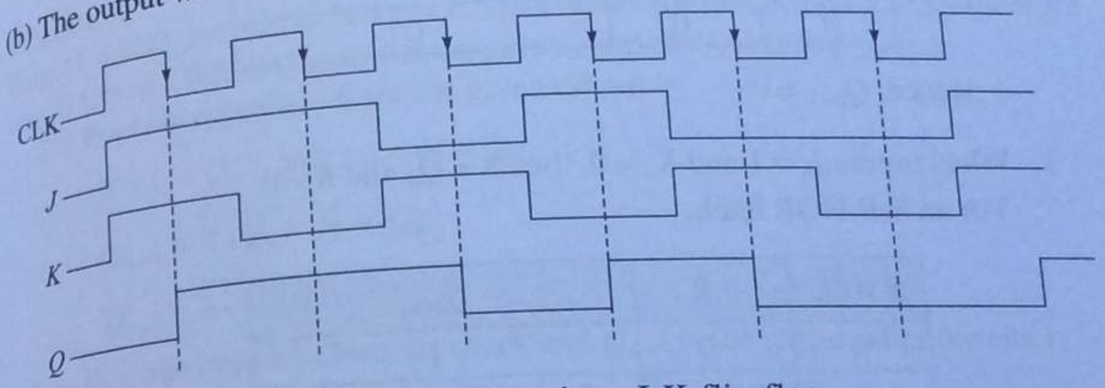
Table 5.21 Truth table of J-K flip-flop

Input		*Output*	*Description*
J_n	K_n	Q_{n+1}	
0	0	Q_n	Same as previous input.
0	1	0	Flip-flop is reset.
1	0	1	Flip-flop is set.
1	1	$\overline{Q}_n$	Output is complement of previous output.

(a) The output waveform for positive edge-triggered J-K flip-flop is



(b) The output waveform for negative edge-triggered J-K flip-flop is



Example 5.2 Find the output of the given J-K flip-flop.

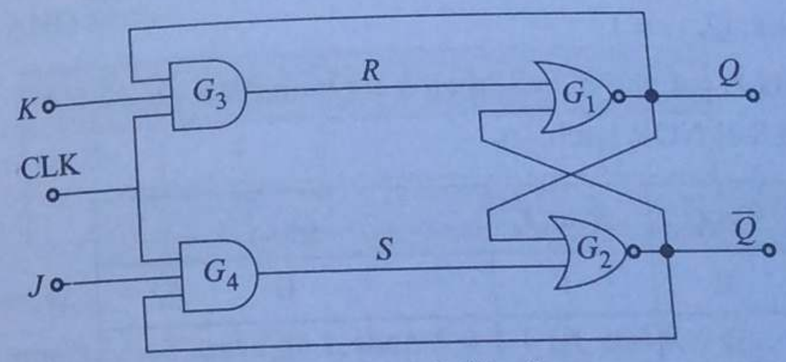


Fig. 5.36 J-K flip-flop

Solution

Here, G_1 and G_2 are the NOR gates. These gates are connected such that the logic circuit acts as a NOR latch. Assume that the output of G_3 is R and the output of G_4 is S.

From the given circuit, we can write the Boolean equations,

$$R = K_n Q_n \quad \text{and} \quad S = J_n \overline{Q}_n$$

$$Q_{n+1} = \overline{R + \overline{Q}_n} = \overline{R} \cdot Q_n$$

$$\overline{Q}_{n+1} = \overline{S + Q_n} = \overline{S} \cdot \overline{Q}_n$$

where Q_n is the output of the circuit before the clock and Q_{n+1} is the output after the clock.

1. When inputs $J_n = 0$ and $K_n = 0$, then $S = 0$ and $R = 0$.
   For an S-R NOR latch,

S	R	Q_{n+1}
0	0	Same as previous output

   Hence, $Q_{n+1} = Q_n$.

2. When inputs $J_n = 0$ and $K_n = 1$, then $S = 0$ and $R = Q_n$.
   For an S-R NOR latch,

S	$R = Q_n$	Q_{n+1}
0	1	0
0	0	0

   Hence, $Q_{n+1} = 0$.

3. When inputs $J_n = 1$ and $K_n = 0$, then $S = \overline{Q}_n$ and $R = 0$.
   For an S-R NOR latch,

$S = \overline{Q}_n$	R	Q_{n+1}
0	0	1
1	0	1

   Hence, $Q_{n+1} = 1$.

4. If inputs $J_n = 1$ and $K_n = 1$, then $S = \overline{Q}_n$ and $R = Q_n$.
   For an S-R NOR latch,

$S = \overline{Q}_n$	$R = Q_n$	Q_{n+1}
0	1	0
1	0	1

   Hence, $Q_{n+1} = \overline{Q}_n$.

The truth table for the given logic circuit is given in Table 5.22.

Table 5.22 Truth table for given logic circuit

Input		*Output*
J	*K*	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_n$

Example 5.3 Find the outputs of the given logic circuit.

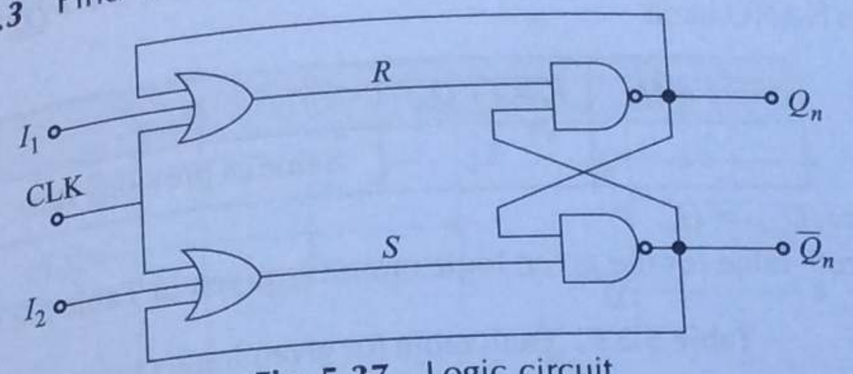


Fig. 5.37 Logic circuit

Solution
Here, G_1 and G_2 are the NAND gates. These gates are connected such that the logic circuit acts as a NAND latch. Assume the output of G_3 is R and G_4 is S.
The Boolean equations from the given circuit is

$$R = I_1 + Q_n \quad \text{and} \quad S = I_2 + \overline{Q}_n$$

$$Q_{n+1} = \overline{R \cdot \overline{Q}_n} = \overline{R} + Q_n$$

$$\overline{Q}_{n+1} = \overline{S \cdot Q_n} = \overline{S} + \overline{Q}_n$$

where Q_n is the output before the clock and Q_{n+1} is the output after the clock.

1. When inputs $I_1 = 0$ and $I_2 = 0$, then $S = \overline{Q}_n$ and $R = Q_n$.
   For a NAND latch,

$S = \overline{Q}_n$	$R = Q_n$	Q_{n+1}
0	1	0
1	0	1

   Hence, $Q_{n+1} = \overline{Q}_n$.

2. When inputs $I_1 = 0$ and $I_2 = 1$, then $S = 1 + \overline{Q}_n$ and $R = Q_n$.
   For a NAND latch,

$S = 1 + \overline{Q}_n$	$R = Q_n$	Q_{n+1}
1	0	1
1	1	1

   Hence, $Q_{n+1} = 1$.

3. When inputs $I_1 = 1$ and $I_2 = 0$, then $S = \overline{Q}_n$ and $R = 1 + Q_n$.
   For a NAND latch,

$S = \overline{Q}_n$	$R = 1 + Q_n$	Q_{n+1}
0	1	0
1	1	0

4. When inputs $I_1 = 1$ and $I_2 = 1$, then $S = 1 + \overline{Q}_n$ and $R = 1 + Q_n$.
For a NAND latch,

$S = 1 + \overline{Q}_n$	$R = 1 + Q_n$	Q_{n+1}
1	1	Same as previous state

Hence, $Q_{n+1} = Q_n$.
The truth table for the given logic circuit is given in Table 5.23.

Table 5.23 Truth table for given logic circuit

Input		*Output*
I_2	I_1	Q_{n+1}
0	0	$\overline{Q_n}$
0	1	0
1	0	1
1	1	Q_n

Example 5.4 Draw the logic circuit of S-R flip-flop using D flip-flop.

Solution

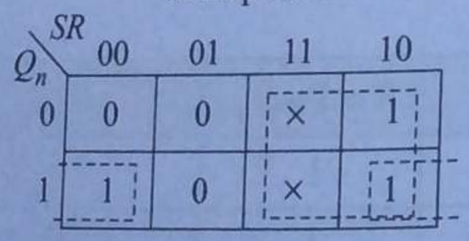
Step 1: The excitation table to convert the D flip-flop to S-R flip-flop is given in Table 5.24.

Table 5.24 Excitation table to convert D flip-flop to S-R flip-flop

Inputs		*Present State*	*Next level state*	*Flip-flop input*
S	*R*	Q_n	Q_{n+1}	*D*
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	X	X
1	1	1	X	X

Step 2: The excitation table is simplified using K-map.

K-map for *D*



$$D = S + \overline{R}Q_n$$

Step 3: The logic circuit of S-R flip-flop using D flip-flop is shown in Fig. 5.38.

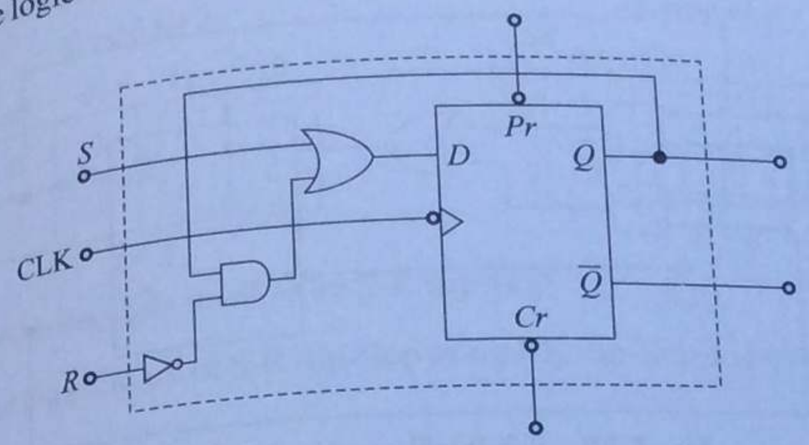


Fig. 5.38 S-R flip-flop using D flip-flop

Example 5.5 Draw the logic circuit of S-R flip-flop using T flip-flop.

Solution

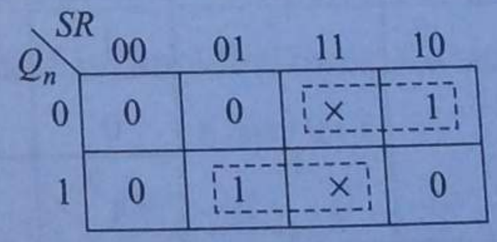
Step 1: The excitation table to convert the T flip-flop to S-R flip-flop is given in Table 5.25.

Table 5.25 Excitation table to convert T flip-flop to S-R flip-flop

Inputs		*Present State*	*Next level state*	*Flip-flop input*
S	*R*	Q_n	Q_{n+1}	*T*
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	X	X
1	1	1	X	X

Step 2: The excitation table is simplified using K-map.

K-map for *T*



$$T = S\,\overline{Q}_n + RQ_n$$

Step 3: The logic circuit of S-R flip-flop using T flip-flop is shown in Fig. 5.39.

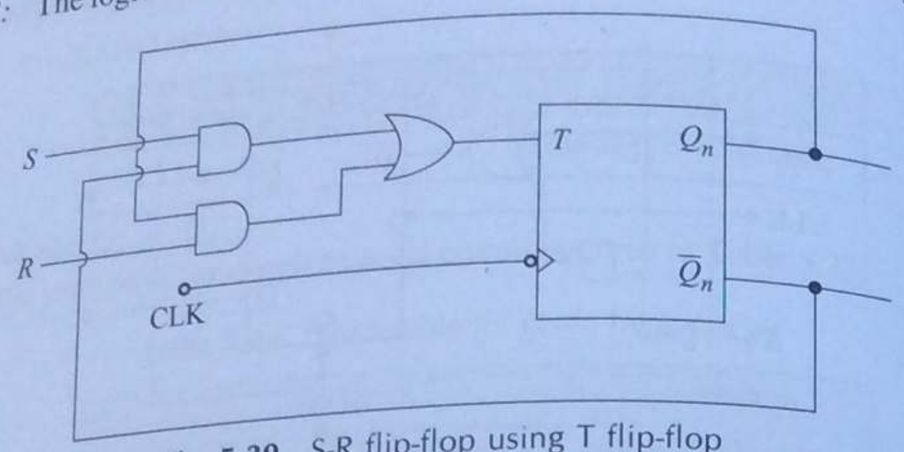


Fig. 5.39 S-R flip-flop using T flip-flop

Example 5.6 Draw the logic circuit of J-K flip-flop to S-R flip-flop.

Solution

Step 1: The excitation table to convert the J-K flip-flop to S-R flip-flop is given in Table 5.26.

Table 5.26 Excitation table to convert J-K flip-flop to S-R flip-flop

Input		*Present state*	*Next level state*	*Flip-flop inputs*			
S	*R*	Q_n	Q_{n+1}	*J*	*K*	*J*	*K*
0	0	0	0	0	0	0	X
				0	1		
0	0	1	1	0	0	X	0
				1	0		
0	1	0	0	0	0	0	X
				0	1		
0	1	1	0	1	1	X	1
				0	1		
1	0	0	1	1	1	1	X
				1	0		
1	0	1	1	0	0	X	0
				1	0		
1	1	0	X	X	X	X	X
1	1	1	X	X	X	X	X

Step 2: The excitation table is simplified using K-map.

K-map for *J*

Q_n \ SR	00	01	11	10
0	0	0	×	1
1	×	×	×	×

$J = S$

K-map for *K*

Q_n \ SR	00	01	11	10
0	×	×	×	×
1	0	1	×	0

$K = R$

The J-K flip-flop acts itself as an S-R flip-flop.

Step 3: The logic circuit of S-R flip-flop using J-K flip-flop is shown in Fig. 5.40.

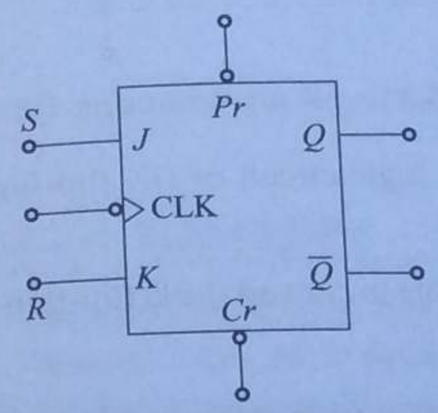


Fig. 5.40 S-R flip-flop using J-K flip-flop

Example 5.7 Draw the logic circuit of J-K flip-flop using T flip-flop.

Solution

Step 1: The excitation table to convert the J-K flip-flop to T flip-flop is given in Table 5.27.

Table 5.27 Excitation table to convert T flip-flop to S-R flip-flop

Inputs		*Present State*	*Next level state*	*Flip-flop input*
J	*K*	Q_n	Q_{n+1}	*T*
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	1

Step 2: The excitation table is simplified using K-map.

K-map for *T*

Q_n \ JK	00	01	11	10
0	0	0	1	1
1	0	1	1	0

$T = J\overline{Q}_n + KQ_n$

Step 3: The logic circuit of J-K flip-flop using T flip-flop is shown in Fig. 5.41.

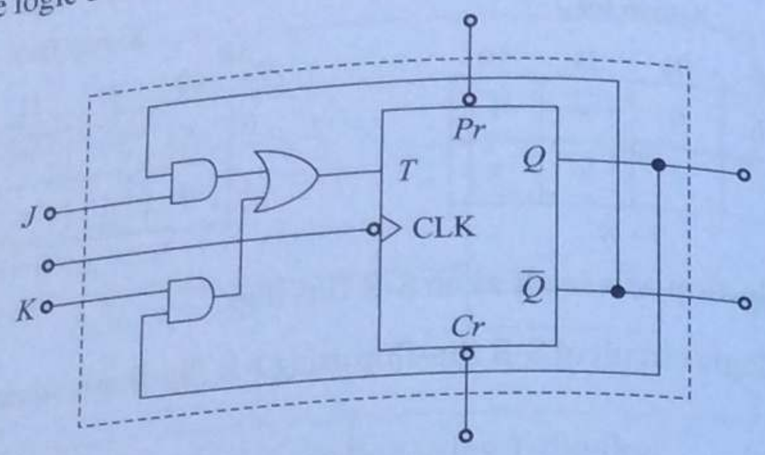


Fig. 5.41 J-K flip-flop using T flip-flop

Example 5.8 Draw the logic circuit of J-K flip-flop using D flip-flop.

Solution

Step 1: The excitation table to convert the D flip-flop to J-K flip-flop is given in Table 5.28.

Table 5.28 Excitation table to convert D flip-flop to J-K flip-flop

Inputs		*Present State*	*Next level state*	*Flip-flop input*
J	*K*	Q_n	Q_{n+1}	*D*
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

Step 2: The excitation table is simplified using K-map.

K-map for *D*

Q_n \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$$D = \overline{K}Q_n + J\overline{Q}_n$$

Step 3: The logic circuit of J-K flip-flop using D flip-flop is shown in Fig. 5.42.

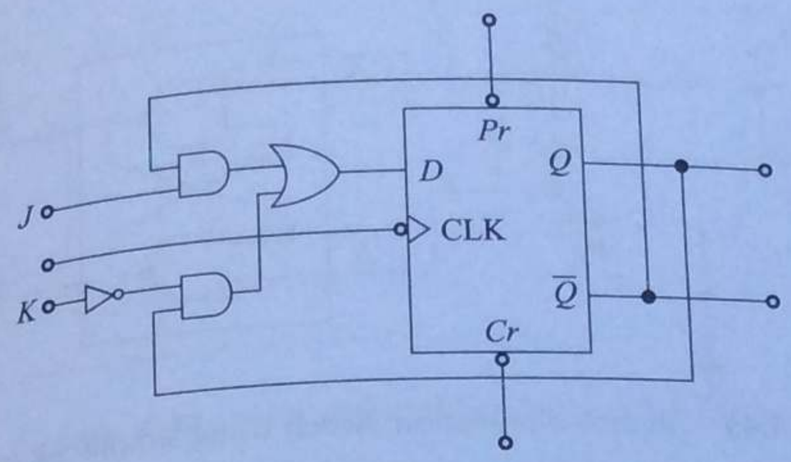


Fig. 5.42 J-K flip-flop using D flip-flop

5.10 APPLICATIONS OF FLIP-FLOPS

In the previous section, we have studied all types of flip-flops and the techniques to convert one flip-flop to another. Now, let us discuss the applications of flip-flops. The flip-flop can be used in (i) Bounce elimination switch, (ii) Registers, (iii) Counters, and (iv) Random Access Memory (RAM).

5.10.1 Bounce Elimination Switch

In digital electronics, the inputs are given as logic 0 or logic 1. These inputs can be generated by the oscillator or by the mechanical switch. Switch bouncing is one of the important problems when mechanical switches are interfaced to the microprocessor system. That is, when the switch is thrown from one position to another, it bounces several times and the system gets incorrect data.

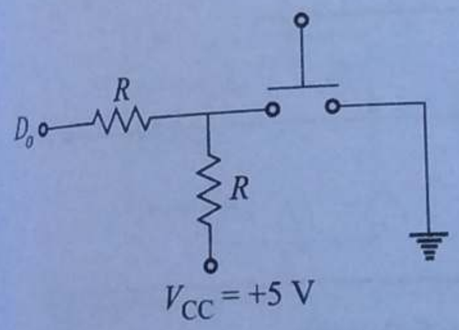


Fig. 5.43 Switching circuit

Consider Fig. 5.43. When the switch is pressed, the data given to the system is either 0 or 1.

When the pressure is released by the user, because of spring effect of switch, the input changes between 0 and 1, and at last becomes 1 as shown in Fig. 5.44. This problem is impossible to avoid in a mechanical switch. The flip-flop is used to avoid the bouncing problem. The bounce elimination switch using a simple S-R latch is shown in Fig. 5.45.

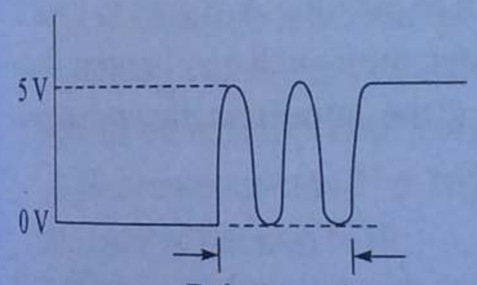


Fig. 5.44 Debounce signal

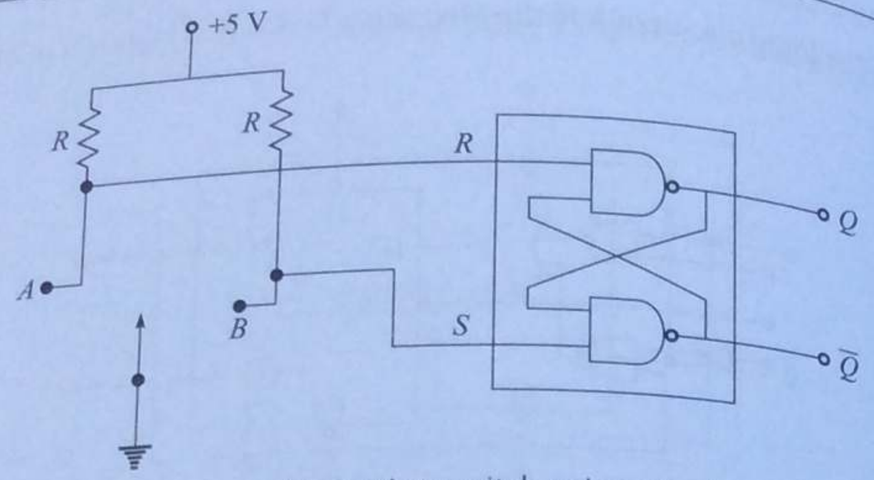


Fig. 5.45 Bounce elimination switch using simple S-R latch

Initially, the switch is at position A, the input R is 0 and S is 1, the output Q is 1 and $\overline{Q}$ is 0. When the switch is thrown from position A to position B at time t_1, R goes to 1 and S goes to 0; $Q = 0$ and $\overline{Q} = 1$. Because of spring action, the switch may be open for a small duration, both the inputs S and R become 1 and the output is unchanged, as shown in Fig. 5.46.

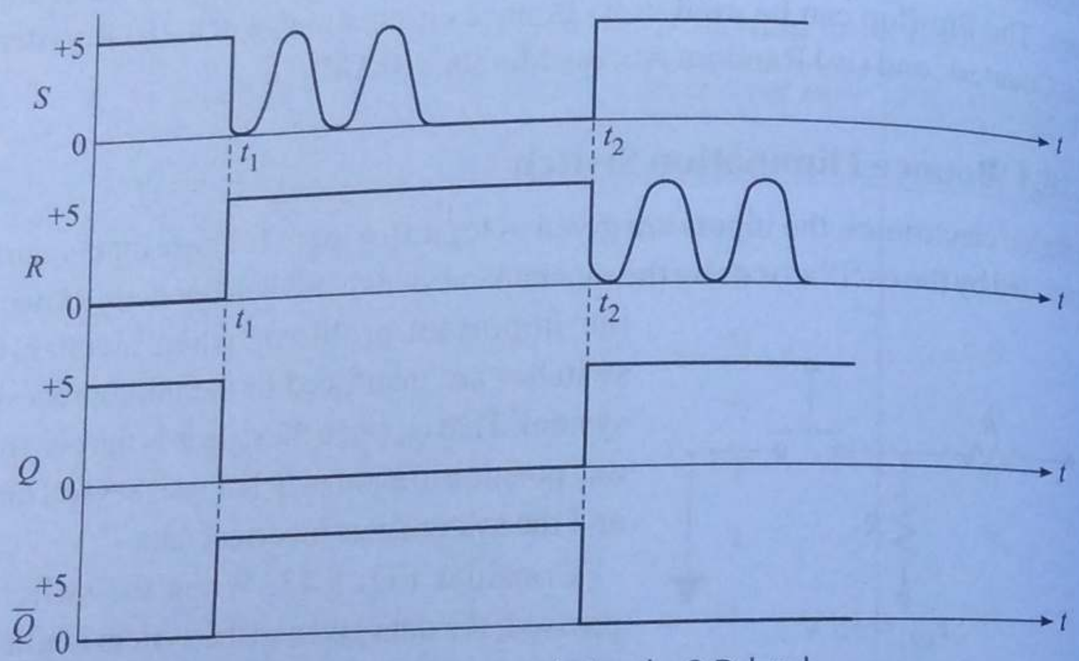


Fig. 5.46 output of simple S-R latch

Let us consider at time t_2, the switch is thrown to position A. S goes to 1 and R goes to 0. Q is 1 and $\overline{Q}$ is 0. Because of spring effect, the switch may be open. Both the inputs S and R become 1 for a small duration and the output remains unchanged.

5.10.2 Registers

The flip-flop is a 1-bit memory cell, which stores the data in logic 1 or logic 0 form. The N-bit flip-flop can store N bits of data. On the other hand, registers can store more than 1-bit data. It is composed of a group of flip-flops. A 4-bit shift register using D flip-flop is shown in Fig. 5.47.

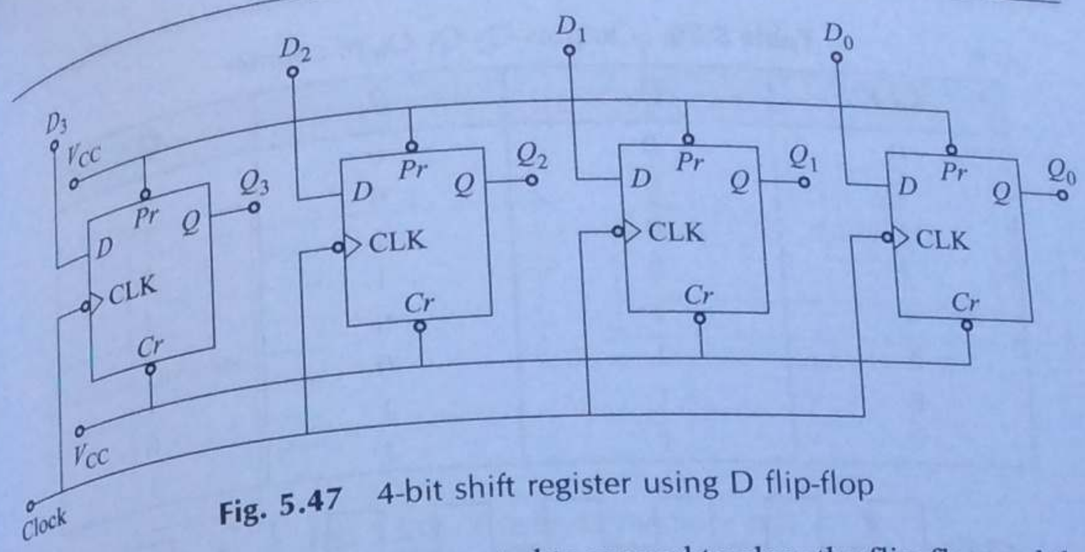


Fig. 5.47 4-bit shift register using D flip-flop

Initially, the clear input is connected to ground to clear the flip-flop and then, both the clear and preset inputs are connected to V_{CC}. The data to be stored is given to the D input. The data will be stored with negative edge of the clock, which can be read from outputs $Q_3Q_2Q_1Q_0$.

5.10.3 Counters

The digital counter is a sequential circuit, which counts electric pulses. It is used to count the events. The transducer converts the events into electric pulses, which can be counted by the digital counter.

The counter is composed of flip-flops. The logic diagram of a 3-bit digital counter is shown in Fig. 5.48.

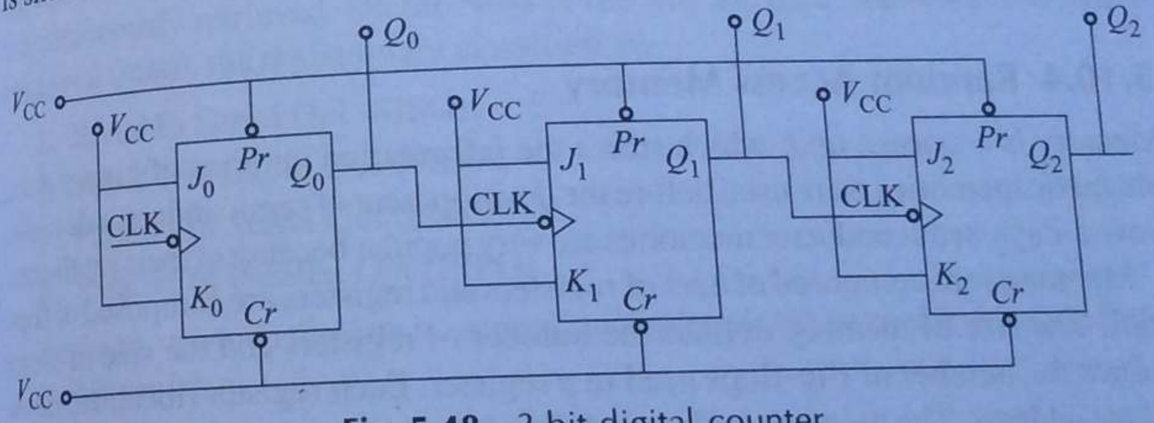


Fig. 5.48 3-bit digital counter

In a counter circuit, the flip-flops work as toggle switches. The outputs of flip-flops change with each clock, and the output of a previous flip-flop becomes the clock for the next flip-flop. The n-bit counter can count 2^n clock pulses. For example, a 4-bit counter counts $2^4 = 16$ pulses from 0 to 15. The design of counters will be discussed in Section 5.18.

Initially, the clear input is connected to ground to clear the flip-flop and then, both the clear and preset inputs are connected to V_{CC}. The outputs $Q_2Q_1Q_0$ of the counter will change with the clock, as shown in Fig. 5.49.

Table 5.29 Outputs Q_2 Q_1 Q_0 of counter

CLK	Q_2	Q_1	Q_0
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

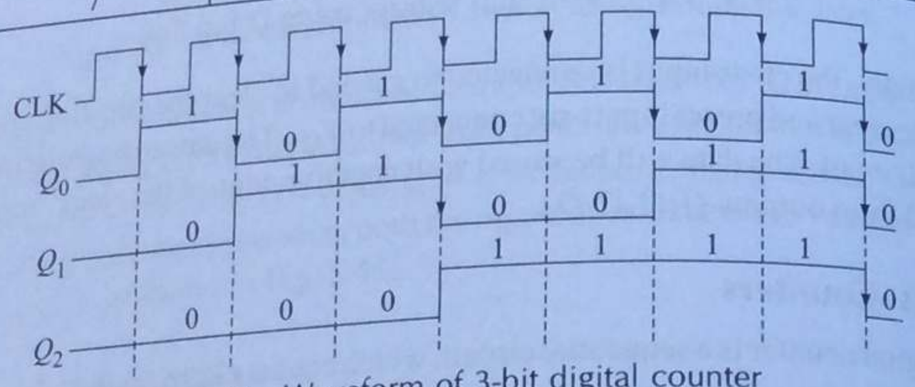


Fig. 5.49 Waveform of 3-bit digital counter

The output Q_0 changes with the negative edge of the input clock. Since Q_0 and Q_1 are the clocks for flip-flop$_1$ and flip-flop$_2$ respectively, the output of flip-flop$_1$ will change with the negative edge of Q_0 and that of flip-flop$_2$ will change with the negative edge of Q_1.

5.10.4 Random Access Memory

Memory is a storage unit, which stores the information in terms of binary data. Magnetic memories were used before the development of semiconductor devices. Now-a-days, semiconductor memories are very popular because of their small size.

Memories are composed of a set of registers and registers are composed of flip-flops. The size of memory defines the number of registers and the size of word defines the number of flip-flops used in a register. Each register (location) has a unique address. The memory is always defined by the equation $M \times N$, where M is the number of registers (location) and N is the size of word.

The digital data is stored into and retrieved from the memory. These write and read operations are done in a random fashion and hence, the memory is said to be random access memory.

The memory has $M \times N$ basic memory elements known as cells (flip-flop). The basic memory cell is shown in Fig. 5.50.

The decoder decodes the address and selects one of the locations. For write operation, the data to be stored is given to the input D_i. R/$\overline{W}$ control signal goes low. The output of gate G_1 changes from 0 to 1, since the flip-flop is positive edge triggered; and the input data is stored.

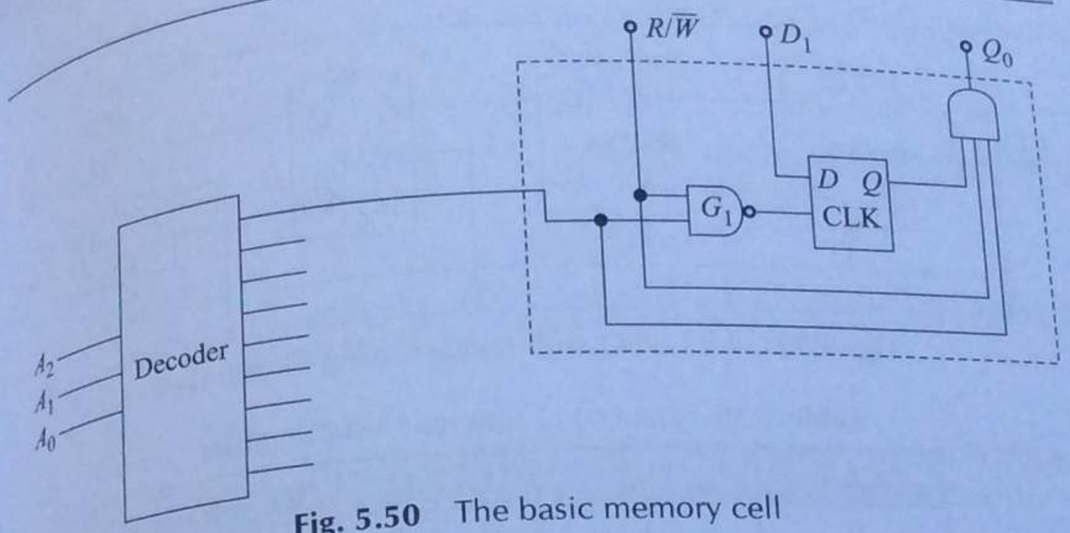


Fig. 5.50 The basic memory cell

For read operation, the R/$\overline{W}$ signal goes high. The output of flip-flop is available at Q_0.

5.11 REGISTERS AND SHIFT REGISTERS

We have seen that a flip-flop is 1-bit memory cell, which stores 1-bit of digital data. A group of flip-flops stores more than 1-bit of digital data. N flip-flops can store N bits of digital data. An array of flip-flops is known as register, which finds its application in a variety of digital systems.

The data can be entered and retrieved in serial or parallel form the serial form means bit-by-bit (one bit at time) and parallel from means all the bits are simultaneously retrieved. On the basis of the way the data is entered (write) and retrieved (read), the registers are classified as,

1. Serial In Serial Out (SISO)
2. Serial In Parallel Out (SIPO)
3. Parallel In Serial Out (PISO)
4. Parallel In Parallel Out (PIPO)

Registers in which the data is entered and/or retrieved in serial form are referred to as *shift register*.

5.11.1 SISO Shift Register

In a SISO shift register, data is entered and retrieved in serial fashion with clock. The logic diagram of a 4-bit SISO shift register using J-K flip-flop is shown in Fig. 5.51.

In Fig. 5.51, X_i is input and Y_o is output of the shift register. The process of entering the digital data starts with the least significant bits. The data input is entered with the falling edge of the clock pulse. The number of clock pulses required to enter the data is equal to the length of digital data or the size of shift register. The data is read bit-by-bit at the output Y_o with clock pulse.

Let us consider the data 0111 is applied to the input. Table 5.30 demonstrates how the data is entered into the shift register. The waveforms of shift register for serial input are shown in Fig. 5.52.

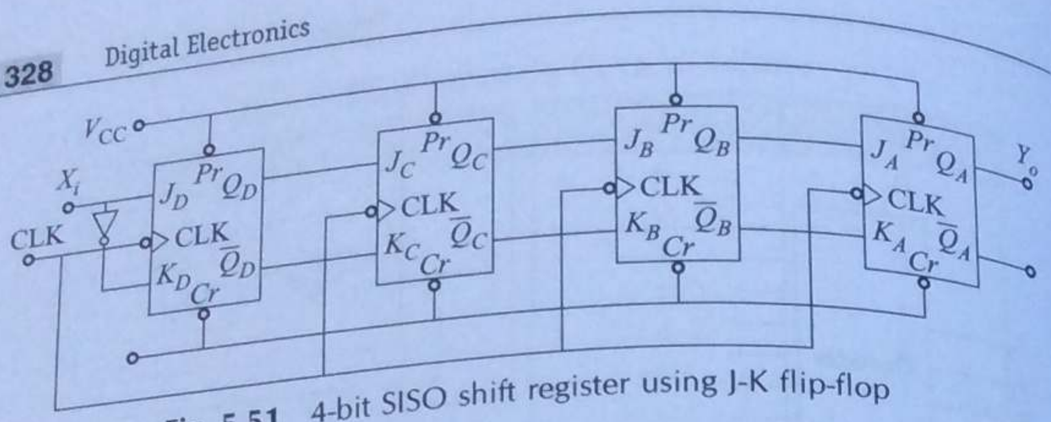


Fig. 5.51 4-bit SISO shift register using J-K flip-flop

Table 5.30 Data 0111 entered in shift register

CLK	X_i	Q_D	Q_C	Q_B	Q_A
0	1	0	0	0	0
1	1	1	0	0	0
2	1	1	1	0	0
3	0	1	1	1	0
4		0	1	1	1

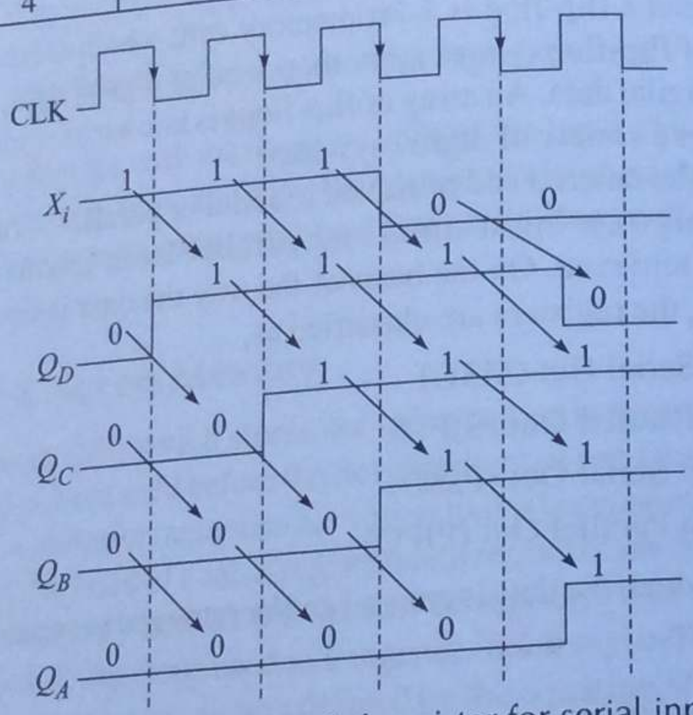


Fig. 5.52 Waveforms of shift register for serial input

Operation

The inputs of flip-flops *D, C, B,* and *A* are X_i, Q_D, Q_C, and Q_B, respectively. All flip-flops operate as D flip-flop and the input applied is 0111.

Initially, the shift register is cleared.

$Q_DQ_CQ_BQ_A = 0000$ and input $X_i = 1$

1. At the negative edge of the first clock pulse, the input data is entered into the flip-flop and at the end of first clock pulse,

   $Q_DQ_CQ_BQ_A = 1000$ and input $X_i = 1$

2. At the negative edge of the second clock pulse, the inputs are entered and at the end of second clock pulse,

   $Q_DQ_CQ_BQ_A = 1100$ and input $X_i = 1$

3. At the negative edge of the third clock pulse, the inputs are entered and at the end of third clock pulse,

   $Q_DQ_CQ_BQ_A = 1110$ and input $X_i = 0$

4. At the negative edge of the fourth clock pulse, the inputs are entered and at the end of fourth clock pulse,

   $Q_DQ_CQ_BQ_A = 0111$

In an *n*-bit SISO shift register, *n* clock pulses are required to enter the data and *n* clock pulses are required to read the data, as explained in Section 5.11.3.

Disadvantages

1. *n* clock pulses are required to enter the *n*-bit data.
2. *n* clock pulses are required to read the *n*-bit data.
3. Once the data is read, it will be lost.

5.11.2 SIPO Shift Register

In a SIPO shift register, data is entered into the register in serial fashion, as in SISO shift register and read from the shift register in parallel fashion. In a serial output shift register, clock pulses are required to read the data and once the data is read, it will be lost, but in parallel out shift register, clock pulse(s) is not required to read the data and the data is retained after the read operation. The logic diagram of a 4-bit SIPO shift register using D flip-flop is shown in Fig. 5.53.

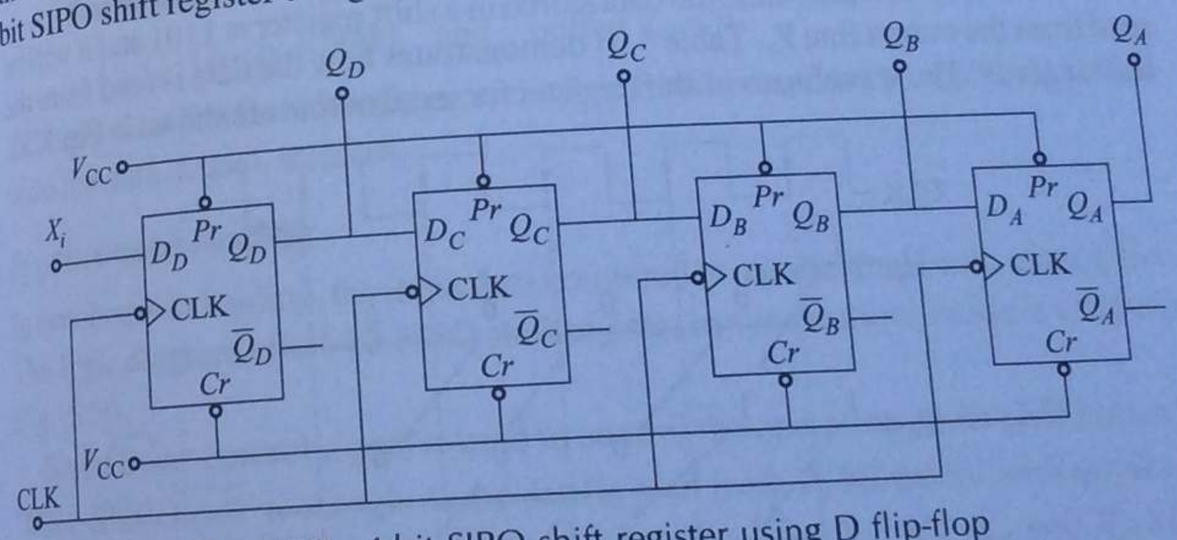


Fig. 5.53 4-bit SIPO shift register using D flip-flop

In the figure, X_i is the serial input and Q_D, Q_C, Q_B, and Q_A are the parallel outputs of the shift register.

5.11.3 PISO Shift Register

In a PISO shift register, data is entered in parallel fashion and is read in serial fashion. There are two types of parallel loading:

1. Asynchronous loading, and
2. Synchronous loading

Asynchronous loading

In asynchronous loading, the preset inputs are used to load the data simultaneously. The logic diagram of a 4-bit PISO shift register with asynchronous loading is shown in Fig. 5.54.

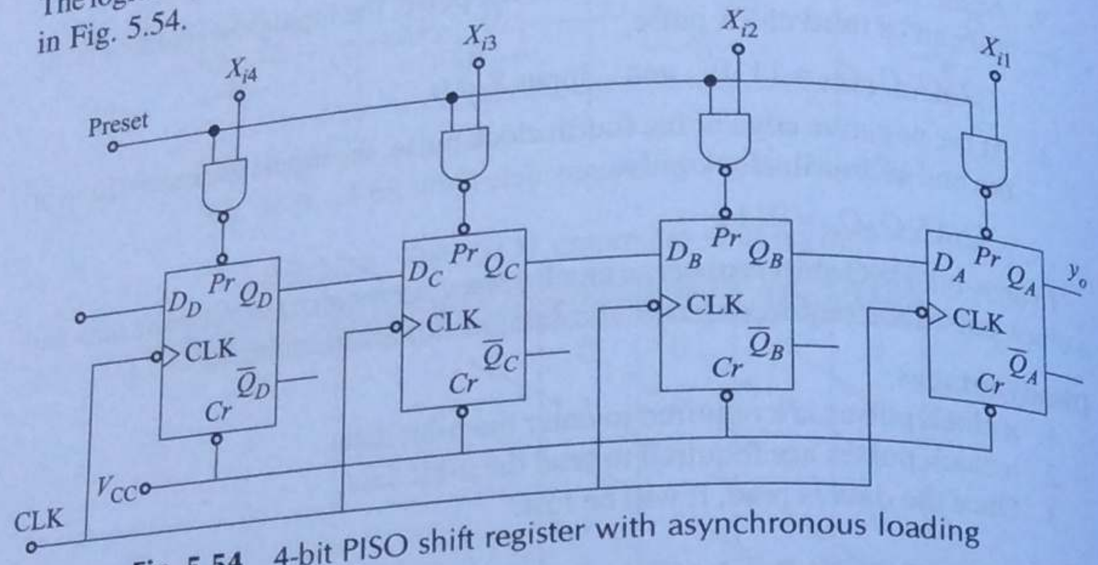


Fig. 5.54 4-bit PISO shift register with asynchronous loading

Initially, connecting the clear input to ground clears the flip-flops. The inputs are given to the parallel inputs (X_{i1}, X_{i2}, X_{i3}, and X_{i4}) and the preset is connected to logic 1, the output of NAND gate is complement of input, the preset input is active low, flip-flop is set for 0 and unchanged for 1. The data is written into the registers. The inputs are written into the registers without the clock pulse. Such type of parallel loading is known as *asynchronous loading*. The data is read from output lines Y_o, bit-by-bit by applying the clock pulse. Once the data is read, it will be lost.

For example, let us assume the data stored in a shift register is 1101 and it will be read from the output line Y_o. Table 5.31 demonstrates how the data is read from the shift register. The waveforms of shift register for serial output are shown in Fig. 5.55.

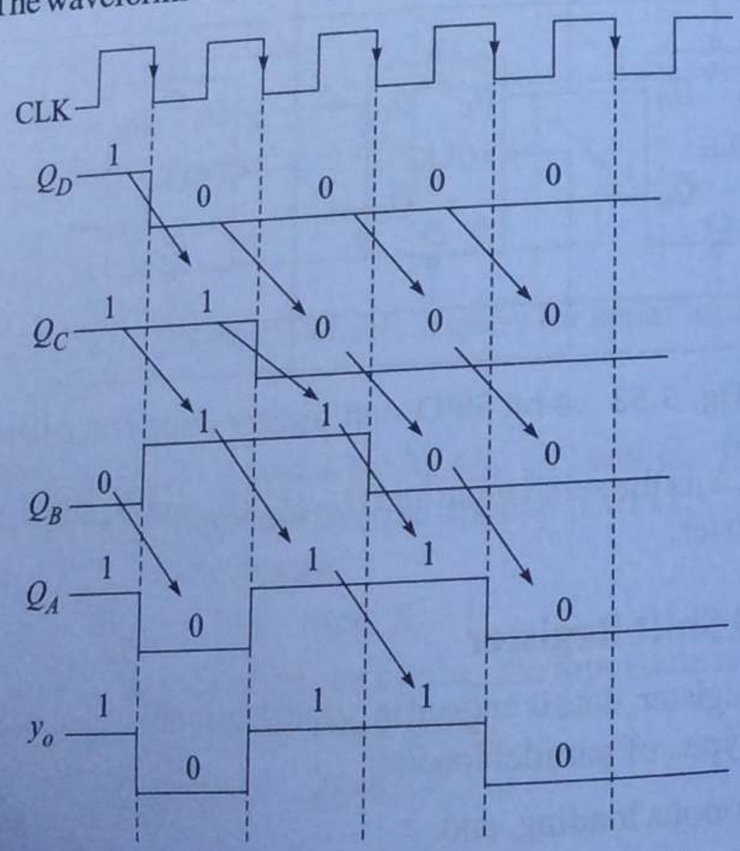


Fig. 5.55 Waveforms of shift register for serial out

Table 5.31 Data 0101 read from the shift register

CLK	Q_D	Q_C	Q_B	Q_A	Y_O
0	1	1	0	1	1
1	0	1	1	0	0
2	0	0	1	1	1
3	0	0	0	1	1
4	0	0	0	0	0

Operation

The inputs of flip-flops *D*, *C*, *B*, and *A* are X_i, Q_D, Q_C, and Q_B, respectively. All flip-flops operate as D flip-flop and the input X_i is logic 0.

Initially, the shift register stores 1101.

$Q_D Q_C Q_B Q_A = 1101$ and output $Y_O = 1$

1. At the negative edge of the first clock pulse,
   $Q_D Q_C Q_B Q_A = 0110$ and output $Y_O = 0$
2. At the negative edge of the second clock pulse,
   $Q_D Q_C Q_B Q_A = 0011$ and output $Y_O = 1$
3. At the negative edge of the third clock pulse,
   $Q_D Q_C Q_B Q_A = 0001$ and output $Y_O = 1$
4. At the negative edge of the fourth clock pulse,
   $Q_D Q_C Q_B Q_A = 0000$ and output $Y_O = 0$

For an *n*-bit PISO shift register, *n* clock pulses are required to read the data and once the data is read, it is lost.

Synchronous loading

In synchronous loading, the input data is entered in parallel form with clock pulse. The logic diagram of a 4-bit PISO shift register in synchronous mode is shown in Fig. 5.56.

Shift/$\overline{\text{Load}}$ control signal is used to control the operation of the shift register. When Shift/$\overline{\text{Load}}$ is at logic 1, the data is read from Y_o bit-by-bit with the clock pulse. When Shift/$\overline{\text{Load}}$ is at logic 0, the data inputs X_{i1}, X_{i2}, X_{i3} and X_{i4} load simultaneously with the clock pulse. Such type of loading is referred to as *synchronous loading*. Consider the data input is 0101.

When Shift/$\overline{\text{Load}}$ signal is 0, the outputs of gates G_1, G_3, G_5 and G_7 are 0 and the outputs of gates G_2, G_4, G_6 and G_8 are same as the inputs 0101. The outputs of OR gates are 0101, and these are the inputs to D flip-flops. It is loaded into the register with the falling edge of the clock pulse.

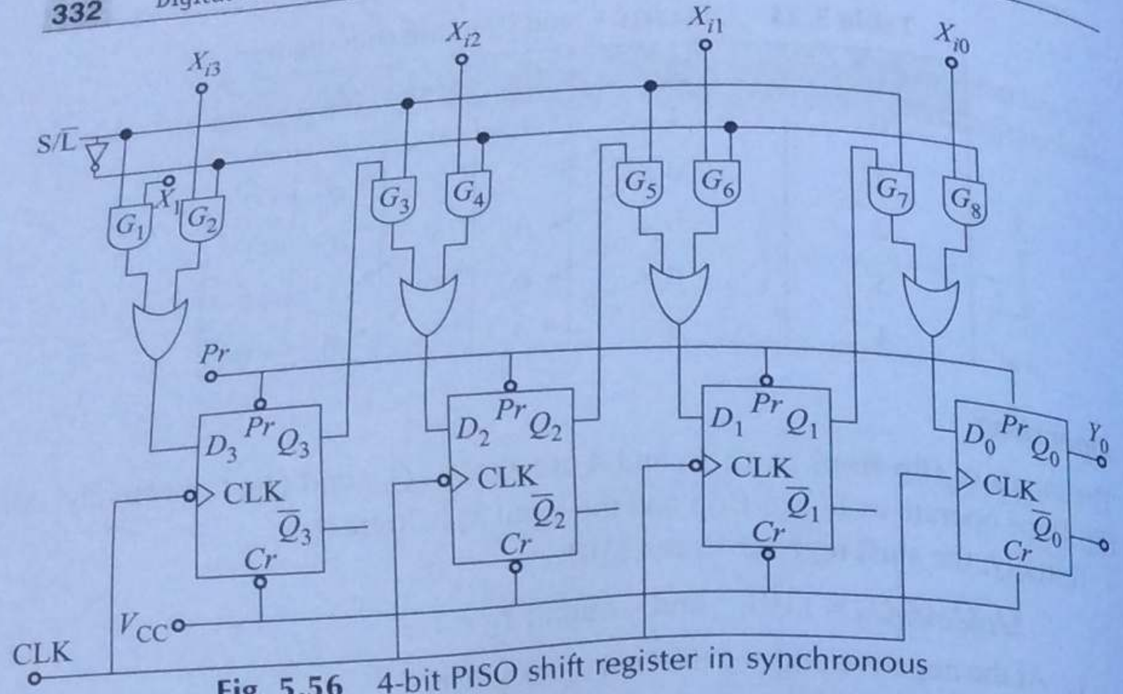


Fig. 5.56 4-bit PISO shift register in synchronous

When Shift/ $\overline{\text{Load}}$ is 1, the outputs of gates G_2, G_4, G_6, and G_8 are 0 and the outputs of gates G_1, G_3, G_5, and G_7 are X_i, Q_3, Q_2, and Q_1. The outputs of OR gates are X_i, Q_3, Q_2, and Q_1, and these are the inputs to D flip-flops. It is loaded into the register with the falling edge the of the clock pulse. It shows that the data is shifted in the register and we get the output at Y_o.

5.11.4 PIPO Shift Register

In a PIPO register, the data is entered as well as read in parallel fashion. There are two types of parallel loading: (i) asynchronous loading, and (ii) synchronous loading. The logic diagram of a 4-bit asynchronous loading PIPO is shown in Fig. 5.57 and the synchronous loading PIPO is shown in Fig. 5.58.

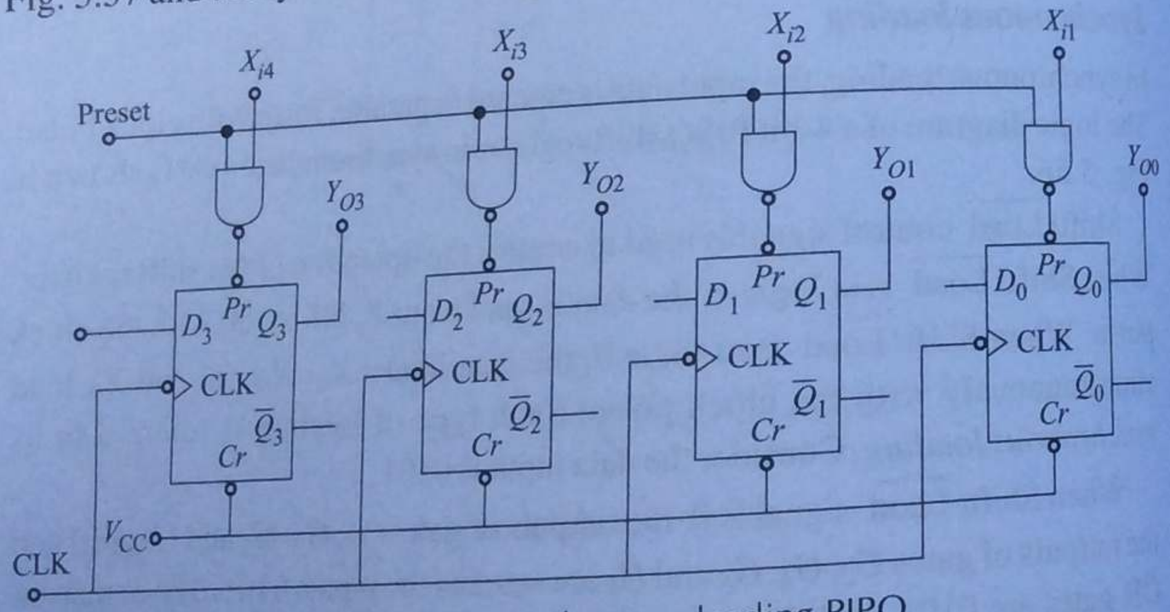


Fig. 5.57 4-bit asynchronous loading PIPO

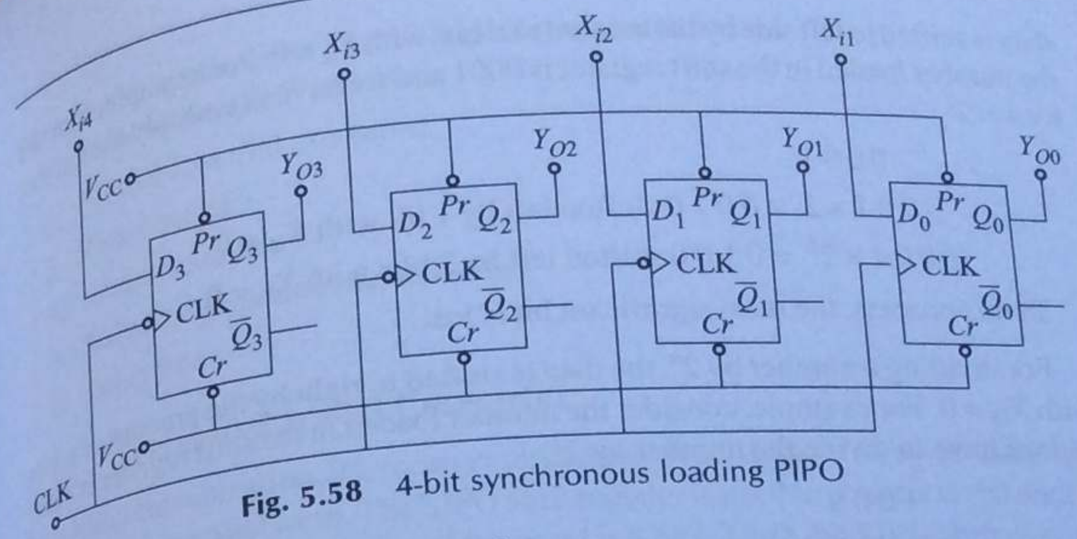


Fig. 5.58 4-bit synchronous loading PIPO

5.11.5 Bidirectional Shift Register

In a bidirectional shift register, the data is shifted in either directions, left or right. The direction is controlled by the control input R/ $\overline{\text{L}}$. The 4-bit bidirectional shift register is shown in Fig. 5.59.

When R/ $\overline{\text{L}}$ control signal is high, the gates G_1, G_3, G_5, and G_7 are enabled. The output of flip-flop A is the input for flip-flop B, the output of flip-flop B is the input for flip-flop C, the output of flip-flop C is the input for flip-flop D, and X_{iR} is the input of flip-flop A. Data is shifted right with the clock pulse.

When R/ $\overline{\text{L}}$ control signal is low, the gates G_2, G_4, G_6, and G_8 are enabled. The output of flip-flop D is the input of flip-flop C, the output of flip-flop C is the input of flip-flop B, the output of flip-flop B is the input of flip-flop A, and X_{iL} is the input of flip-flop D. Data is shifted left with the clock pulse.

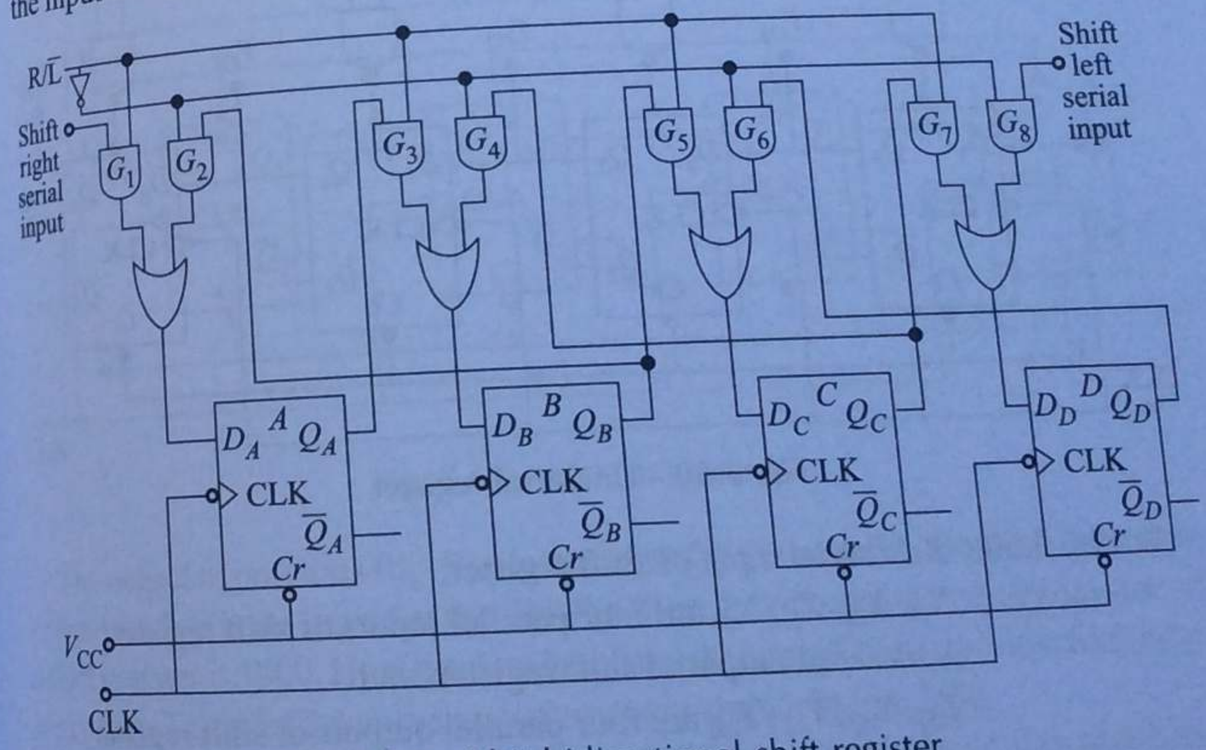


Fig. 5.59 4-bit bidirectional shift register

Application of Bidirectional Shift Register

The bidirectional shift register is used to multiply or divide the number by 2^n, provided that 1 is not shifted out of register. For multiplying a number by 2^n, the

data is shifted to left side by the amount of n bits with $X_{iL}=0$. For example, consider the number loaded in the shift register is 0001 and we have to multiply the number by $4=2^2$.

$$0001$$
$$0001\times 2 = 0010 \text{ shifted left by 1 bit with } X_{iL}=0$$
$$0001\times 2^2 = 0100 \text{ shifted left by 2 bits with } X_{iL}=0$$

In this process, the most significant bit is lost.

For dividing a number by 2^n, the data is shifted to right by the amount of n bits with $X_{iR}=0$. For example, consider the number loaded in the shift register is 1000 and we have to divide the number by 2^2.

$$1000$$
$$1000/2 = 0100 \text{ shifted right by 1 bit with } X_{iR}=0$$
$$1000/2^2 = 0010 \text{ shifted right by 2 bits with } X_{iR}=0$$

In this process, the least significant bit is lost.

5.11.6 Universal Register

The universal shift register operates in all possible four modes (SISO, SIPO, PISO, PIPO) and also as bidirectional shift registers. The logic diagram of a 4-bit shift register is shown in Fig. 5.60. It is operated in all the modes, but not as bidirectional and shift register.

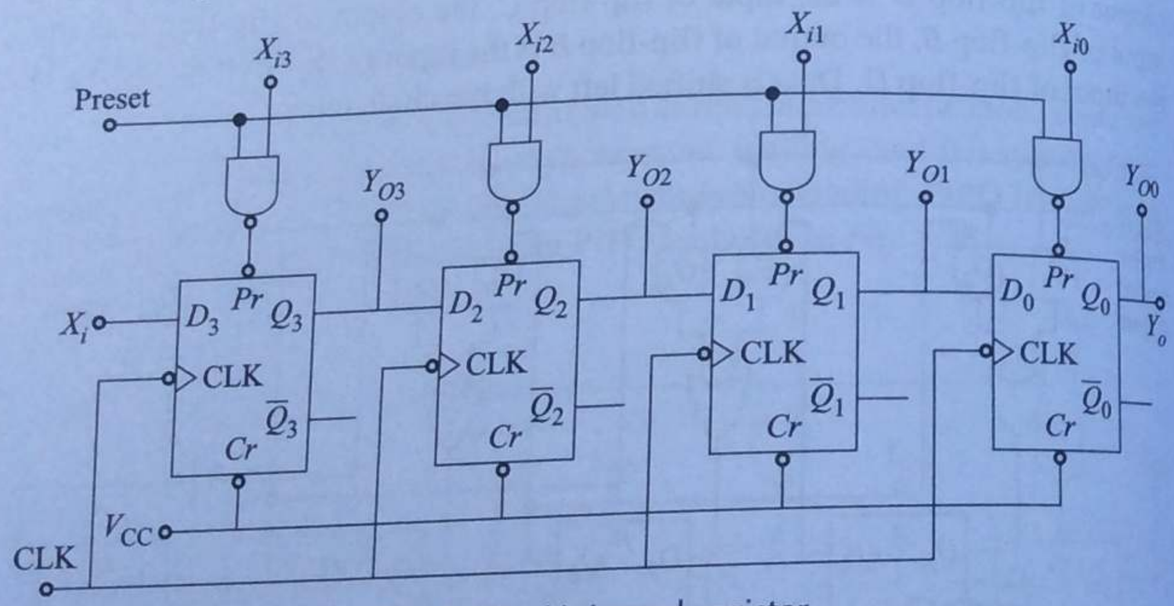


Fig. 5.60 Universal register

In Fig. 5.60, X_i is serial input of shift register,
$X_{i3}, X_{i2}, X_{i1}, X_{i0}$ are four parallel inputs of shift register,
Y_o is serial output of shift register, and
$Y_{o3}, Y_{o2}, Y_{o1}, Y_{o0}$ are four parallel outputs of shift register.

5.12 APPLICATIONS OF SHIFT REGISTER

The register is a storage unit. It stores the information in terms of 0 and 1. It is also used for bit multiplication.

The common applications of shift registers are listed below.

1. Serial-to-parallel converter
2. Parallel-to-serial converter
3. Ring counter
4. Johnson counter
5. Sequence generator and detector
6. Delay line

5.12.1 Serial-to-Parallel Converter

In serial communication, the receiver circuit receives the data bit-by-bit, and then converts it in parallel form. The SIPO shift register is used to convert serial data in parallel form. Consider the size of the word is 8 bits. The 8-bit SIPO shift register receives the data bit-by-bit with clock pulse. At the end of 8th clock pulse, the 8-bit data is available over the eight output lines of the shift register.

5.12.2 Parallel-to-Serial Converter

In serial communication, the transmitter circuit gets the eight bits of data, and then transmits it bit-by-bit. The PISO shift register is used to convert parallel data in serial form.

5.12.3 Ring Counter

Figure 5.61 shows the logic diagram of a 4-bit ring counter. The SISO shift register acts as a ring counter, if the output Q_0 is connected to the serial input X_i.

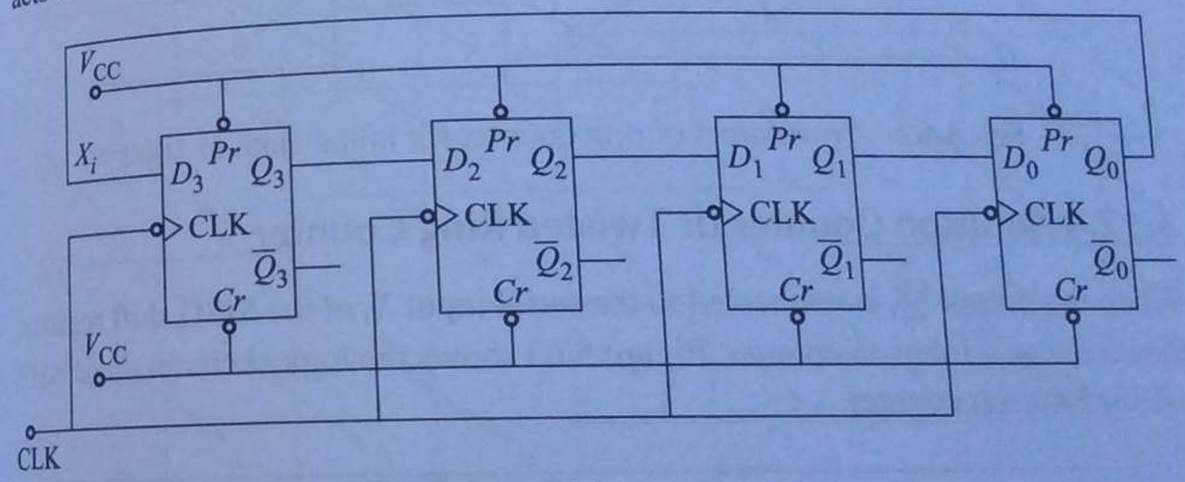


Fig. 5.61 4-bit ring counter

The output of previous flip-flop is the input for the next flip-flop and the output of last flip-flop is the input for the first flip-flop. Let us consider the initial state of a shift register is 1000. How the data is shifted in the shift register with clock pulse is given in Table 5.32 and the waveforms are shown in Fig. 5.62.

The waveforms show that the initial data is circulated in the shift register and the initial state is obtained after the 4th clock pulse. This circuit is used to count the clock pulses and it is known as a ring counter. The output is available at Q_3, Q_2, Q_1, and Q_0. There is no need of a decoding circuit.

Table 5.32 Data shifted in ring counter

CLK	X_i	Q_3	Q_2	Q_1	Q_0
0	0	1	0	0	0
1	0	0	1	0	0
2	0	0	0	1	0
3	1	0	0	0	1
4	0	1	0	0	0

The N-bit ring counter counts N clock pulses. The frequency of the pulses obtained at Q_0 is the frequency of clock pulse divided by N. Hence it is also known as *divided-by-N counter*. The outputs (Q_3, Q_2, Q_1, Q_0) are sequential non-overlapping pulses, which are useful to rotate the stepper motor.

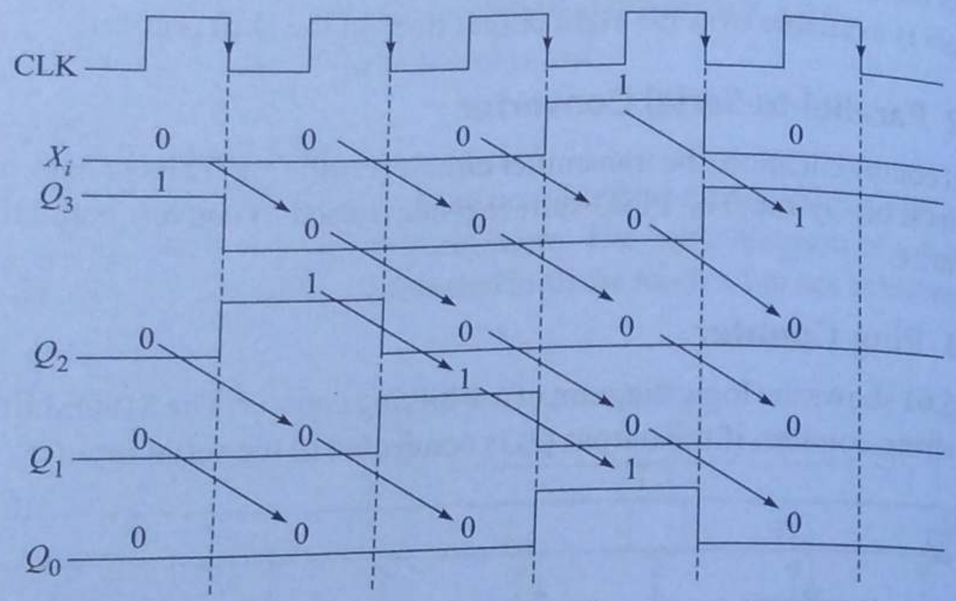


Fig. 5.62 Waveforms of ring counter for initial state of 1000

5.12.4 Johnson Counter or Twisted Ring Counter

When the output $\overline{Q}_0$ is connected to the serial input X_i of the SISO shift register, then it act as a Johnson counter. Figure 5.63 shows the logical circuit diagram of a 4-bit Johnson counter.

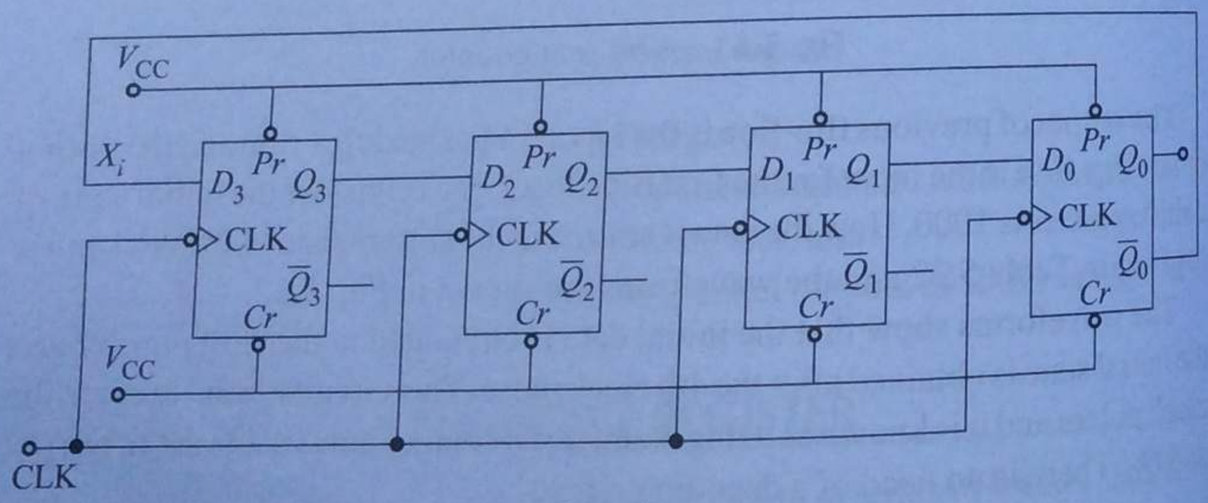


Fig. 5.63 4-bit Johnson counter

Initially, all the flip-flops are cleared by connecting the clear input to ground. Then both the clear and preset inputs are connected to V_{CC} and the clock is applied to the register. How the data is shifted in Johnson counter with clock pulse is given in Table 5.32 and the waveforms are shown in Fig. 5.65. $Q_0=0$, the input $X_i=\overline{Q}_0=1$, the input is loaded into the flip-flop with the falling edge of the clock.

Table 5.33 Data shifted in Johnson counter

CLK	X_i	Q_3	Q_2	Q_1	Q_0
0	1	0	0	0	0
1	1	1	0	0	0
2	1	1	1	0	0
3	1	1	1	1	1
4	0	1	1	1	1
5	0	0	1	1	1
6	0	0	0	1	1
7	0	0	0	0	1
8	1	0	0	0	0

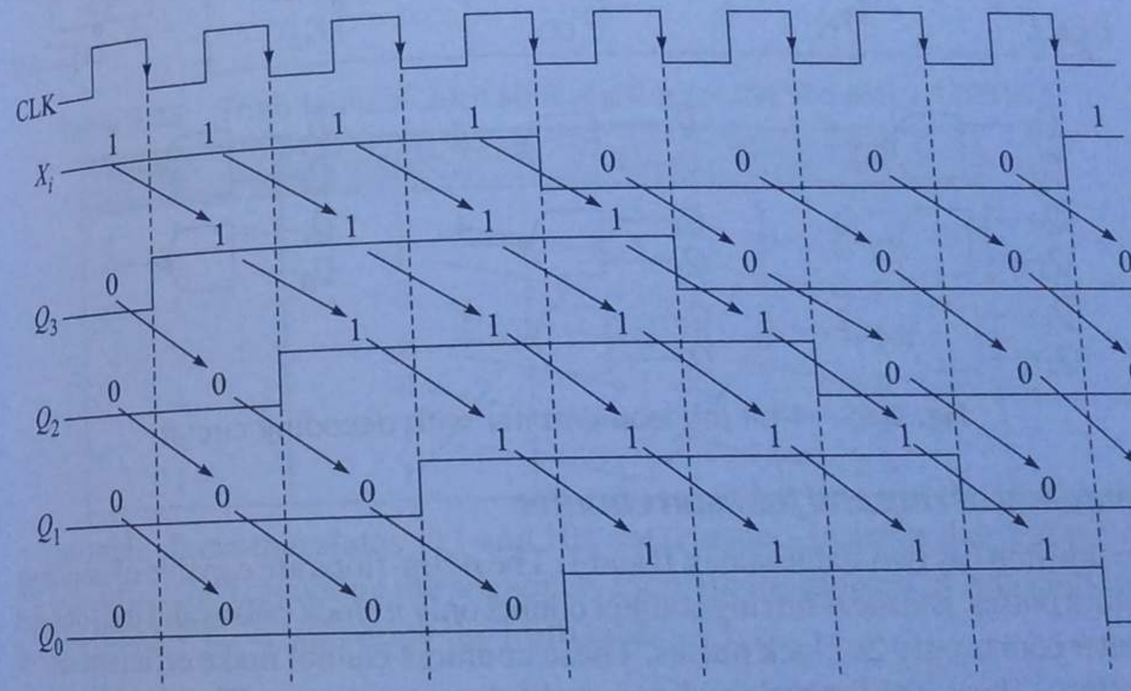


Fig. 5.64 Waveforms of Johnson counter for initial state of 0000

The frequency of the pulses obtained at Q_0 is the frequency of clock pulse divided by $2N$. Hence, it is known as *divided-by-2N counter* or *twisted counter*. The 4-bit Johnson counter counts the eight clock pulses, i.e. the n-bit Johnson counter counts the $2n$ clock pulses.

Johnson counter requires the decoder circuit to decode the count. Table 5.34 shows the counter sequence and Boolean equations for the decoder circuit.

Table 5.34 Counter sequence and Boolean equations for decoder circuit

CLK	Q_3	Q_2	Q_1	Q_0	Boolean Equation
0	0	0	0	0	$\overline{Q}_3\overline{Q}_0$
1	1	0	0	0	$Q_3\overline{Q}_2$
2	1	1	0	0	$Q_2\overline{Q}_1$
3	1	1	1	0	$Q_1\overline{Q}_0$
4	1	1	1	1	Q_3Q_0
5	0	1	1	1	$\overline{Q}_3Q_2$
6	0	0	1	1	$\overline{Q}_2Q_1$
7	0	0	0	1	$\overline{Q}_1Q_0$

The logic diagram of a 4-bit Johnson or twisted counter with decoding circuit is shown in Fig. 5.65.

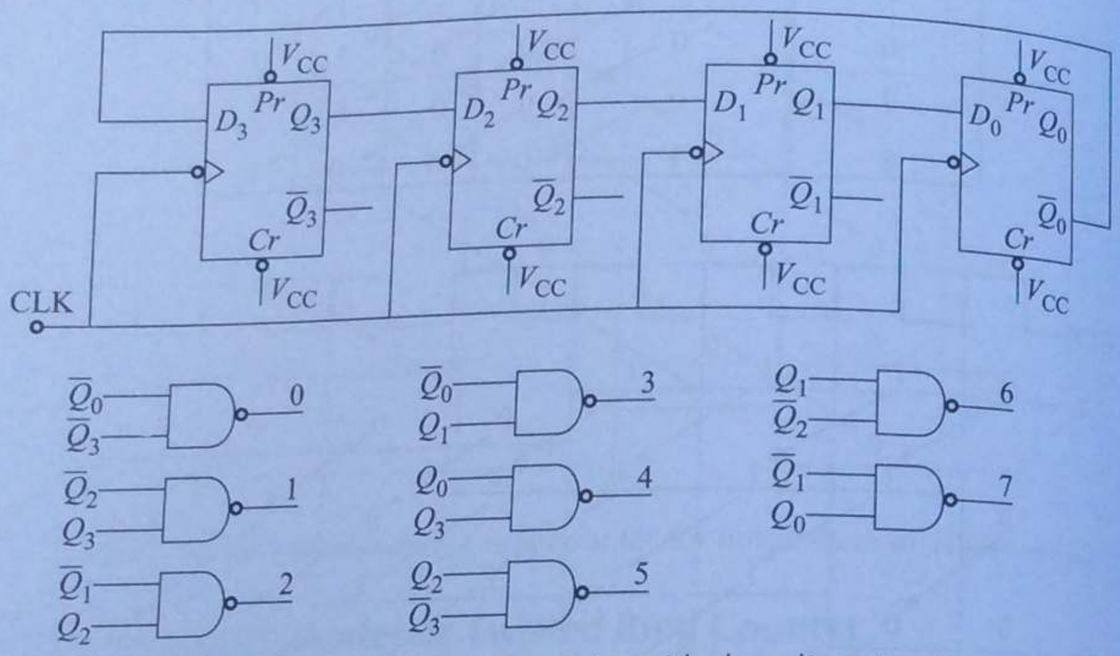


Fig. 5.65 4-bit Johnson counter with decoding circuit

Limitation of ring and Johnson counter

The flip-flop has two stable states, 0 and 1. The n flip-flops are capable of counting 2^n clock pulses. But the n-bit ring counter counts only n clock pulses and the Johnson counter counts only $2n$ clock pulses. These counters cannot make efficient use of flip-flops. This is the limitation of ring and Johnson counters. The ripple counter overcomes this limitation.

5.12.5 Sequence Generator

The logic circuit that generates a defined sequence of 0 and 1 in synchronism with the clock is known as *sequence generator*. The basic block diagram of a sequence generator is shown in Fig. 5.66.

The output of combinational circuit is a function of the output of shift register. The output of combinational circuit is applied as input to the shift register. The procedure to design the sequence generator is as follows:

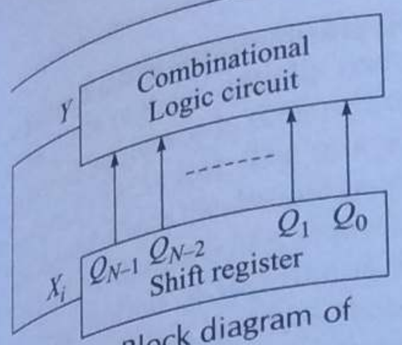


Fig. 5.66 Block diagram of sequence generator

1. Find the minimum number of flip-flops required to generate the sequence.
   For the length of sequence S, the minimum number of flip-flops required is N. The relation is given by
   $$S \le 2^N - 1$$
2. Prepare the truth table of N-bit shift register. The defined sequence is listed under Q_{N-1} and Y is the past input of N–1 flip-flop.
3. Check whether one or more states are repeated in the table. If yes, it shows that N flip-flops are not sufficient to generate the sequence. Repeat step 2 for N+1 flip-flop. When there is no repetition of states, then draw the K-map for the output Y and simplify.
4. Draw the combinational logic circuit on the basis of Boolean equation Y.

Consider the sequence to be generated as 1101011.

Step 1: Length of the sequence is $S = 7$ and the minimum number of flip-flops required is N.

$$S \le 2^N - 1$$
$$7 \le 2^N - 1$$
$$N = 3$$

Step 2:

Table 5.35 Truth table of 3-bit shift register for the sequence 1101011

Y	Q_2	Q_1	Q_0
1	1	1	1
0	1	1	1
1	0	1	1
0	1	0	1
1	0	1	0
1	1	0	1
1	1	1	0

The table shows that states 111 and 101 are repeated. It shows that N flip-flops are not sufficient to generate the sequence. Increase the number of flip-flops by 1, i.e. $N + 1 = 3 + 1 = 4$, and draw the truth table.

Table 5.36 Truth table of 4-bit shift register for the sequence 1101011

Y	Q_3	Q_2	Q_1	Q_0
1	1	1	1	0
0	1	1	1	1
1	0	1	1	1
0	1	0	1	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

The table shows that all the states are different. It is possible to design the sequence generator with four flip-flops.

Step 3: K-map for output Y.

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	×	×	×	×
01	×	1	1	×
11	×	1	0	0
10	×	×	1	1

$$Y = \overline{Q}_3 + \overline{Q}_1 + \overline{Q}_0$$

Step 4: The logic diagram to generate the defined sequence is shown in Fig. 5.67.

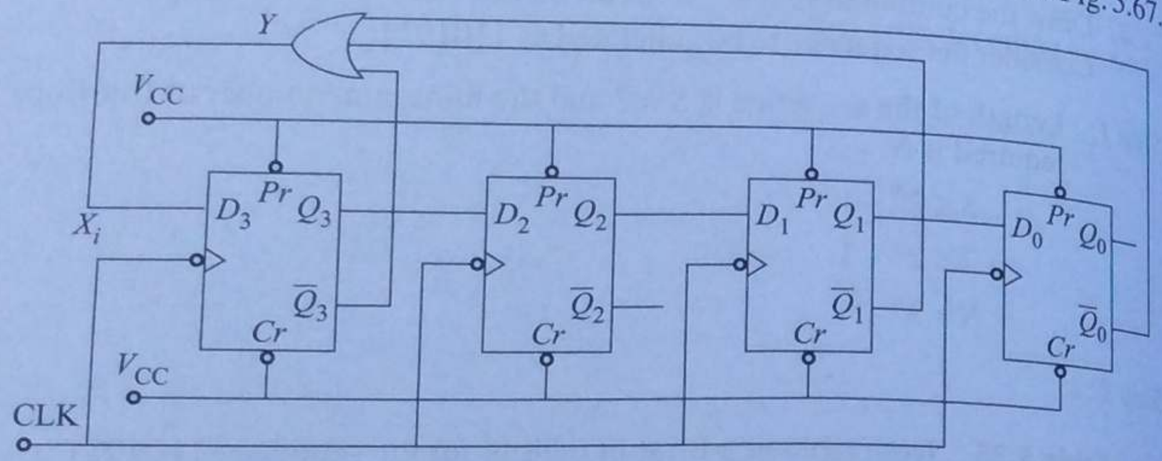


Fig. 5.67 Logic diagram to generate the sequence 1101011

5.12.6 Sequence Detector

Sequence detectors are used in receiver circuits to detect the correct sequence, which is sent by the transmitter. The shift register is used to detect the sequence. The size

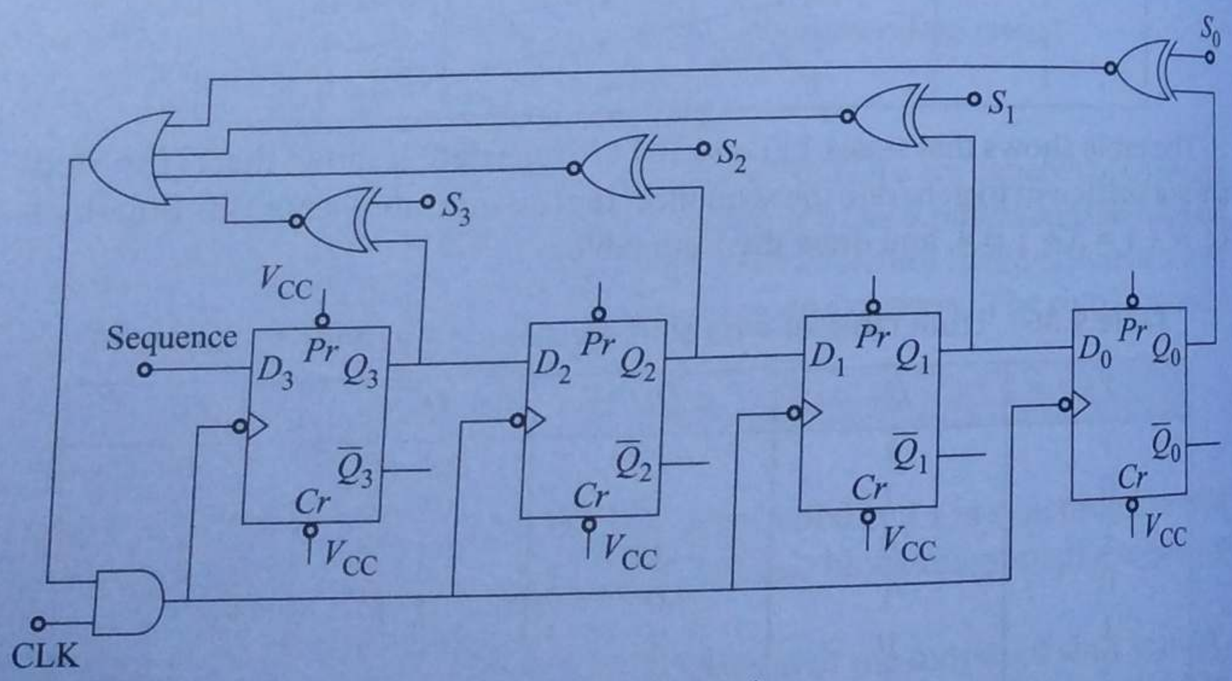


Fig. 5.68 Sequence detector

of the shift register used to detect the sequence is equal to the length of the sequence. The shift register initially receives the sequence and then, starts to check if the desired sequence is obtained or not. If not, then it receives the next bit and the leftmost bit is lost. The procedure is continued till the desired sequence is not detected. Once the desired sequence is detected, it stops applying the clock pulse. The logic diagram of a sequence detector is shown in Fig 5.68.

The output of flip-flop is compared with corresponding bits of the desired sequence. If it is matched, then the outputs of all Ex-NOR gates are 0, the outputs of OR and AND gates are 0, and the clock given to the flip-flop is stopped.

5.13 COMMONLY USED ICs AS SHIFT REGISTER

74L91	8-bit SISO shift register
74164	8-bit SIPO shift register
74165	8-bit PISO shift register
74194	4-bit PIPO shift register
74198	8-bit PISO shift register
54L91	8-bit SIPO shift register
54164	8-bit SIPO shift register
54165	8-bit PISO shift register
54194	4-bit PIPO shift register
54198	8-bit PIPO shift register

Example 5.9 Design a pulse train generator using shift register for the following train.

---------- 1000110, 1000110 ----------

Solution

Step 1: The length of sequence is $S = 7$ and the minimum number of flip-flops required is N.

$$S \le 2^N - 1$$

$$7 \le 2^N - 1$$

$$N = 3$$

Step 2: Table 5.37 shows that all the states are different. It is possible to design the sequence generator with three flip-flops.

Table 5.37 Truth table of 3-bit shift register for the sequence 1000110

Y	Q_2	Q_1	Q_0
1	0	1	0
1	1	0	1
0	1	1	0
0	0	1	1
0	0	0	1
1	0	0	0
0	1	0	0

Step 3: K-map for output *Y*.

Q_0 \ Q_2Q_1	00	01	11	10
0	1	1	0	0
1	0	0	×	1

$$Y = \overline{Q}_2\overline{Q}_0 + Q_2Q_0$$
$$= (Q_2 \odot Q_0)$$

Step 4: The logic diagram to generate the defined sequence is shown in Fig. 5.69.

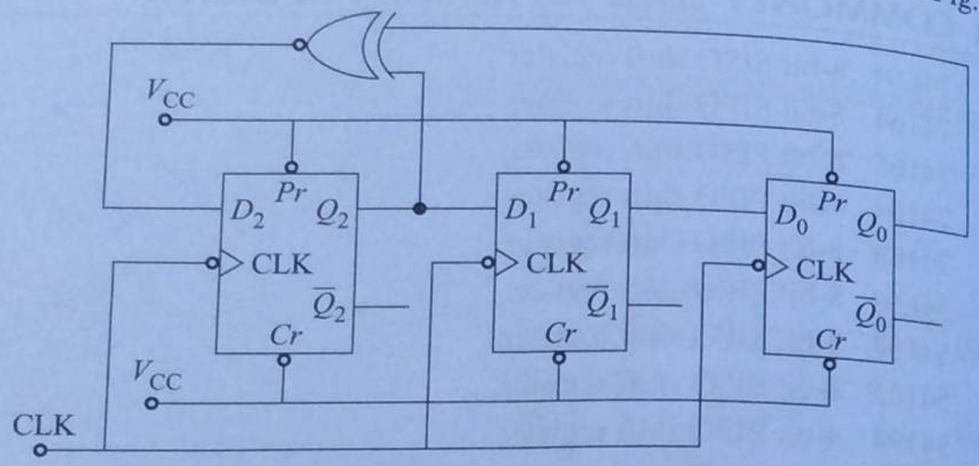


Fig. 5.69 Logic diagram to generate the sequence 1000110

Example 5.10 Design a sequence detector using the shift register for the following sequence: 10011.

Solution

The logic diagram of a sequence detector is shown in Fig. 5.70.

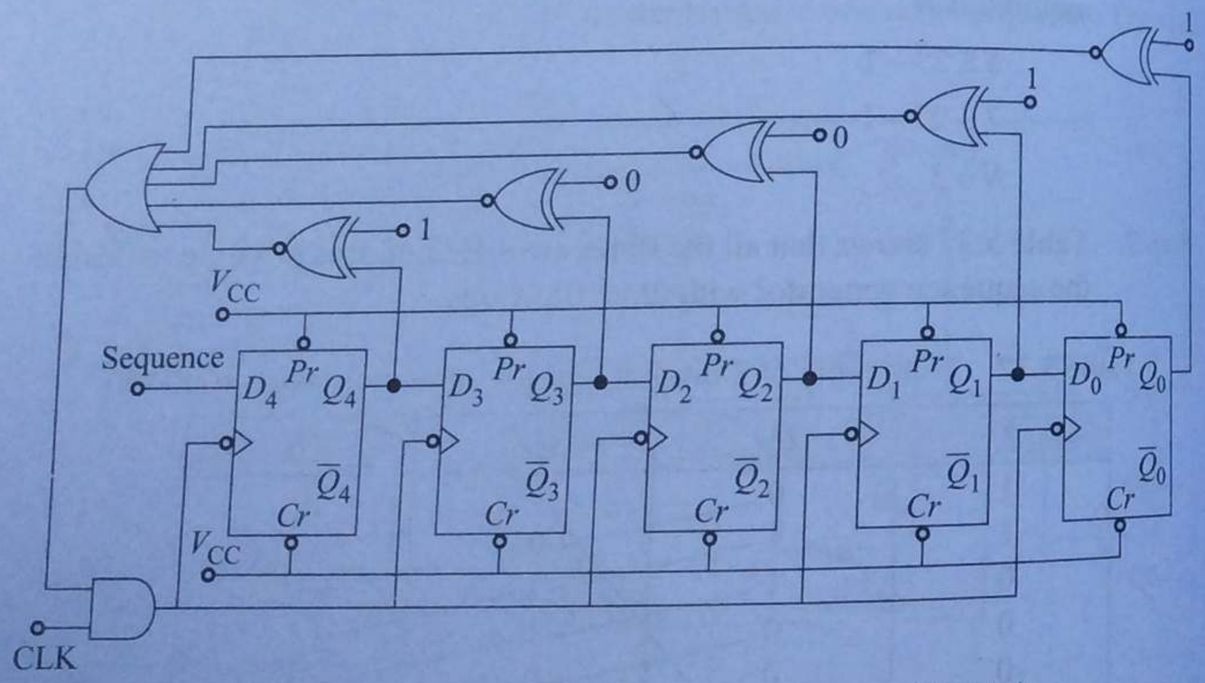


Fig. 5.70 Logic diagram of sequence detector (10011)

The output of flip-flop is compared with corresponding bits of the desired sequence. When it is matched, then the outputs of all Ex-NOR gates are 0, the outputs of NOR and AND gates are 0, and the clock given to the flip-flop is stopped.

5.14 RIPPLE COUNTER

The ripple counter can be designed using T or J-K flip-flops. The flip-flops are used in toggle mode. The logic diagram of a 4-bit ripple counter using J-K flip-flop is shown in Fig. 5.71.

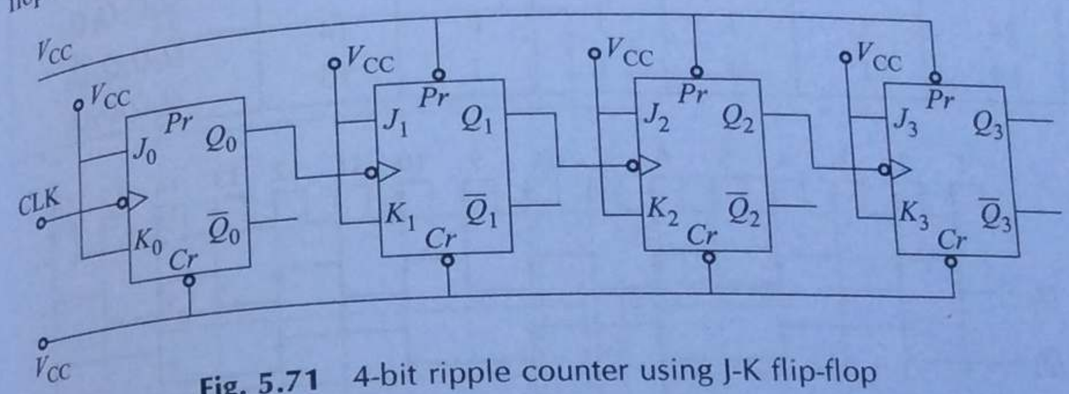


Fig. 5.71 4-bit ripple counter using J-K flip-flop

The ripple counter makes efficient use of flip-flops. The *n*-bit ripple counter uses *n* flip-flops, which counts 2^n clock pulses. In this counter, the flip-flops are not clocked simultaneously, the output of previous flip-flop becomes the clock input for the next flip-flop, and hence, it is also known as *asynchronous counter*.

The inputs *J* and *K* of flip-flops are connected to logic 1. The flip-flop acts as a toggle switch, and the output of the flip-flop is complement to the negative going pulse. Q_0 is complement to the negative going input clock pulse. Q_1 is complemented, when Q_0 changes from 1 to 0. Q_2 is complemented, when Q_1 changes from 1 to 0. Similarly Q_3 is complemented, when Q_2 changes from 1 to 0. The counting sequence of a 4-bit ripple counter is given in Table 5.38 and the waveforms are shown in Fig. 5.72.

Table 5.38 Counting sequence of 4-bit ripple counter

CLK	Q_3	Q_2	Q_1	Q_0	*Count*	*Boolean equation*
0	0	0	0	0	0	$\overline{Q}_3\overline{Q}_2\overline{Q}_1\overline{Q}_0$
1	0	0	0	1	1	$\overline{Q}_3\overline{Q}_2\overline{Q}_1Q_0$
2	0	0	1	0	2	$\overline{Q}_3\overline{Q}_2Q_1\overline{Q}_0$
3	0	0	1	1	3	$\overline{Q}_3\overline{Q}_2Q_1Q_0$
4	0	1	0	0	4	$\overline{Q}_3Q_2\overline{Q}_1\overline{Q}_0$
5	0	1	0	1	5	$\overline{Q}_3Q_2\overline{Q}_1Q_0$
6	0	1	1	0	6	$\overline{Q}_3Q_2Q_1\overline{Q}_0$
7	0	1	1	1	7	$\overline{Q}_3Q_2Q_1Q_0$

Contd.

Contd.

CLK	Q_3	Q_2	Q_1	Q_0	Count	Boolean equation
8	1	0	0	0	8	$Q_3\overline{Q}_2\overline{Q}_1\overline{Q}_0$
9	1	0	0	1	9	$Q_3\overline{Q}_2\overline{Q}_1Q_0$
10	1	0	1	0	10	$Q_3\overline{Q}_2Q_1\overline{Q}_0$
11	1	0	1	1	11	$Q_3\overline{Q}_2Q_1Q_0$
12	1	1	0	0	12	$Q_3Q_2\overline{Q}_1\overline{Q}_0$
13	1	1	0	1	13	$Q_3Q_2\overline{Q}_1Q_0$
14	1	1	1	0	14	$Q_3Q_2Q_1\overline{Q}_0$
15	1	1	1	1	15	$Q_3Q_2Q_1Q_0$

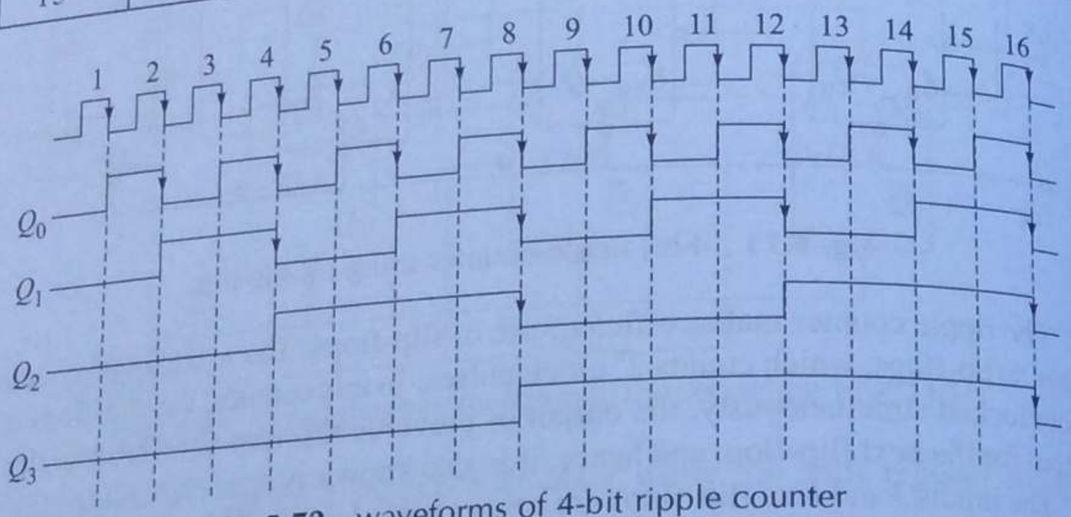


Fig. 5.72 waveforms of 4-bit ripple counter

The decoder circuit for decoding the count is shown in Fig. 5.73.

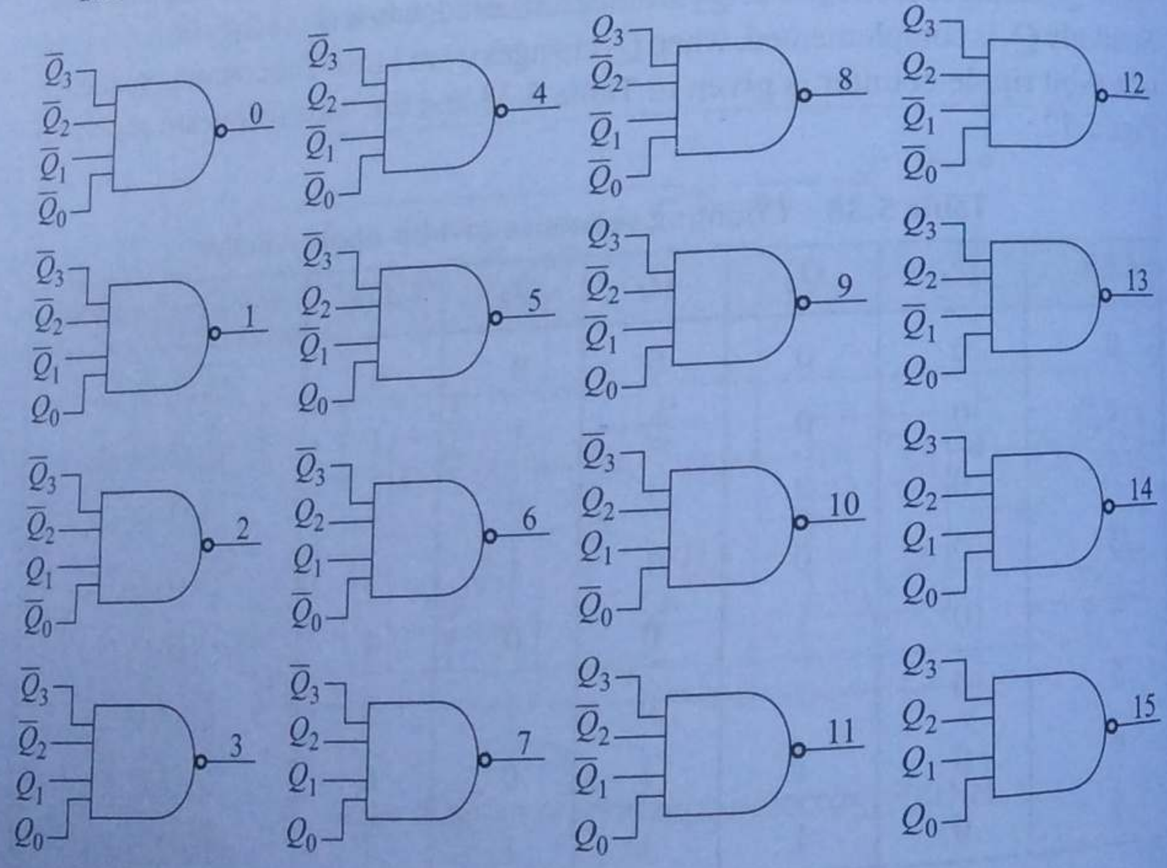


Fig. 5.73 Decoder circuit

The output of the flip-flop does not change exactly with the falling edge of the clock pulse, because of the delay of flip-flops. Hence a wrong pulse is present for a short duration. This wrong pulse is known as a *spike*. For example, consider the initial output of ripple counter is 0111. For the next clock pulse, Q_0 changes from 1 to 0 after a delay of Δt_d (Δt_d is delay of flip-flop), Q_1 changes from 1 to 0 after a delay of $2\Delta t_d$, Q_2 changes from 1 to 0 after a delay of $3\Delta t_d$ as shown in Fig. 5.74. The spike occurs at the output of decoder 6 and 4 for the duration of Δt.

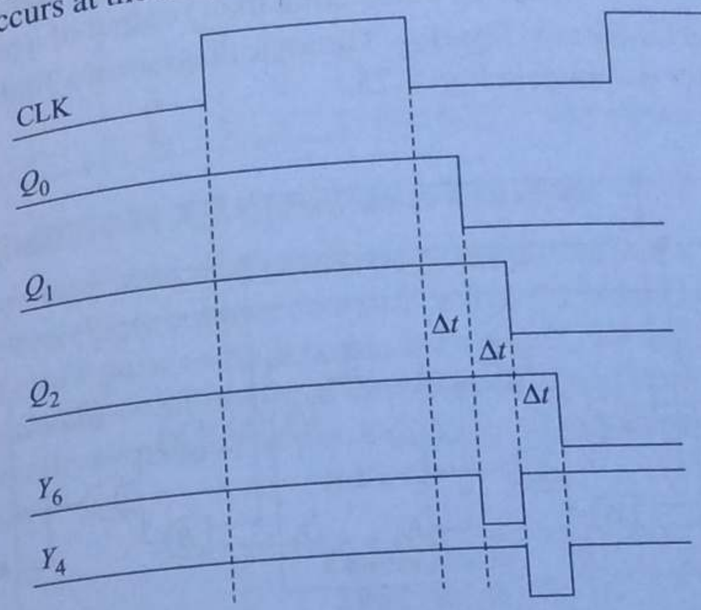


Fig. 5.74 Spike at the output of decoder

These spikes at the decoder's output can be eliminated using a strobe pulse input. The strobe signal is given to the decoder with a delay greater than or equal to $n \times \Delta t_d$, where n is the number of bits of the ripple counter. The decoding will occur only when all the flip-flops come to a steady state. The frequency of the clock pulse for reliable operation of the counter is

$$f \leq \frac{1}{N(t_d) + T_s}$$

where N = number of flip-flops,
t_d = propagation delay of flip-flop, and
T_S = strobe pulse width.

Example 5.11 Find the maximum frequency of a clock pulse at which the 4-bit ripple counter operates reliably. Assume delay of the flip-flop is 40 ns and the pulse width of strobe signal is 25 ns.

Solution

Given: $N = 4$, $t_d = 40$ ns, $T_S = 25$ ns.

$$f \leq \frac{1}{N(t_d) + T_s}$$

$$\leq \frac{1}{4(40 \times 10^{-9}) + 25 \times 10^{-9}}$$

$$\leq 5.4 \text{ MHz}$$

$$f_{max} = 5.4 \text{ MHz}$$

5.14.1 UP/DOWN Asynchronous Counter

The UP/DOWN counter counts the output increase/decrease with the clock pulse. In an UP counter, the output count is increased with the clock pulse, as explained in ripple counter. In a DOWN counter, the output count is decreased with the clock pulse. The UP asynchronous counter is obtained by connecting the Q output of a previous flip-flop to the clock input of the next flip-flop and the DOWN asynchronous counter is obtained by connecting the $\overline{Q}$ output of a previous flip-flop to the clock input for the next flip-flop. The logic diagram of a 3-bit UP/DOWN asynchronous counter is shown in Fig. 5.75.

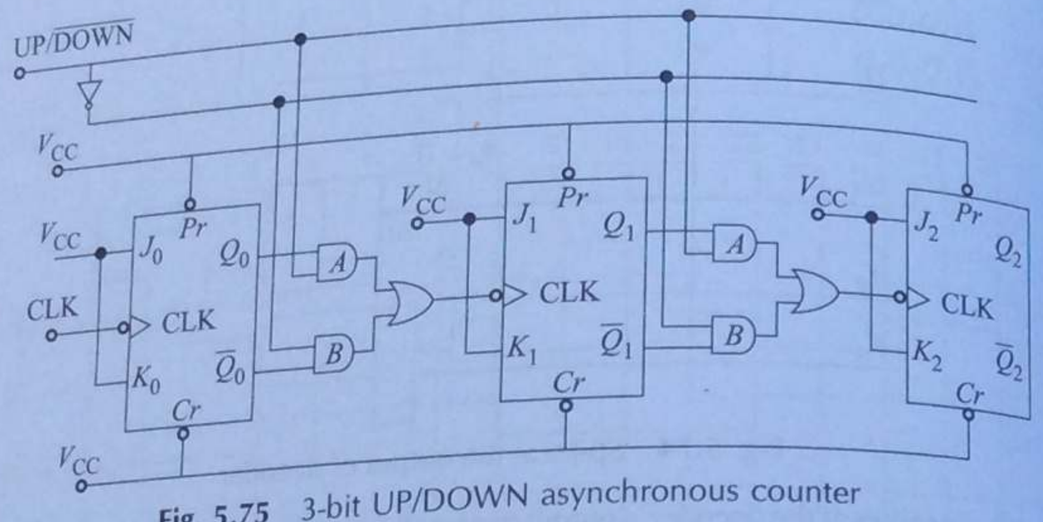


Fig. 5.75 3-bit UP/DOWN asynchronous counter

The inputs $J = K = V_{CC}$, and the flip-flops act as a toggle switch. When UP/$\overline{\text{DOWN}}$ is logic 1, the gates 'A' are enabled and the output Q of the previous flip-flop acts as a clock for the next flip-flop. Since flip-flops are negative edge-triggered, the output of flip-flops changes from 0 to 1 or 1 to 0 with the falling edge of Q, and the circuit acts as an UP counter. The waveforms are shown in Fig. 5.76.

Table 5.39 Count of UP counter

Clock	Q_2	Q_1	Q_0
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

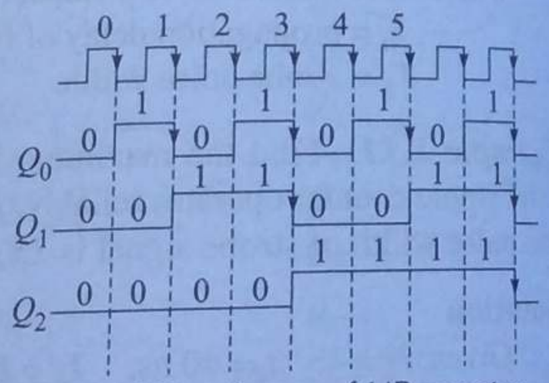


Fig. 5.76 Waveforms of UP counter

When UP/$\overline{\text{DOWN}}$ is logic 0, the gates 'B' are enabled and the output $\overline{Q}$ of previous flip-flop acts as the clock pulse for the next flip-flop. Since flip-flops are negative edge-triggered, the output of flip-flops changes from 0 to 1 or 1 to 0 with the falling edge of $\overline{Q}$, that is with the rising edge of Q, and the circuit acts as a DOWN counter. The waveforms are shown in Fig. 5.77.

Table 5.40 Count of DOWN counter

Clock	Q_2	Q_1	Q_0
0	0	0	0
1	0	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

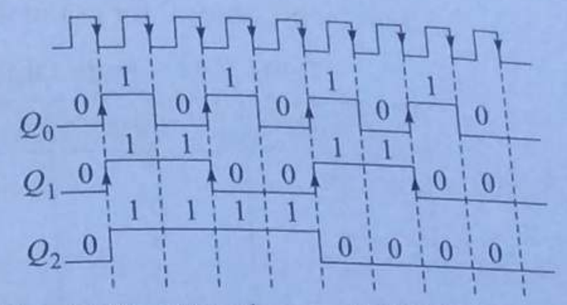


Fig. 5.77 Waveforms of DOWN counter

5.14.2 Modulus-M Asynchronous Counter

We have seen that, n-bit asynchronous counters count $N = 2^n$ clock pulses. For example, a 3-bit ripple counter counts $2^3 = 8$ clock pulses and the count is 0 to 7. To count M clock pulses which is less than $N = 2^n$, we need to take the help of a reset terminal. The combinational circuit is designed such that all the flip-flops are reset after the count M. The block diagram of modulus M counter is shown in Fig. 5.78.

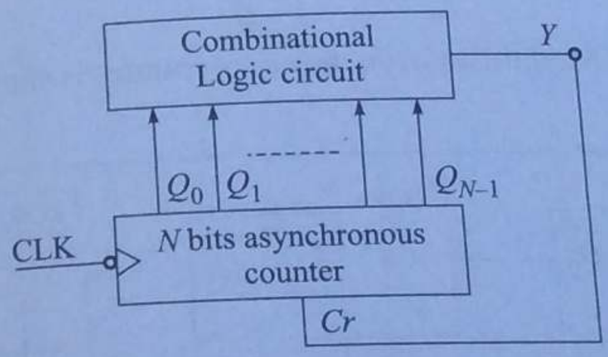


Fig. 5.78 Block diagram of Modulus-M counter

The output of a combinational logic circuit must be 0 after the M clock pulse, which resets the flip-flops. The procedure to design a Modulus-M asynchronous counter is as follows:

1. Find the minimum number of flip-flops required.
   For a Modulus-M counter, the minimum number of flip-flops required is n, $M < 2^n$.
2. Prepare the sequence and design the combinational circuit such that all the flip-flops are reset after the M clock pulse.
   (a) Draw the truth table of a ripple counter with the output of a combinational logic circuit Y, such that $Y = 1$ for the valid state and $Y = 0$ for the invalid state.
   (b) Draw the K-map for output Y and simplify.

Let us consider the example of a Modulus-4 (Mod-4) counter.

1. The minimum number of flip-flops required is
   $M < 2^n$ or $4 < 2^n$ or $n = 3$
2. Design the combinational logic circuit such that the flip-flops are clear after four clock pulses.

(a) The truth table of a 3-bit UP counter is given in Table 5.41, where Y is the output of the combinational circuit. It is equal to 1 for counts 0 to 3 and it becomes 0 for count 4.

Table 5.41 Truth table of 3-bit UP counter with output Y

Clock	Q_2	Q_1	Q_0	Y
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	X
7	1	1	1	X

(b) K-map for Y:

Q_n \ Q_2Q_1	00	01	11	10
0	1	1	×	0
1	1	1	×	×

$Y = \overline{Q}_2$

The logic circuit of Modulus-4 asynchronous counter is shown in Fig. 5.79.

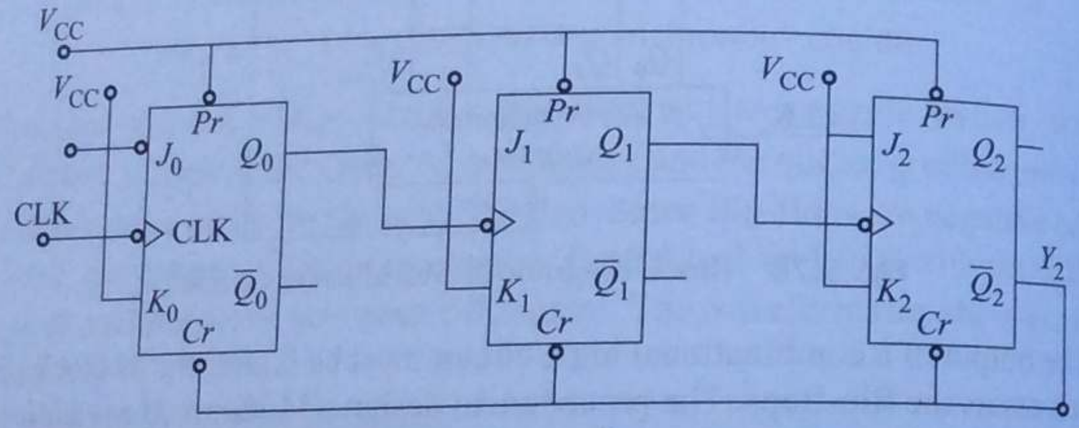


Fig. 5.79 Modulus-4 asynchronous counter

The waveforms of Mod-4 ripple counter are shown in Fig. 5.80.

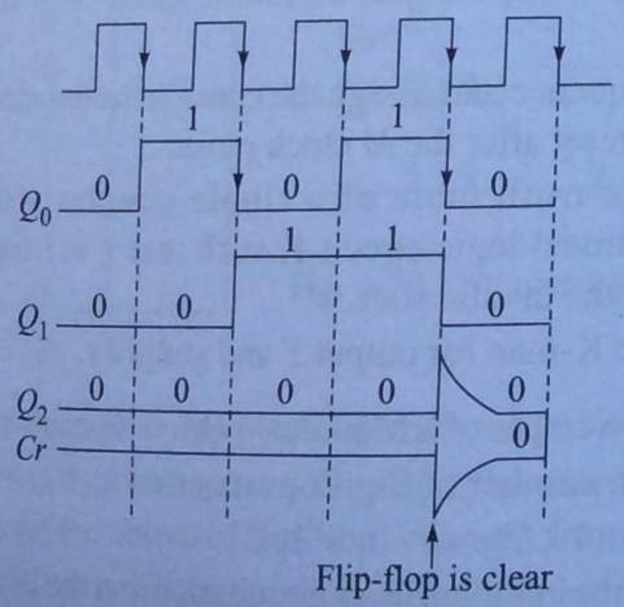


Fig. 5.80 Waveforms of Mod-4 ripple counter

Example 5.11 Design a Mod-9 ripple counter using T flip-flop and draw the output waveforms.

Solution

1. The minimum number of flip-flops required is

$$M \le 2^n \quad \text{or} \quad 9 \le 2^n \quad \text{or} \quad n = 4$$

2. Design the combinational logic circuit such that the flip-flops are clear after nine clock pulses.

(a) The truth table of a 4-bit UP counter is given in Table 5.42, where Y is the output of the combinational circuit. It is equal to 1 for the counts 0 to 8 and it becomes 0 for count 9.

Table 5.42 Truth table of 4-bit UP counter with output Y

Clock	Q_3	Q_2	Q_1	Q_0	Y
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	X
11	1	0	1	1	X
12	1	1	0	0	X
13	1	1	0	1	X
14	1	1	1	0	X
15	1	1	1	1	X

(b) K-map for Y:

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	1	1	×	1
01	1	1	×	0
11	1	1	×	×
10	1	1	×	×

$Y = \overline{Q}_3 + \overline{Q}_0$

The logic circuit of Mod-9 asynchronous counter is shown in Fig. 5.81.

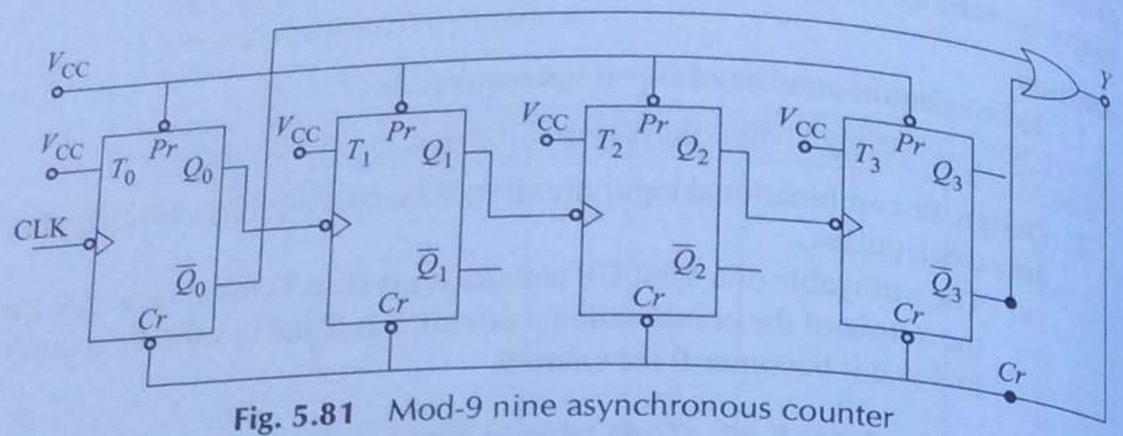


Fig. 5.81 Mod-9 nine asynchronous counter

The waveforms of Mod-9 ripple counter are shown in Fig. 5.82.

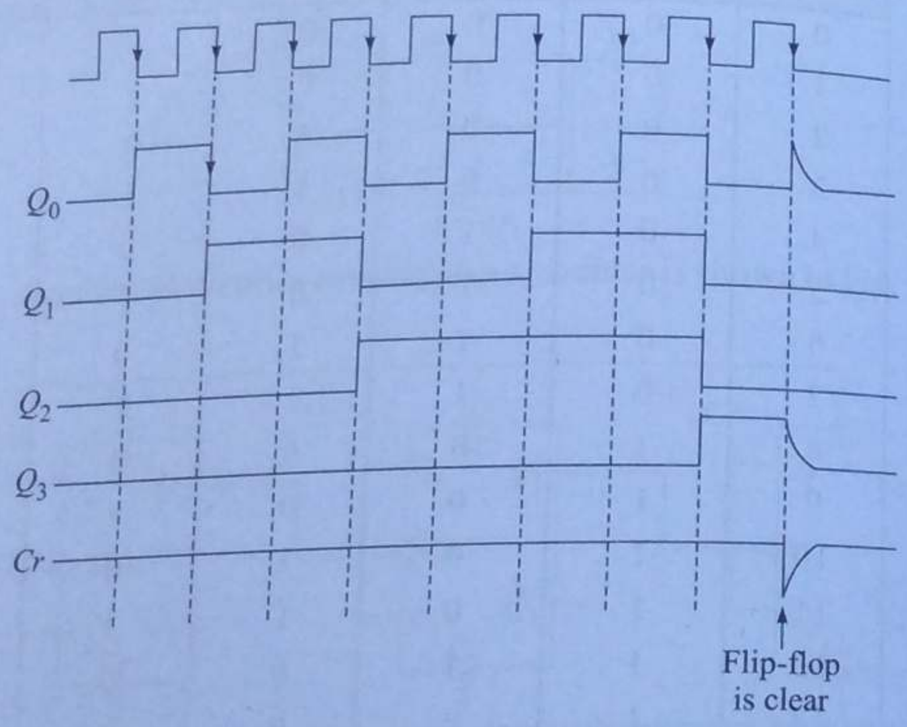


Fig. 5.82 Waveforms of Mod-9 ripple counter

5.14.3 Commonly used ICs for Asynchronous Counter

The commonly used TTL ICs of asynchronous counter are given below. On the basis of their features, ICs are classified into three groups: group A, group B, and group C.

Group A :

7490	BCD counter
74290	BCD counter
74490	Dual BCD counter

Group B :

7492	Divided-by-12 counter
7493	4-bit binary counter

74390	Dual decade counter
74393	Dual 4-bit binary counter

Group C :

74176	Presettable BCD counter
74196	Presettable BCD counter
74177	Presettable 4-bit binary counter

IC 7490

IC 7490 is a BCD asynchronous counter. It consists of four flip-flops, internally connected so as to provide Mod-2 and Mod-5 counter functions, as shown in Fig. 5.83. It comes under the group of set and reset and referred as group A. These ICs have set and reset inputs. These inputs help in designing a Modulus-M counter.

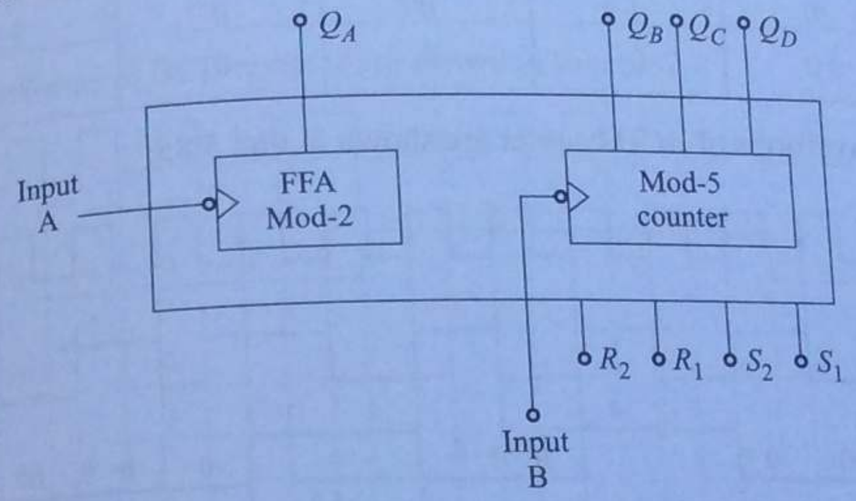


Fig. 5.83 Block diagram of IC 7490

The Mod-2 and Mod-5 counters can be used independently or in combinations. Q_A is the output of Mod-2 counter and Q_D, Q_C, Q_B are the outputs of Mod-5 counter. R_1 and R_2 are the reset inputs; these inputs are connected to 1 to reset the flip-flops. S_1 and S_2 are the set inputs; these inputs set the counter 1001 by connecting to logic 1. For normal operation, set and reset inputs are connected to 0.

IC 7490 is used as a BCD counter, when Q_A is connected as the clock input of Mod-5 counter and the external clock is given to the clock inputs of Mod-2, as shown in Fig. 5.84. The sequence of BCD counter is given in Table 5.43.

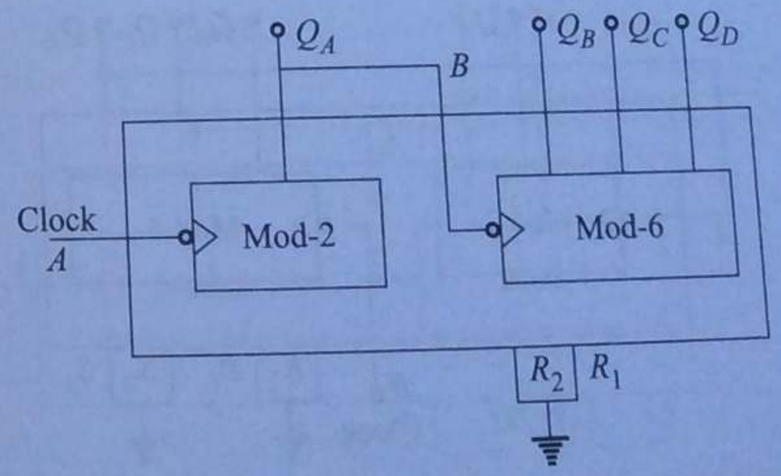


Fig. 5.84 IC 7490 as BCD counter (Mod-2 and Mod-5)

Table 5.43 Sequence of BCD counter (Mod-2 and Mod-5)

CLK	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	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0

The waveforms of BCD counter are shown in Fig. 5.85.

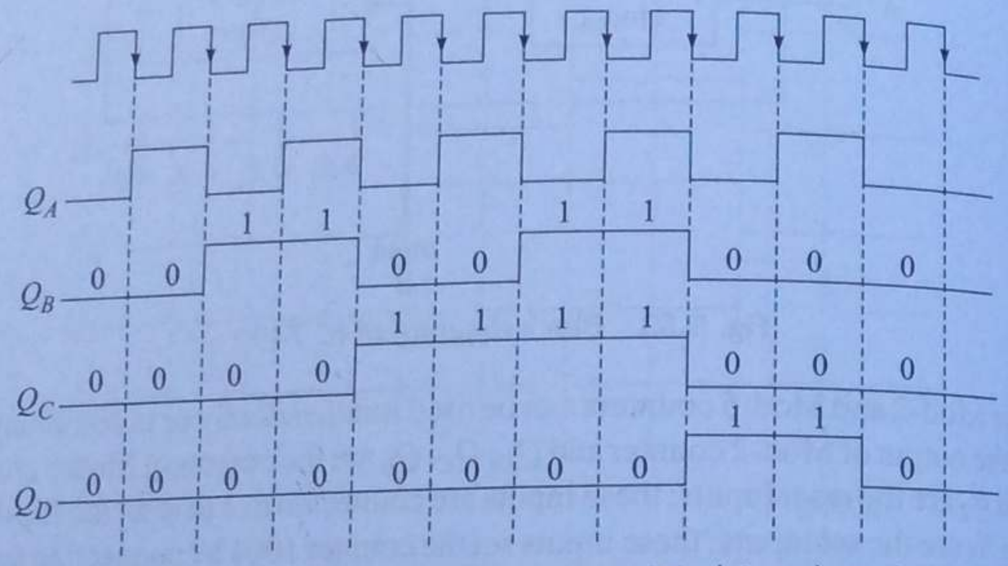


Fig. 5.85 Waveforms of BCD counter (Mod-2 and Mod-5)

Q_D is connected as the clock input of Mod-2 counter and the external clock is given to the clock inputs of Mod-5, as shown in Fig. 5.86.

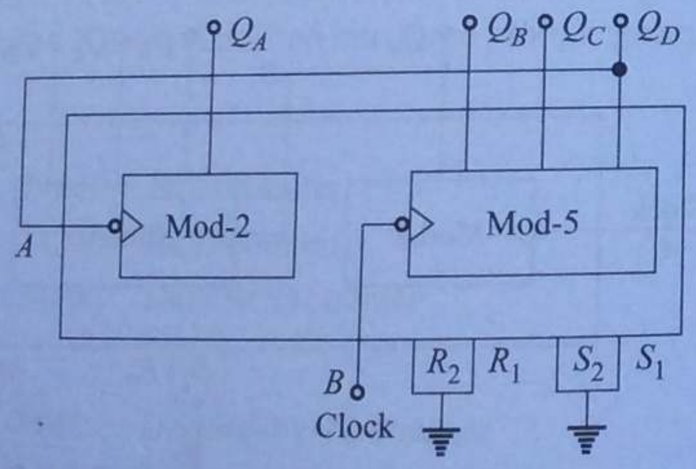


Fig. 5.86 IC 7490 as BCD counter (Mod-5 and Mod-2)

The sequence of BCD counter is given in Table 5.44.

Table 5.44 Sequence of BCD counter (Mod-5 and Mod-2)

CLK	Q_D	Q_C	Q_B	Q_A
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	0	0	0	1
6	0	0	1	1
7	0	1	0	1
8	0	1	1	1
9	1	0	0	1
10	0	0	0	0

The waveforms of BCD counter are shown in Fig. 5.87.

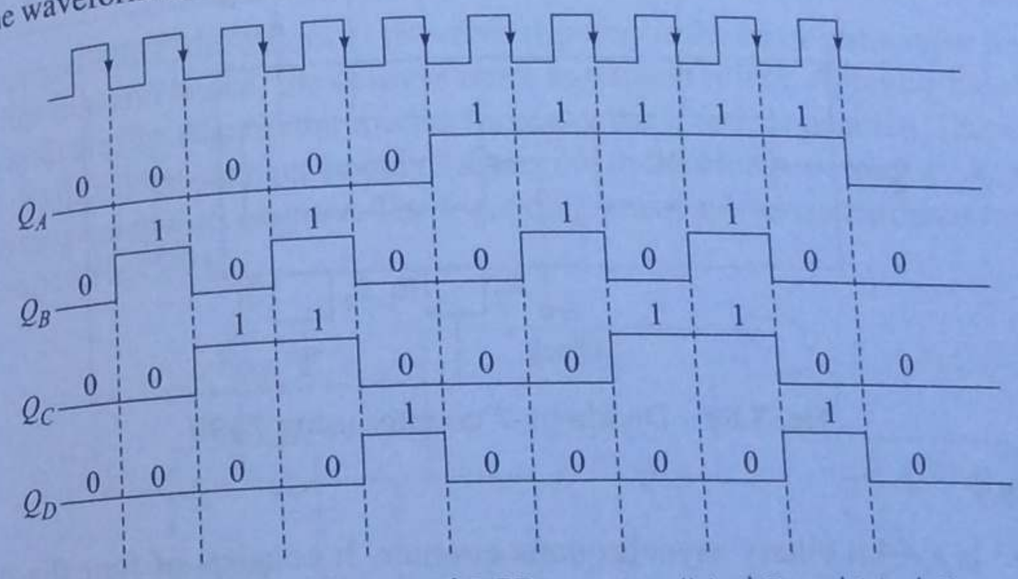


Fig. 5.87 Waveforms of BCD counter (Mod-5 and Mod-2)

Example 5.12 Design a divide-by-7 counter using 7490.

Solution

When Q_A is connected as the clock input of Mod-5 counter and the external clock is given to the clock input of Mod-2, then 7490 acts as a BCD counter. It counts 0

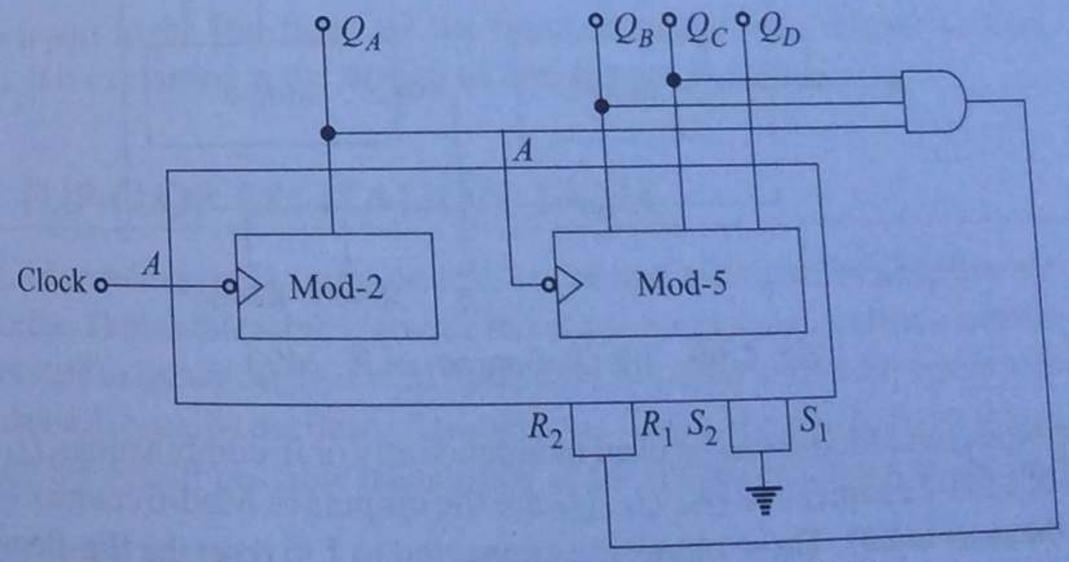


Fig. 5.88 Divide-by-7 counter using 7490

to 9. To design a divide-by-7 counter using 7490, we need to design the combinational logic circuit such that when the counter reaches 7 (0111), the output of the combinational circuit is high, which is connected to reset input, and 7490 is reset, as shown in Fig. 5.88.

Example 5.13 Design a divide-by-4 counter using 7490.

Solution

IC 7490 consists of four flip-flops, internally connected so as to provide Mod-2 and Mod-5 counter functions. When the clock is given to Mod-5 counter (input *B*), the counter acts as a Mod-5 counter and the output count is available at Q_D, Q_C, and Q_B. To design a divide-by-4 counter using 7490, we need to design the combinational logic circuit such that when the counter reaches 4 (0100), the output of the combinational circuit is high, which is connected to reset input, and 7490 is reset, as shown in Fig. 5.89.

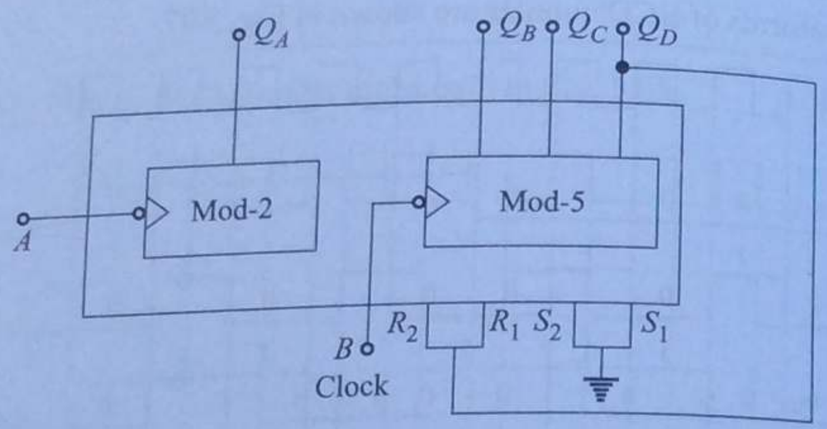


Fig. 5.89 Divide-by-7 counter using 7490

IC 7493

IC 7493 is a 4-bit binary asynchronous counter. It consists of four flip-flops, internally connected so as to provide Mod-2 and Mod-6 counter functions, as shown in Fig. 5.90. It comes under the group of reset, referred as group B. This IC has reset inputs; these inputs help in designing a Modulus-M counter.

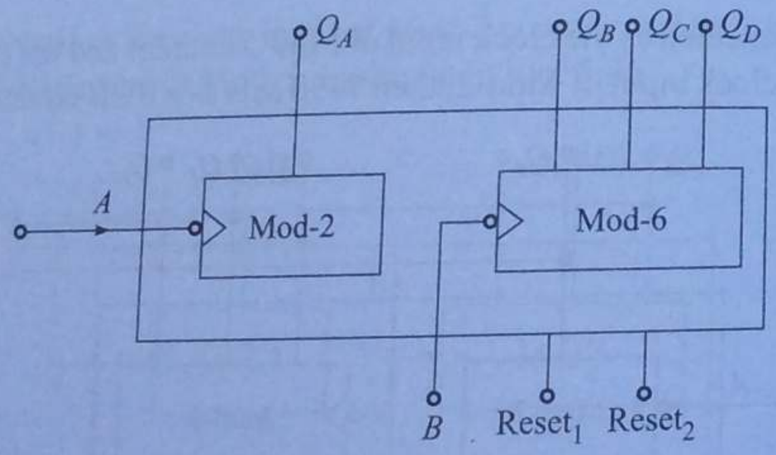


Fig. 5.90 Block diagram of IC 7493

The Mod-2 and Mod-6 can be used independently or in combinations. Q_A is the output of Mod-2 counter and Q_D, Q_C, Q_B are the outputs of Mod-6 counter. R_1 and R_2 are the reset inputs. These inputs are connected to 1 to reset the flip-flops. The

Mod-6 counter does not count in a straight binary sequence. The sequence of Mod-6 counter is given in the Table 5.45.

Table 5.45 Sequence of Mod-6 counter

CLK	Q_D	Q_C	Q_B
0	0	0	0
1	0	0	1
2	0	1	0
3	1	0	0
4	1	0	1
5	1	1	0

This IC is not used as a counter, but it is used for frequency division.

5.15 SYNCHRONOUS COUNTER

In a ripple counter, the output of previous flip-flop is the clock pulse input for the next flip-flop and hence, the delay is more and speed is less. Applying the clock pulse to all the flip-flops of the counter increases the speed of operation. The output of all the flip-flops changes in synchronism with the clock and such a counter is called as *synchronous counter*. The logic diagram of a 3-bit synchronous counter is shown in Fig. 5.91.

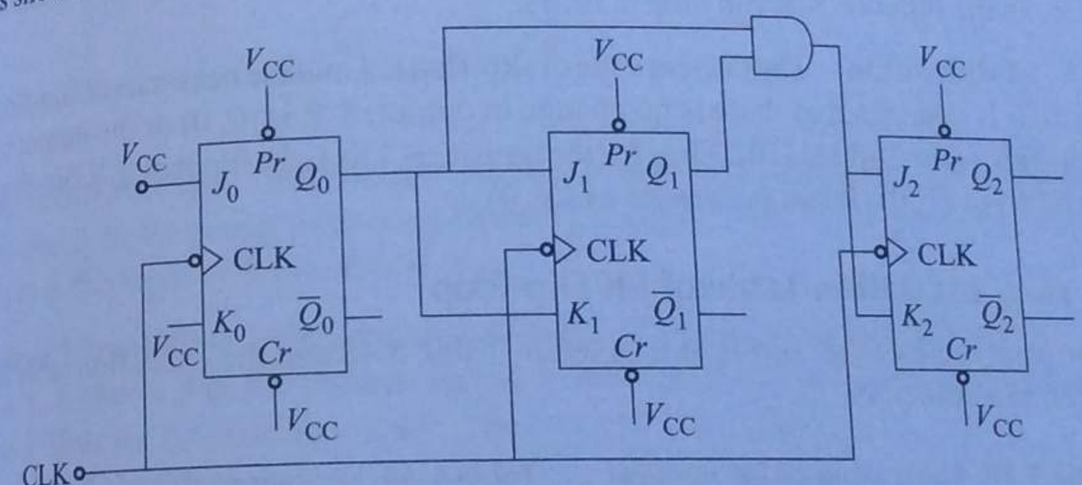


Fig. 5.91 3-bit synchronous counter

The inputs to the flip-flops are the function of previous outputs or logic 0 or logic 1. It is explained in the design of synchronous counter.

5.16 FLIP-FLOP EXCITATION TABLE

The problem with synchronous circuits is defined in terms of state tables or a set of statements. These state tables provide the sequence of states and the transition from present state to its next state. From these data, we need to find the inputs to the flip-flops that will provide the desired transitions. The table which shows the inputs of the flip-flop for all possible transitions at the output is known as the excitation table.

5.16.1 Excitation Table of R-S Flip-Flop

The truth table of R-S flip-flop is given in Table 5.46 and the excitation table is given in Table 5.47.

Table 5.46 Truth table of R-S flip-flop

S_n	R_n	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	?

Table 5.47 Excitation table of R-S flip-flop

Present output Q_n	*Next output* Q_{n+1}	*Input* S_n	*Input* R_n
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

(0 → 0) transition The present state of flip-flop is 0 and the next state of flip-flop is also 0. It shows that there is no change in output, S-R input is (0, 0) or the output of flip-flop is reset, S-R is (0, 1). Thus, for the transition 0 to 0, the input of S-R flip-flop is (0, 0) or (0, 1). It can be written as (0, X).

(0 → 1) transition The present state of flip-flop is 0 and the next state of flip-flop is 1. It shows that the output of flip-flop is set, S-R is (1, 0). Thus, for the transition 0 to 1, the input of S-R flip-flop is (1, 0).

(1 → 0) transition The present state of flip-flop is 1 and the next state of flip-flop is 0. It shows that the output of flip-flop is reset, S-R is (0, 1). Thus for the transition 1 to 0, the input of S-R flip-flop is (0, 1).

(1 → 1) transition The present state of flip-flop is 1 and the next state of flip-flop is also 1. It shows that there is no change in output, S-R is (0, 0) or the output of flip-flop is set, S-R is (1, 0). Thus for the transition 1 to 1, the input of S-R flip-flop is (0, 0) or (1, 0). It can be written as (X, 0).

5.16.2 Excitation Table of J-K Flip-Flop

The truth table of J-K flip-flop is given in Table 5.48 and the excitation table is given in Table 5.49.

Table 5.48 Truth table of J-K flip-flop

J_n	K_n	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_n$

Table 5.49 Excitation table of J-K flip-flop

Present output Q_n	*Next output* Q_{n+1}	*Input* J_n	*Input* K_n
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

(0 → 0) transition The present state of flip-flop is 0 and the next state of flip-flop is also 0. It shows that there is no change in output, J-K is (0, 0) or the output of flip-flop is reset, J-K is (0, 1). Thus for the transition 0 to 0, the input of J-K flip-flop is (0, 0) or (0, 1). It can be written as (0, X).

(0 → 1) transition The present state of flip-flop is 0 and the next state of flip-flop is 1. It shows that the present output is the complement of previous output, J-K is (1, 1) or the output of flip-flop is set, J-K is (1, 0). Thus for the transition 0 to 1, the input of J-K flip-flop is (1, 1) or (1, 0). It can be written as (1, X).

(1 → 0) transition The present state of flip-flop is 1 and the next state of flip-flop is 0. It shows that the present output is the complement of previous output, J-K is (1, 1) or the output of flip-flop is reset, J-K is (0, 1). Thus for the transition 1 to 0, the inputs of J-K flip-flop is (1, 1) or (0, 1). It can be written as (X, 1).

(1 → 1) transition The present state of flip-flop is 1 and the next state of flip-flop is also 1. It shows that there is no change in output, J-K is (0, 0) or the output of flip-flop is set, J-K is (1, 0). Thus for the transition 1 to 1, the input of J-K flip-flop is (0, 0) or (1, 0). It can be written as (X, 0).

5.16.3 Excitation Table of T Flip-Flop

The truth table of T flip-flop is given in Table 5.50 and the excitation table is given in Table 5.51.

Table 5.50 Truth table of T flip-flop

T_n	Q_{n+1}
0	Q_n
1	$\overline{Q}_n$

Table 5.51 Excitation table of T flip-flop

Present output Q_n	*Next output* Q_{n+1}	*Input* T_n
0	0	0
0	1	1
1	0	1
1	1	0

(0 → 0) transition The present state of flip-flop is 0 and the next state of flip-flop is also 0. It shows that there is no change in output, T is 0. Thus for the transition 0 to 0, the input of T flip-flop is 0.

(0 → 1) transition The present state of flip-flop is 0 and the next state of flip-flop is 1. It shows that the present output is the complement of previous output, T is 1.Thus for the transition 0 to 1, the input of T flip-flop is 1.

(1 → 0) transition The present state of flip-flop is 1 and the next sate of flip-flop is 0. It shows that the present output is the complement of previous output, T is 1.Thus for the transition 1 to 0, the input of T flip-flop is 1.

(1 → 1) transition The present state of flip-flop is 1 and the next state of flip-flop is also 1. It shows that there is no change in output, T is 0. Thus for the transition 1 to 1, the input of T flip-flop is 0.

5.16.4 Excitation Table of D Flip-Flop

The truth table of D flip-flop is given in Table 5.52 and the excitation table is given in Table 5.53.

Table 5.52 Truth table of D flip-flop

D_n	Q_{n+1}
0	0
1	1

Table 5.53 Excitation table of D flip-flop

Present output Q_n	Next output Q_{n+1}	Input D_n
0	0	0
0	1	1
1	0	0
1	1	1

(0 → 0) transition The present state is 0 and the next state is also 0. It shows that after the clock, the flip-flop is reset, input D is 0. Thus for the transition 0 to 0, the input of D flip-flop is 0.

(0 → 1) transition The present state of flip-flop is 0 and the next state of flip-flop is 1. It shows that after the clock, the flip-flop is set, D is 1. Thus for the transition 0 to 1, the input of D flip-flop is 1.

(1 → 0) transition The present state of flip-flop is 1 and the next state of flip-flop is 0. It shows that after the clock, the flip-flop is reset, D is 0. Thus for the transition 1 to 0, the input of D flip-flop is 0.

(1 → 1) transition The present state of flip-flop is 1 and the next state of flip-flop is 1. It shows that after the clock, the flip-flop is set, D is 1. Thus for the transition 1 to 1, the input of D flip-flop is 1.

5.17 SYNCHRONOUS COUNTER DESIGN

In a synchronous counter, the clock pulse is given simultaneously to all the flip-flops and the transition at the output is in synchronism with the clock inputs. The design procedure of synchronous counter is as follows:

1. Find the number of flip-flops required.
   For a Modulus-M counter, the number of flip-flops required is n, such that $M \leq 2^n$.
2. It is noted that an n-bit counter is also known as a modulo 2^n counter.
3. Write the given sequence in the form of present state and next state (i.e. Q_n and Q_{n+1}).
4. With the help of excitation table, find the inputs of flip-flop for the given transition.
5. Prepare K-map for each input of flip-flops in terms of output of the flip-flops and simplify.
6. Connect the inputs to the flip-flops as per the simplified Boolean equations.

Example 5.14 Design a 3-bit synchronous counter using J-K flip-flops.

Solution

For a 3-bit counter, three flip-flops are required, $n = 3$

The sequence in the form of present state and next state with the excitation table of J-K flip-flop is given in Table 5.54.

Table 5.54 Present state and next state with excitation table of J-K flip-flop

Present state			Next state			Inputs					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	0	0	1	1	X	0	X	0	X
0	0	1	0	1	0	X	1	1	X	0	X
0	1	0	0	1	1	1	X	X	0	0	X
0	1	1	1	0	0	X	1	X	1	1	X
0	0	0	1	0	1	1	X	0	X	X	0
0	0	1	1	1	0	X	1	1	X	X	0
1	0	1	1	1	1	1	X	X	0	X	0
1	1	0	1	1	0	X	1	X	1	X	1
1	1	1	0	0	0						

K-map for J_0

Q_0 \ Q_2Q_1	00	01	11	10
0	1	1	1	1
1	×	×	×	×

$J_0 = 1$

K-map for K_0

Q_0 \ Q_2Q_1	00	01	11	10
0	×	×	×	×
1	1	1	1	1

$K_0 = 1$

K-map for J_1

Q_0 \ Q_2Q_1	00	01	11	10
0	0	×	×	0
1	1	×	×	1

$J_0 = Q_0$

K-map for K_1

Q_0 \ Q_2Q_1	00	01	11	10
0	×	0	0	×
1	×	1	1	×

$K_1 = Q_0$

K-map for J_2

Q_0 \ Q_2Q_1	00	01	11	10
0	0	0	×	×
1	0	1	×	×

$J_2 = Q_1Q_0$

K-map for K_2

Q_0 \ Q_2Q_1	00	01	11	10
0	×	×	0	0
1	×	×	1	0

$K_2 = Q_1Q_0$

The 3-bit synchronous counter using J-K flip-flop is shown in Fig. 5.92.

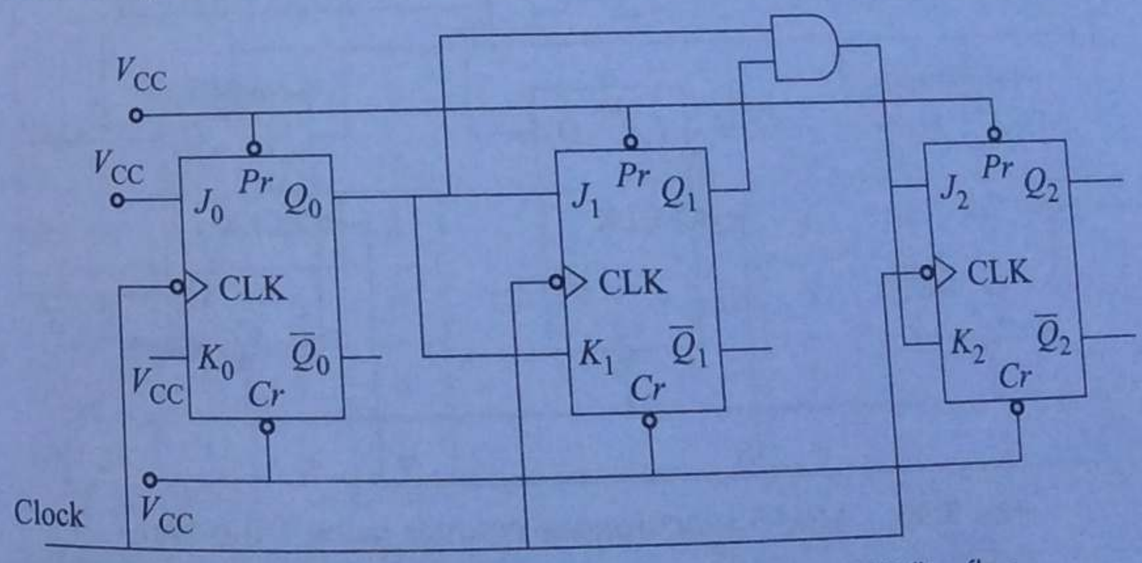


Fig. 5.92 3-bit synchronous counter using J-K flip-flop

Example 5.15 Design a Mod-5 synchronous counter using T flip-flops.

Solution

For a Mod-M counter, the number of flip-flops required is n.

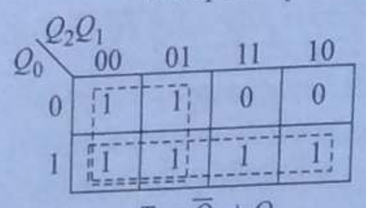
$$M \le 2^n \quad \text{or} \quad 5 \le 2^n \quad \text{or} \quad n = 3$$

The sequence in the form of present state and next state with the excitation table of T flip-flop is given in Table 5.55.

Table 5.55 Present state and next state with excitation table of T flip-flop

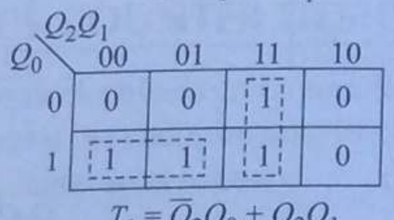
Present state			*Next state*			*Inputs*		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	T_0	T_1	T_2
0	0	0	0	0	1	1	0	0
0	0	1	0	1	0	1	1	0
0	1	0	0	1	1	1	0	0
0	1	1	1	0	0	1	1	1
1	0	0	0	0	0	0	0	1
1	0	1	0	0	0	1	0	1
1	1	0	0	0	0	0	1	1
1	1	1	0	0	0	1	1	1

K-map for T_0



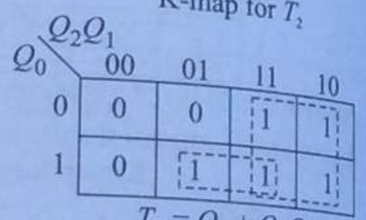
$T_0 = \overline{Q}_2 + Q_0$

K-map for T_1



$T_1 = \overline{Q}_2 Q_0 + Q_2 Q_1$

K-map for T_2



$T_2 = Q_2 + Q_1 Q_0$

The Mod-5 synchronous counter using T flip-flop is shown in Fig. 5.93.

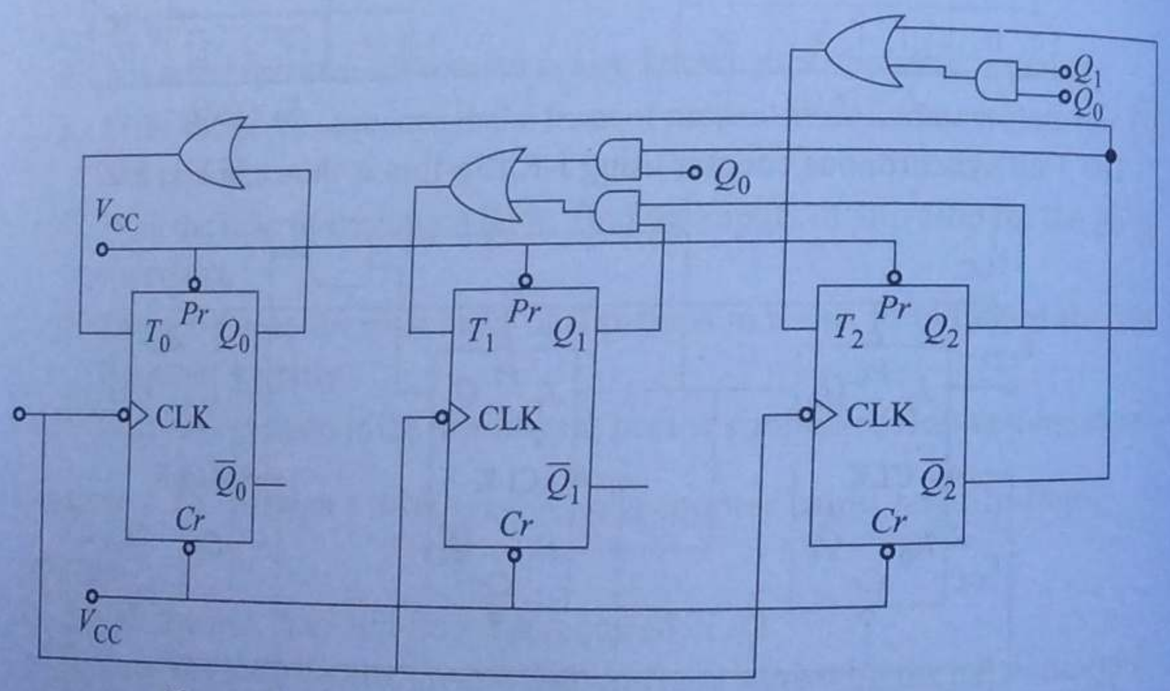


Fig. 5.93 Mod-5 synchronous counter using T flip-flop

Example 5.16 Design a Mod-12 synchronous counter using D flip-flops.

Solution

For Mod-M counter, the number of flip-flops required is n.

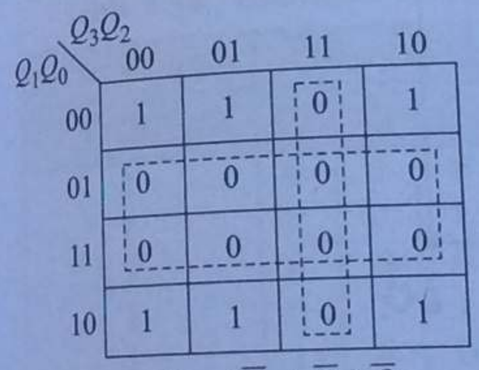
$$M \le 2^n \quad \text{or} \quad 12 \le 2^n \quad \text{or} \quad n = 4$$

The sequence in the form of present state and next state with the excitation table of D flip-flop is given in Table 5.56.

Table 5.56 Present state and next state with excitation table of T flip-flop

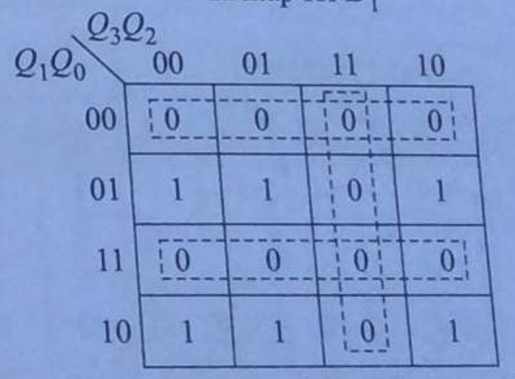
Present state				*Next state*				*Inputs*			
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	D_0	D_1	D_2	D_3
0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	1	0	0	1	0	0	1	0	0
0	0	1	0	0	0	1	1	1	1	0	0
0	0	1	1	0	1	0	0	0	0	1	0
0	1	0	0	0	1	0	1	1	0	1	0
0	1	0	1	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1	1	1	1	0
0	1	1	1	1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	1	0	0	1
1	0	0	1	1	0	1	0	0	1	0	1
1	0	1	0	1	0	1	1	1	1	0	1
1	0	1	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0

K-map for D_0



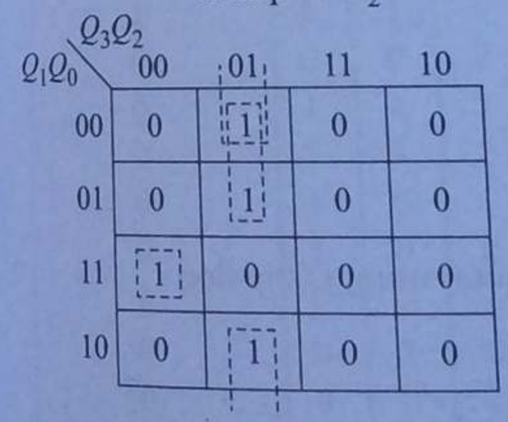
$D_0 = (\overline{Q}_3 + \overline{Q}_2)\,\overline{Q}_0$

K-map for D_1



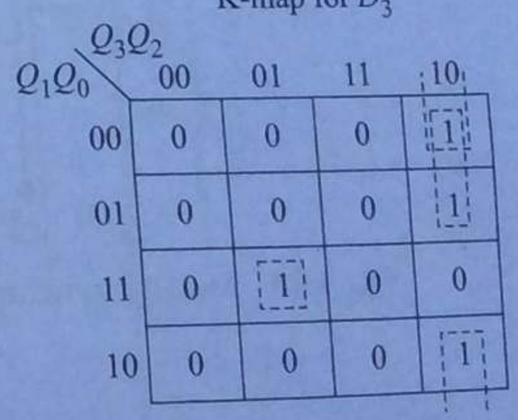
$D_1 = (Q_1 + Q_0)\,(\overline{Q}_3 + \overline{Q}_2)\,(\overline{Q}_1 + \overline{Q}_0)$

K-map for D_2



$D_2 = \overline{Q}_3 Q_2 \overline{Q}_1 + \overline{Q}_3 Q_2 \overline{Q}_0 + \overline{Q}_3 \overline{Q}_2 Q_1 Q_0$

K-map for D_3



$D_3 = Q_3 \overline{Q}_2 \overline{Q}_1 + Q_3 \overline{Q}_2 \overline{Q}_0 + \overline{Q}_3 Q_2 Q_1 Q_0$

The Mod-12 synchronous counter using D flip-flop is shown in Fig. 5.94.

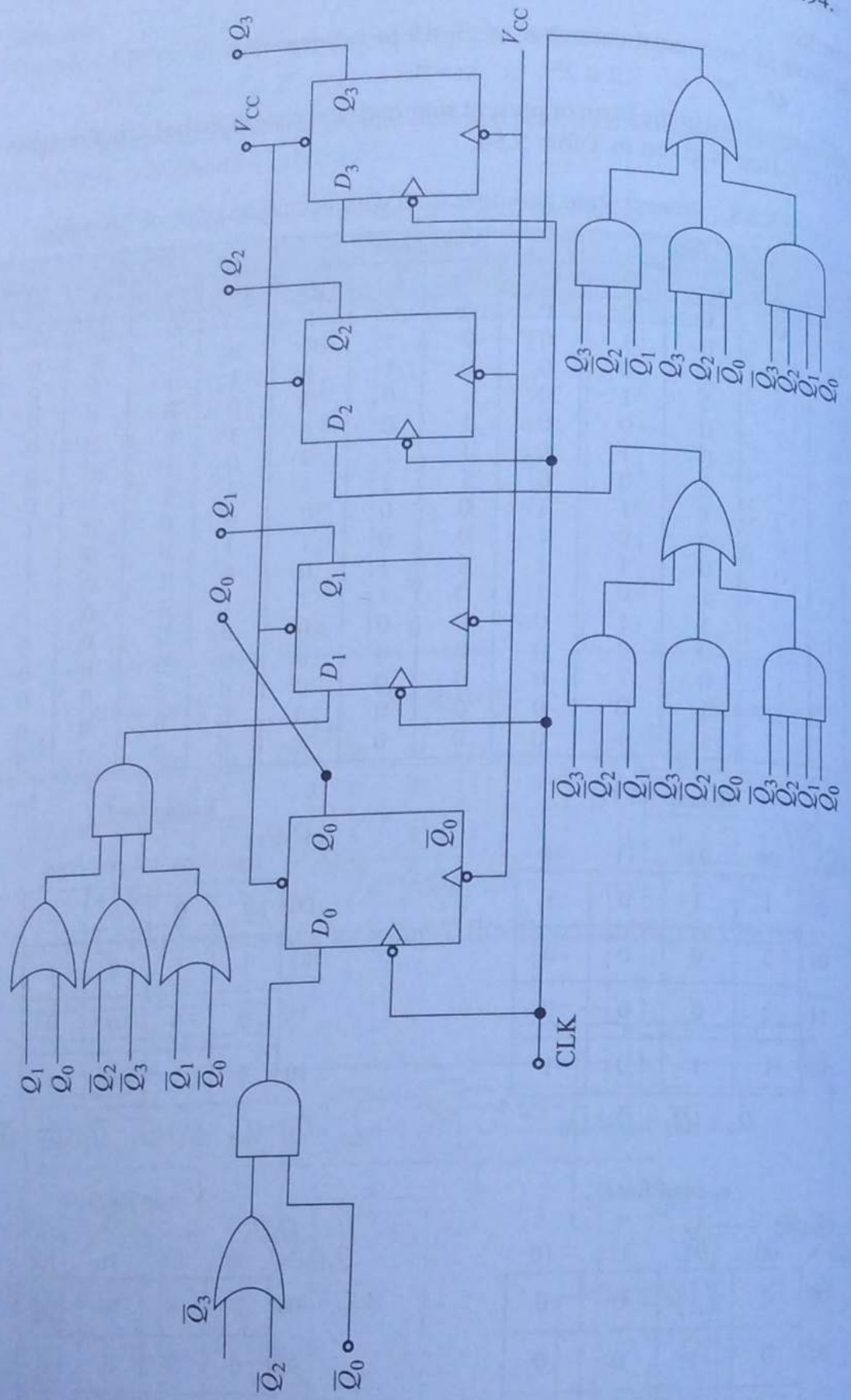


Fig. 5.94 Mod-12 synchronous counter using *D* flip-flop

5.18 UP/DOWN COUNTER

The UP/DOWN counter counts in increasing or decreasing direction as per the control inputs. Whenever the circuit is designed to operate for more than one operation, the control inputs are provided to control the operation. Let us consider $UP/\overline{DOWN}$ is a control input, which controls the direction of the counter. When $UP/\overline{DOWN}$ input is 0, the direction of counter is in decreasing direction and the counter is said to be a *down counter*. When $UP/\overline{DOWN}$ input is 1, the direction of counter is in increasing direction and the counter is said to be an *up counter*. The state table of 3-bit $UP/\overline{DOWN}$ synchronous counter is given in Table 5.57.

Table 5.57 State table of 3-bit $UP/\overline{DOWN}$ synchronous counter

$UP/\overline{DOWN} = M$	Q_0	Q_1	Q_2	$UP/\overline{DOWN}$	Q_0	Q_1	Q_2
1	0	0	0	0	1	1	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	0	1
1	0	1	1	0	1	0	0
1	1	0	0	0	0	1	1
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	0

The excitation table of 3-bit UP/DOWN counter for J-K flip-flops is given in Table 5.58.

Table 5.58 Excitation table of 3-bit UP/DOWN counter for J-K flip-flops

Control Input	*Present state*			*Next state*			*Flip-flops input*					
$UP/\overline{DOWN} = M$	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
1	0	0	0	0	0	1	1	X	0	X	0	X
1	0	0	1	0	1	0	X	1	1	X	0	X
1	0	1	0	0	1	1	1	X	X	0	0	X
1	0	1	1	1	0	0	X	1	X	1	1	X
1	1	0	0	1	0	1	1	X	0	X	X	0
1	1	0	1	1	1	0	X	1	1	X	X	0
1	1	1	0	1	1	1	1	X	X	0	X	0
1	1	1	1	0	0	0	X	1	X	1	X	1
0	0	0	0	1	1	1	1	X	1	X	1	X
0	1	1	1	1	1	0	X	1	X	0	X	0
0	1	1	0	1	0	1	1	X	X	1	X	0
0	1	0	1	1	0	0	X	1	0	X	X	0
0	1	0	0	0	1	1	1	X	1	X	X	1
0	0	1	1	0	1	0	X	1	X	0	0	X
0	0	1	0	0	0	1	1	X	X	1	0	X
0	0	0	1	0	0	0	X	1	0	X	0	X

K-map for J_0

Q_1Q_0 \ MQ_2	00	01	11	10
00	1	1	1	1
01	×	×	×	×
11	×	×	×	×
10	1	1	1	1

$J_0 = 1$

K-map for K_0

Q_1Q_0 \ MQ_2	00	01	11	10
00	×	×	×	×
01	1	1	1	1
11	1	1	1	1
10	×	×	×	×

$K_0 = 1$

K-map for J_1

Q_1Q_0 \ MQ_2	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	×	×	×	×
10	×	×	×	×

$J_1 = \overline{M}Q_0 + M\overline{Q}_0 = M \oplus Q_0$

K-map for K_1

Q_1Q_0 \ MQ_2	00	01	11	10
00	×	×	×	×
01	×	×	×	×
11	1	1	0	0
10	0	0	1	1

$K_1 = \overline{M}Q_0 + M\overline{Q}_0 = M \oplus Q_0$

K-map for J_2

Q_1Q_0 \ MQ_2	00	01	11	10
00	0	×	×	1
01	0	×	×	0
11	1	×	×	0
10	0	×	×	0

$J_2 = M\overline{Q}_1\overline{Q}_0 + \overline{M}Q_1Q_0$

K-map for K_2

Q_1Q_0 \ MQ_2	00	01	11	10
00	×	0	1	×
01	×	0	0	×
11	×	1	0	×
10	×	0	0	×

$K_2 = M\overline{Q}_1\overline{Q}_0 + \overline{M}Q_1Q_0$

The logic diagram of a 3-bit UP/DOWN counter using J-K flip-flops is shown in Fig. 5.95.

5.18.1 Commonly used ICs for Synchronous Counter

The commonly used TTL ICs of synchronous counter are given below. On the basis of their features, ICs are classified into four groups: group A, group B, group C, and group D.

Group A:

74160 Decade UP counter
74161 4-bit binary UP counter
74162 Decade UP counter

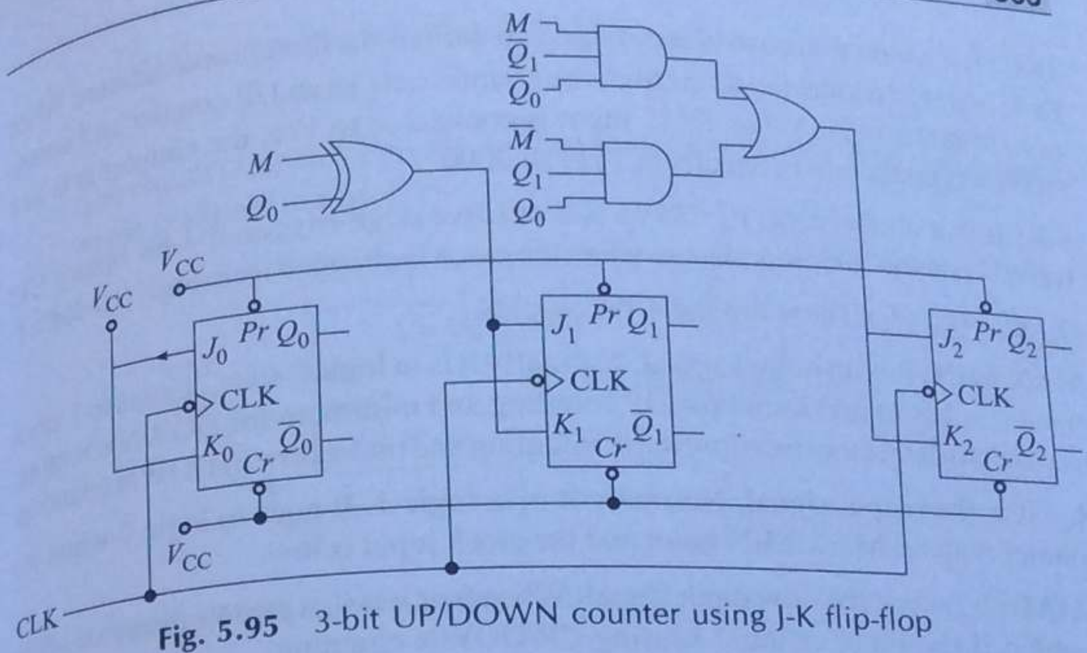


Fig. 5.95 3-bit UP/DOWN counter using J-K flip-flop

Group B:

74168 Decade UP/DOWN counter
74169 4-bit binary UP/DOWN counter

Group C:

74190 Decade UP/DOWN counter
74191 4-bit binary UP/DOWN counter

Group D:

74192 Decade UP/DOWN counter
74193 4-bit binary UP/DOWN counter

5.18.2 IC 74191

IC 74191 is a 4-bit binary UP/DOWN synchronous counter. It has an asynchronous preset input but no clear input. The functional diagram of IC 74191 is shown in Fig. 5.96.

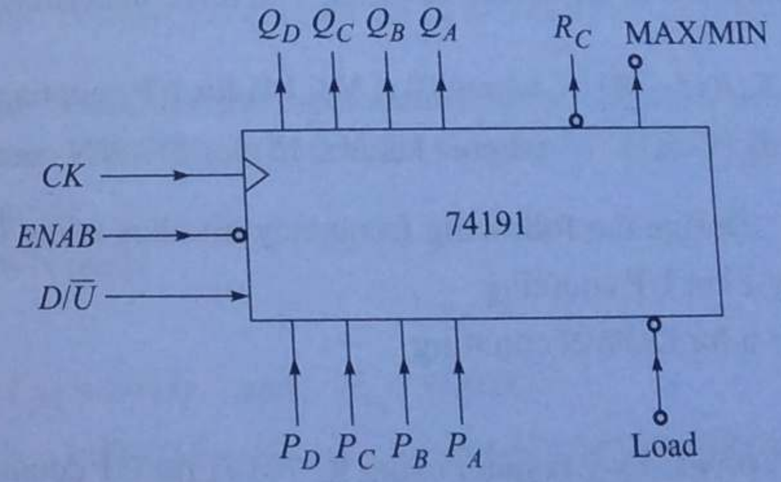


Fig. 5.96 Functional diagram of IC 74191

ENAB: It is an active low enable input signal. The chip gets enabled when this input is connected to ground.

D/$\overline{U}$: It is a direction control input signal. It defines the direction of counter. When D/$\overline{U}$ input is connected to ground, the counter acts as an UP counter and counts from 0000 to 1111. When D/$\overline{U}$ input is connected to V_{CC}, the counter acts as a DOWN counter and counts from 1111 to 0000.

CK: It is a clock input. IC 74191 is a positive edge-triggered 4-bit binary UP/DOWN counter. Outputs change when the clock is changed from logic 0 to logic 1.

Q_D, Q_C, Q_B, Q_A: These are the 4-bit outputs.

MAX/MIN: It is an output signal. Normally it is in logic 0 state. It goes to 1 when count reaches to maximum for UP counting and minimum for DOWN counting. It is used to detect the overflow in UP counting and underflow in DOWN counting.

R_C: It is the output signal. Normally it is in logic 1. It goes to logic 0 when the counter reaches MAX/MIN point and the clock input is low.

LOAD: It is an active low input signal. Whenever it is 0, it presets the mode of the counter. It should be in logic 1during UP/DOWN counting.

P_D,P_C,P_B,P_A: These are the four asynchronous preset inputs. These inputs are used to preset the counter 74191.

The functional table of IC 74191 is given in Table 5.59.

Table 5.59 Functional table of IC 74191

Clock	*Enable*	*Load*	*D/$\overline{U}$*	*Mode*
X	1	X	X	Stop count
X	0	0	X	Preset
↑	0	1	0	Up count
↑	0	1	1	Down count

Applications of IC 74191

IC 74191 can be used as a programmable frequency divider. In this application, the R_C output is connected to the load input and the required output waveform is obtained at the MAX/MIN output. The frequency of the output signal is a function of the input connected to the preset terminal. The relation between F_{out}, F_{in}, and N is

$F_{out} = F_{in}/(15 - N)$ where $(0 \le N \le 14)$ for UP counting

$F_{out} = F_{in}/N$ where $(1 \le N \le 15)$ for DOWN counting

Example 5.17 Design the following frequency dividers using IC 74191

(a) Divide-by-2 for UP counting

(b) Divide-by-5 for DOWN counting

Solution

(a) To design a divide-by-2 counter using IC 74191 for UP counting, calculate the value of N using the equation

$F_{out} = F_{in}/(15 - N)$ where $(0 \le N \le 14)$ for UP counting

$F_{in}/2 = F_{in}/(15 - N)$

$\therefore \quad N = 13\ (1101)$

The circuit diagram of divide-by-2 counter using IC 74191 for UP counting mode is shown in Fig. 5.97.

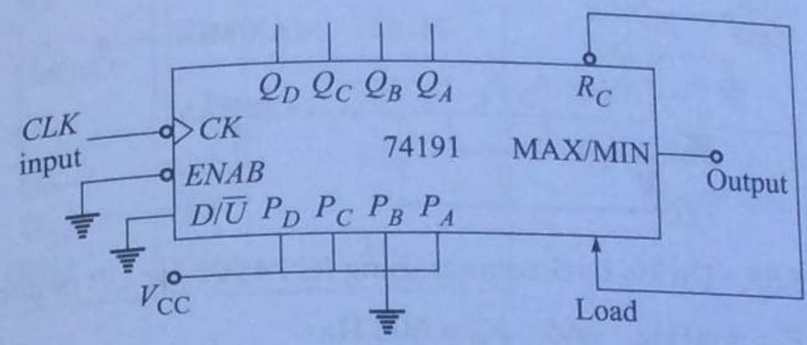


Fig. 5.97 Divide-by-2 counter using IC 74191 for UP counting

(b) To design a divide-by-5 counter using IC 74191 for DOWN counting, calculate the value of N using the equation

$F_{out} = F_{in}/N$ where $(1 \le N \le 15)$ for DOWN counting

$F_{in}/5 = F_{in}/N$

$\therefore \quad N = 5\ (0101)$

The circuit diagram of divide-by-5 counter using IC 74191 for DOWN counting mode is shown in Fig. 5.98.

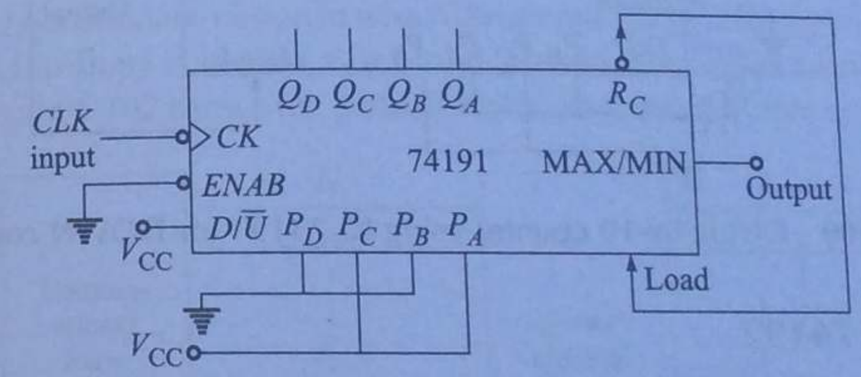


Fig. 5.98 Divide-by-5 counter using IC 74191 for DOWN counting

Example 5.18 Consider the input frequency of the signal is 600 Hz. Design the circuit using 74191 to get the output frequency signal of frequency 60 Hz

(a) in UP mode

(b) in DOWN mode

Solution

(a) Given $F_{out} = 60$ Hz and $F_{in} = 600$ Hz

$F_{out} = F_{in}/(15 - N)$ where $(0 \le N \le 14)$ for UP counting

$60 = 600/(15 - N)$

$\therefore \quad N = 5\ (0101)$

The circuit diagram of divide-by-5 counter using IC 74191 for UP counting mode is shown in Fig. 5.99.

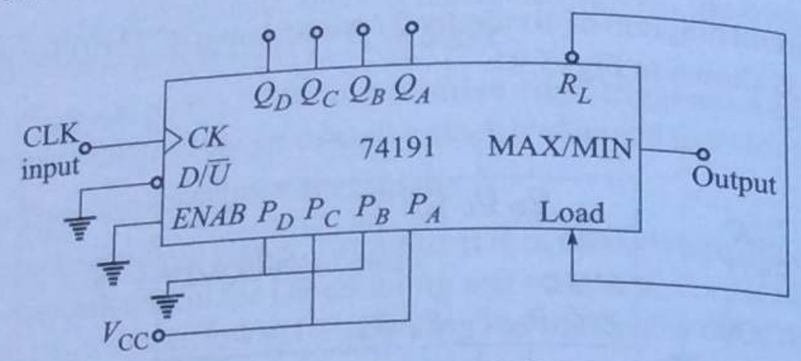


Fig. 5.99 Divide-by-5 counter using IC 74191 for UP counting

(b) Given $F_{out} = 60$ Hz and $F_{in} = 600$ Hz

$F_{out} = F_{in}/N$ where $(1 \le N \le 15)$ for DOWN counting

$60 = 600/N$

$\therefore \quad N = 10\ (1010)$

The circuit diagram of divide-by-10 counter using IC 74191 for DOWN counting mode is shown in Fig. 5.100.

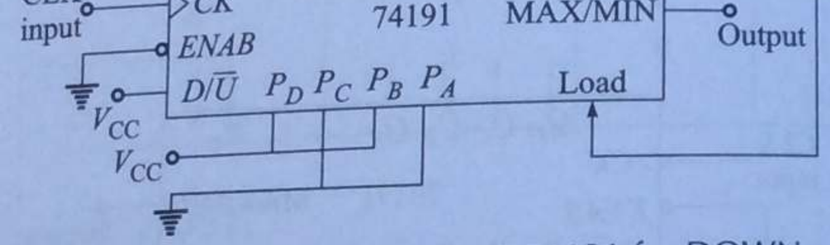


Fig. 5.100 Divide-by-10 counter using IC 74191 for DOWN counting

5.18.3 IC 74192

IC 74192 is an UP/DOWN decade counter with separate UP/DOWN clock. Its supply voltage is 5V, input voltage is 5V, and operating temperature is 0 to 70°C. Maximum count frequency is 30 MHz. It is a 16-pin TTL IC. The functional diagram of IC 74192 is given in Fig. 5.101.

$P_0 - P_3$: Parallel inputs

$Q_0 - Q_3$: Flip-flop output

CP_D: Count down clock input

CP_U: Count up clock input

$\overline{TC_D}$: Terminal count down

$\overline{TC_U}$: Terminal count up

MR: Asynchronous master reset input

$\overline{PL}$: Asynchronous parallel input

Pin	Description	Pin	Description
1	P_1	9	P_3
2	Q_1	10	P_2
3	Q_0	11	$\overline{PL}$
4	CP_D	12	$\overline{TC_U}$
5	CP_U	13	$\overline{TC_D}$
6	Q_2	14	MR
7	Q_3	15	P_0
8	GND	16	V_{CC}

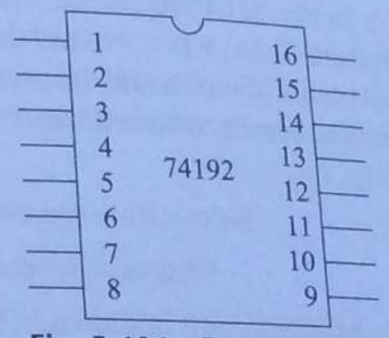


Fig. 5.101 Functional diagram of IC 74192

5.19 CLOCKED SEQUENTIAL CIRCUIT

The clocked sequential circuit is a synchronous circuit. In a clocked synchronous circuit, flip-flops are the basic elements. All the flip-flops used in the circuit are clocked simultaneously. The next state of clocked synchronous circuits depends on present input(s) and the present state of clocked circuits. The output of a clocked synchronous circuit may or may not be a function of its present input, therefore the clocked synchronous circuits are classified into two types: (i) Moore circuit, and (ii) Mealy circuit.

5.19.1 Moore Circuit

The clocked synchronous circuit in which the output Y depends only on the present state of the flip-flops is known as a Moore circuit. It is independent of present input(s). Figure 5.102 shows the general block diagram of Moore circuit.

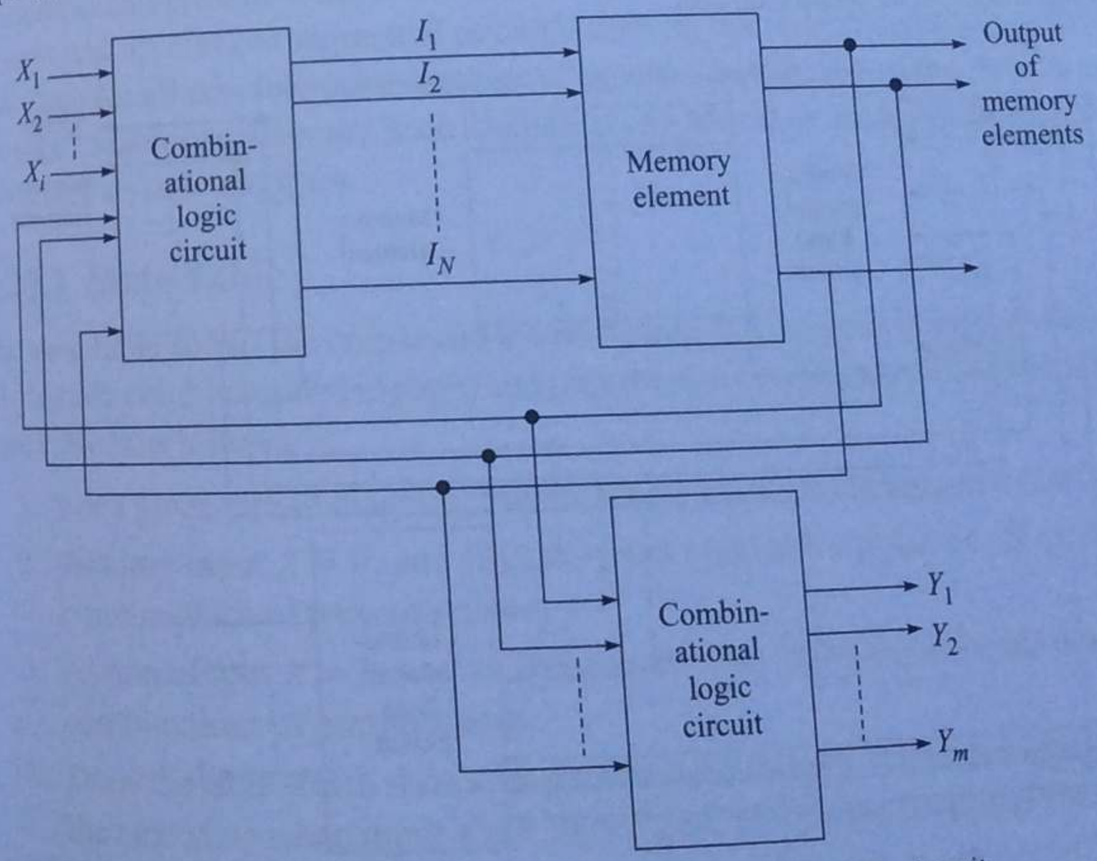


Fig. 5.102 General block diagram of Moore circuit

In Fig. 5.102, $X_1, X_2, ..., X_i$ are the inputs. $I_1, I_2, ..., I_n$ are the outputs of the next state decoder, which control the next state of memory elements. $I_1, I_2, ..., I_n$ are the functions of inputs and the present output state of memory elements. $Y_1, Y_2, ..., Y_m$ are the outputs, which are the functions of present output state of memory elements only, i.e.

Next state = f(present state, inputs)

Output = f(present state)

The Moore circuit is shown in Fig. 5.103.

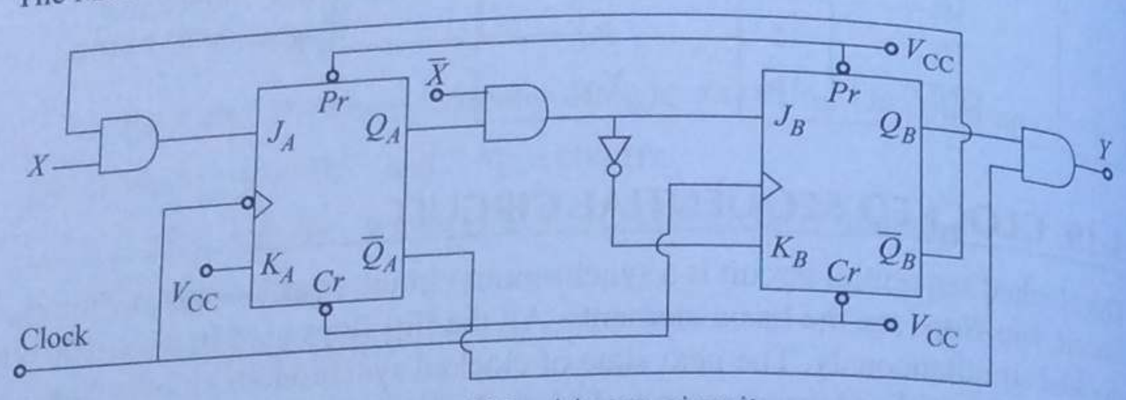


Fig. 5.103 Moore circuit

The clock is given simultaneously to both the flip-flops. Therefore, the circuit is said to be a clocked circuit. The output $Y = Q_B \cdot \overline{Q}_A$. It is a function of the present state only and it is independent of input X.

5.19.2 Mealy Circuit

The clocked sequential circuit in which the output depends on the present state of flip-flop(s) and input(s) is known as a Mealy circuit. Figure 5.104 shows the general block diagram of Mealy circuit.

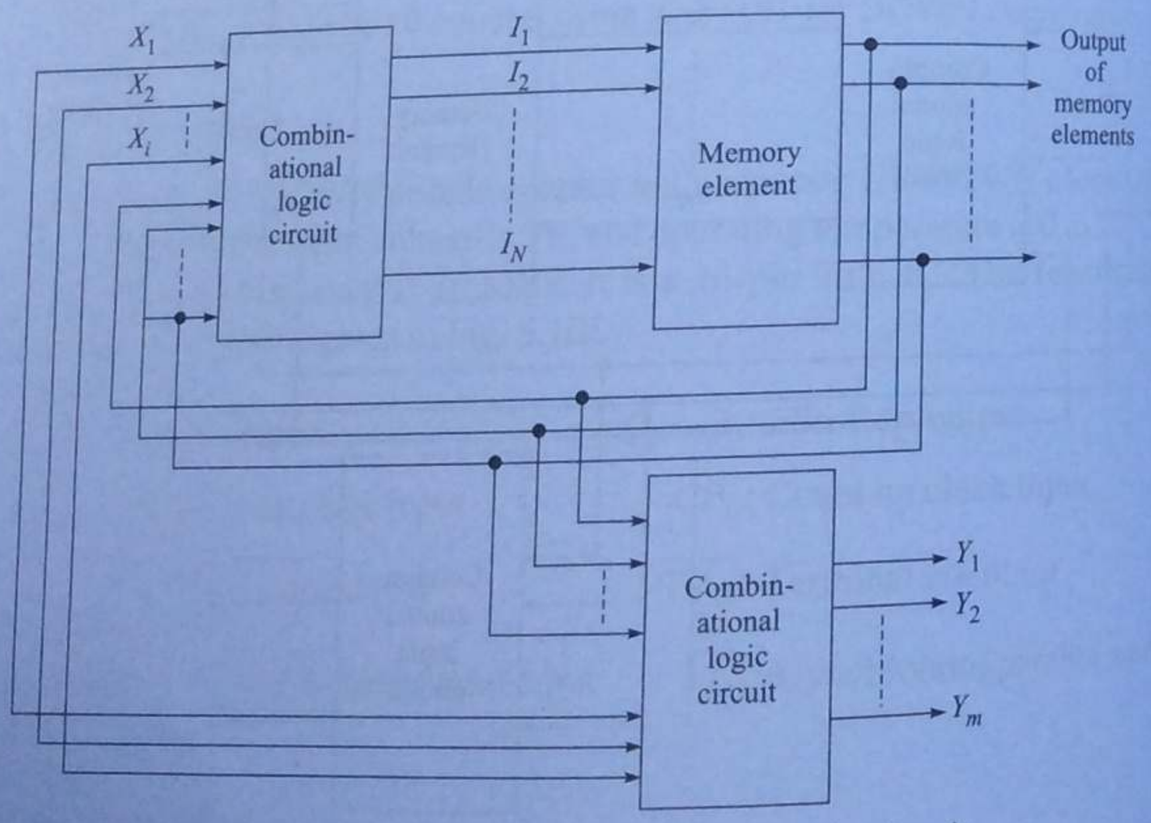


Fig. 5.104 General block diagram of Mealy circuit

In Fig. 5.104, $X_1, X_2, ..., X_i$ are the inputs. $I_1, I_2, ..., I_n$ are the outputs of the next state decoder, which control the next state of memory elements. $I_1, I_2, ..., I_n$ are the functions of inputs and the present output state of memory elements. $Y_1, Y_2, ..., Y_m$ are the outputs; these are the functions of present output state of memory elements and inputs, i.e.

Next state = f(present state, inputs)

Output = f(present state, inputs)

The Mealy circuit is shown in Fig. 5.105.

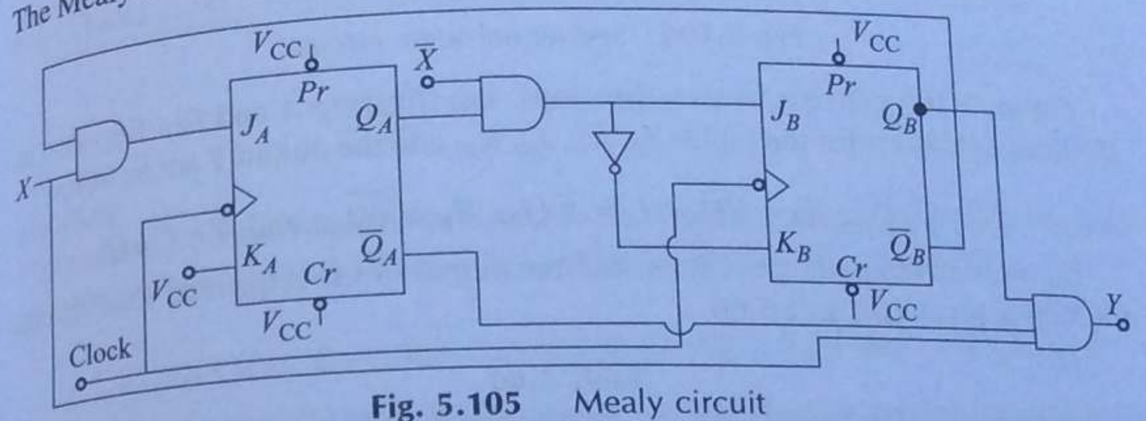


Fig. 5.105 Mealy circuit

The clock is given simultaneously to both the flip-flops. Therefore, the circuit is said to be a clocked circuit. The output $Y = Q_B \cdot \overline{Q}_A$. It is a function of the present state of the circuit and input.

5.20 ANALYSIS OF CLOCKED SEQUENTIAL CIRCUIT

In a sequential circuit, the output and the next state of flip-flops are the functions of input(s) and present state; it depends on the type of circuit, Moore or Mealy. The analysis of clocked sequential circuit is nothing but to find the output and the next state for all possible combinations of input(s) and present states, which are shown by a table or a diagram. Such a table is referred as *state table* and the diagram is referred as *state diagram*.

5.20.1 State Table

The state table shows the output and the next state of a sequential logic circuit for all possible combination of input(s) and present state(s). The procedure to find the state table is as follows:

1. For a given circuit diagram, find the expression for the inputs of flip-flops.
2. Assume input $X = 0$, and find the next state and output for all possible combinations of present states.
3. Assume input $X = 1$, and find the next state and output for all possible combinations of present states.
4. Draw the table which shows the present state, the next state when input $X = 0$, the next state when input $X = 1$, the output when input $X = 0$, and the output when $X = 1$.

Consider the sequential logic circuit as shown in Fig. 5.106.

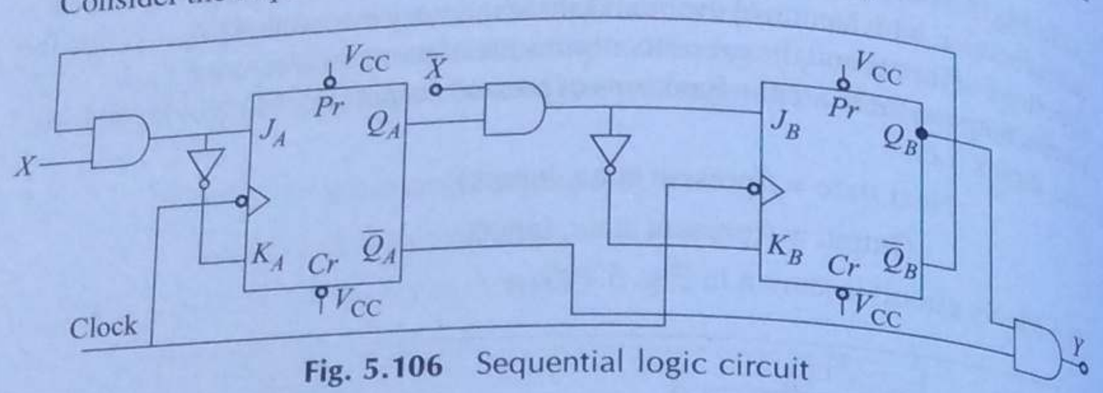


Fig. 5.106 Sequential logic circuit

Figure 5.106 consists of two flip-flops, i.e. flip-flop *A* and flip-flop *B*. The Boolean equations for the inputs J_A, K_A, J_B, K_B, and the output Y are as follows:

$$J_A = X \cdot \overline{Q}_B,\ K_A = \overline{X\overline{Q}_B},\ J_B = \overline{X}\,Q_A,\ K_B = \overline{\overline{X}Q_A},\ \text{and}\ Y = Q_B\,\overline{Q}_A$$

The present state, the next state, and the output of the sequential logic circuit for X=0 is given in Table 5.60.

Table 5.60

Present state						*Next state output*		
Q_A	Q_B	J_A	K_A	J_B	K_B	Q_A	Q_B	Y
0	0	0	1	0	1	0	0	0
0	1	0	1	0	1	0	0	0
1	0	0	1	1	0	0	1	1
1	1	0	1	1	0	0	1	1

The present state, the next state, and the output of the sequential logic circuit for X=1 is given in Table 5.61.

Table 5.61

Present state						*Next state output*		
Q_A	Q_B	J_A	K_A	J_B	K_B	Q_A	Q_B	Y
0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	0
1	0	1	0	0	1	1	0	0
1	1	0	1	0	1	0	0	0

The state table for the given sequential circuit is given in Table 5.62.

Table 5.62

Present state		*Next state*				*Output*	
		X=0		*X=1*		*X=0*	*X=1*
Q_A	Q_B	Q_A	Q_B	Q_A	Q_B	Y	Y
0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0
1	0	0	1	1	0	1	0
1	1	0	1	0	0	1	0

Example 5.19 Draw the state table for the following circuit.

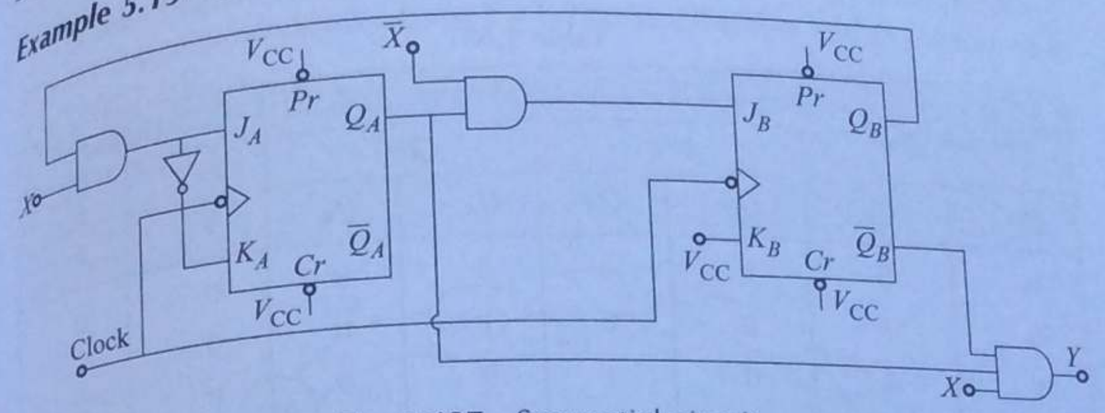


Fig. 5.107 Sequential circuit

Solution

Figure 5.107 consists of two flip-flops, i.e. flip-flop *A* and flip-flop *B*. The Boolean equation for the inputs J_A, K_A, J_B, K_B, and the output Y are as follows:

$$J_A = X \cdot Q_B,\ K_A = \overline{XQ_B},\ J_B = \overline{X}\,Q_A,\ K_B = V_{CC},\ \text{and}\ Y = Q_A\,\overline{Q}_B X$$

The present state, the next state and the output of the sequential logic circuit for X=0 is given in Table 5.63.

Table 5.63

Present state						*Next state output*		
Q_A	Q_B	J_A	K_A	J_B	K_B	Q_A	Q_B	Y
0	0	0	1	0	1	0	0	0
0	1	0	1	0	1	0	0	0
1	0	0	1	1	1	0	1	0
1	1	0	1	1	1	0	0	0

The present state, the next state, and the output of the sequential logic circuit for X=1 is given in Table 5.64.

Table 5.64

Present state						*Next state output*		
Q_A	Q_B	J_A	K_A	J_B	K_B	Q_A	Q_B	Y
0	0	0	1	0	1	0	0	0
0	1	1	0	0	1	1	0	1
1	0	0	1	0	1	0	0	0
1	1	1	0	0	1	1	0	1

The state table for given sequential circuit is given in Table 5.65.

Table 5.65

Present state		*Next state*				*Output*	
		X=0		*X=1*		*X=0*	*X=1*
Q_A	Q_B	Q_A	Q_B	Q_A	Q_B	*Y*	*Y*
0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	1	0	0	1	0	0	1

5.20.2 State Diagram

The state diagram is a graphical way of representing the state table. The state is represented by a circle, with the state indicated inside the circle. Directed lines connecting the circles indicate the transition between the states. The input and output conditions are labeled with two binary numbers separated by the symbol '/' along with directed lines. The first bit indicates the input and the bit after the symbol '/' indicates the output. The state diagram for a 3-bit up counter is shown in Fig. 5.108. When the present state is 000, the next state is 001, and so on. This state diagram shows that the next state depends on the present state only.

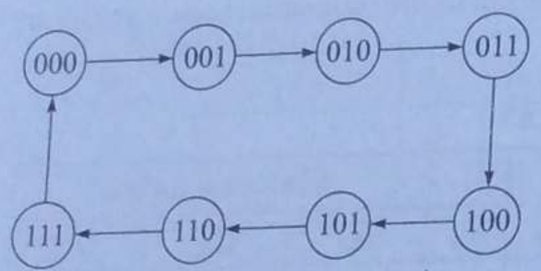


Fig. 5.108 State diagram for 3-bit up counter

The state diagram for a 3-bit up/down is shown in Fig. 5.109. The sequential circuit acts as an up counter, if $X = 0$, otherwise the circuit acts as a down counter. When the present state 000, the next state is 001 for $X = 0$ and it is 111 for $X = 1$.

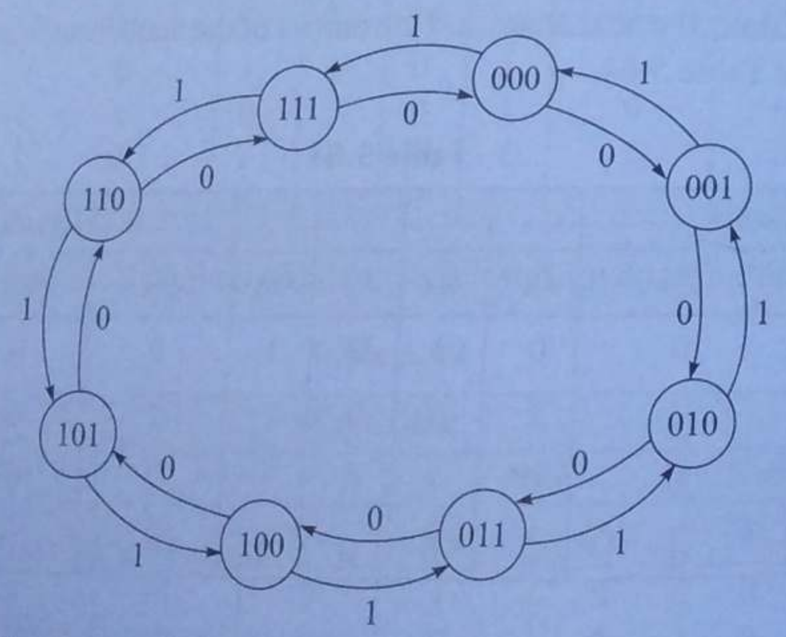


Fig. 5.109 State diagram for 3-bit up/down counter

The state diagram of a 3-bit up/down counter is shown in Fig. 5.110. The circuit acts as an up counter when $X = 0$ and $Y = 1$. The circuit acts as a down counter when $X = 1$ and $Y = 0$.

When $X = 0$, $Y = 1$, the next state of 010 is 011. When $X = 1$, $Y = 0$, the next state of 010 is 001.

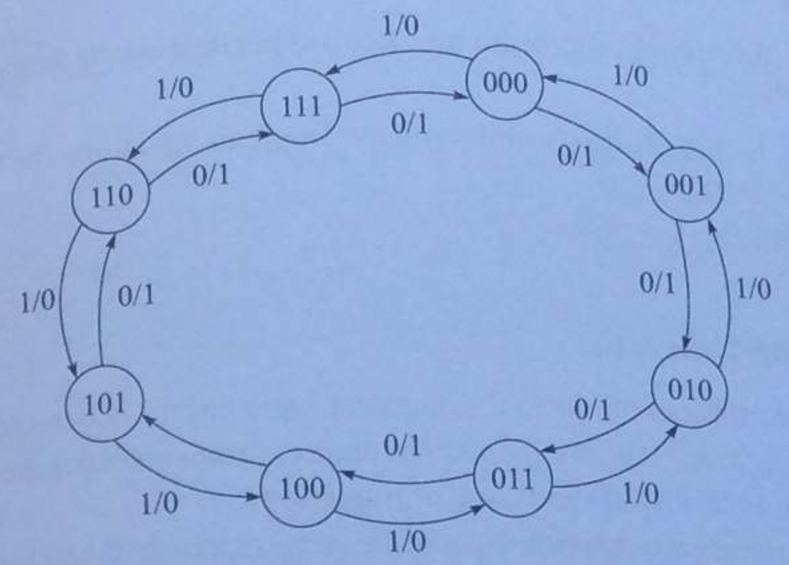


Fig. 5.110 State diagram for 3-bit up/down counter

In case of Moore circuit, the output is not a function of input, therefore the directed lines are located with only one binary number, which represents the state of the input. The output state is indicated within the circle below the present state, as shown in Fig. 5.111.

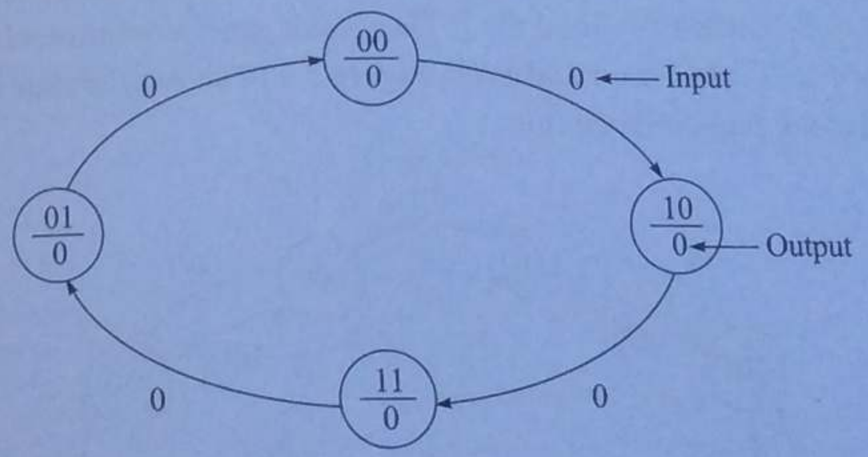


Fig. 5.111 Sequential Moore circuit

5.21 DESIGN OF CLOCKED SEQUENTIAL CIRCUIT

The clocked sequential circuit is defined by a set of statements or state table or state diagram.

1. When the clocked sequential circuit is defined by a set of statements, then the first step of designing a clocked sequential circuit is to prepare a state table from the set of statements.

2. From the state table, draw the state diagram.
3. Using the state reduction technique, reduce the number of states.
4. If the states are not given in binary, assign binary values to each state in the state table, as explained in Section 5.22.5.
5. Determine the number of flip-flops required to implement the given state table.
6. Draw the excitation table for the circuit as per the state table and selection of flip-flops.
7. Using K-map, simplify and write the Boolean equations for the input and output of flip-flops.
8. Draw the logic diagram.

5.21.1 State Diagram

Generally the problem of clocked sequential logic circuit is defined in terms of state diagram or as a set of statements. The state diagram provides the information of state, input, output value and flow of transition.

The binary numbers or variables inside the circle represent the state. The directed lines are labeled with a single or two binary numbers separated by a symbol '/'. When directed lines are labeled by a single value, the single value shows the input value that causes the state transition. When directed lines are labeled with two binary numbers, the value before the symbol '/' is the input value that causes the state transition and the value after the symbol '/' is the output value.

Consider the design of a clocked sequential circuit which generates the sequence 0, 2, 4, 6, 1, 3, 5, 7, when $X=0$ and $Y=1$. The circuit generates the reverse sequence 7, 5, 3, 1, 6, 4, 2, 0, when $X=1$ and $Y=0$. Figure 5.112 shows the state diagram for the desired clock sequential circuit.

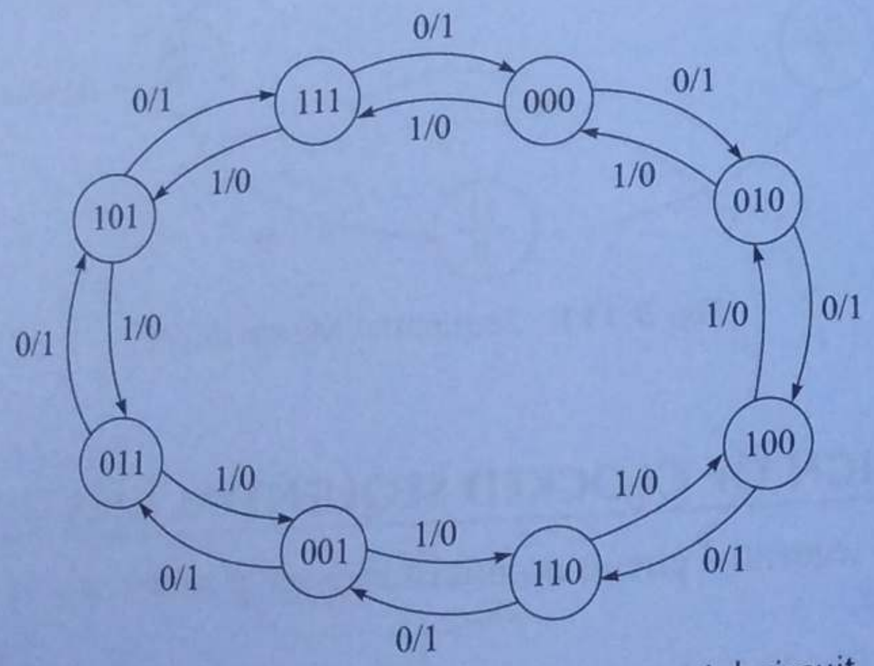


Fig. 5.112 State diagram for clock sequential circuit

5.21.2 State Table

A state table shows the output and the next state of a sequential logic circuit for all possible combinations of present state and input(s). While analysing a clocked sequential circuit, the state table is obtained from the logic circuit. The procedure is explained in Section 5.20.1.

While designing a clocked sequential circuit, the state table is obtained from the state diagram.

For example, the state table for the state diagram of Fig. 5.112 is given in Table 5.66.

Table 5.66 State table for state diagram (Fig. 5.112)

Present state			*Next state*						*Output*	
			X=0			*X=1*			*X=0*	*X=1*
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	*Y*	*Y*
0	0	0	0	1	0	1	1	1	1	0
0	0	1	0	1	1	1	1	0	1	0
0	1	0	1	0	0	0	0	0	1	0
0	1	1	1	0	1	0	0	1	1	0
1	0	0	1	1	0	0	1	0	1	0
1	0	1	1	1	1	0	1	1	1	0
1	1	0	0	0	1	1	0	0	1	0
1	1	1	0	0	0	1	0	1	1	0

The state table for the state diagram of Fig. 5.113 is given in Table 5.67.

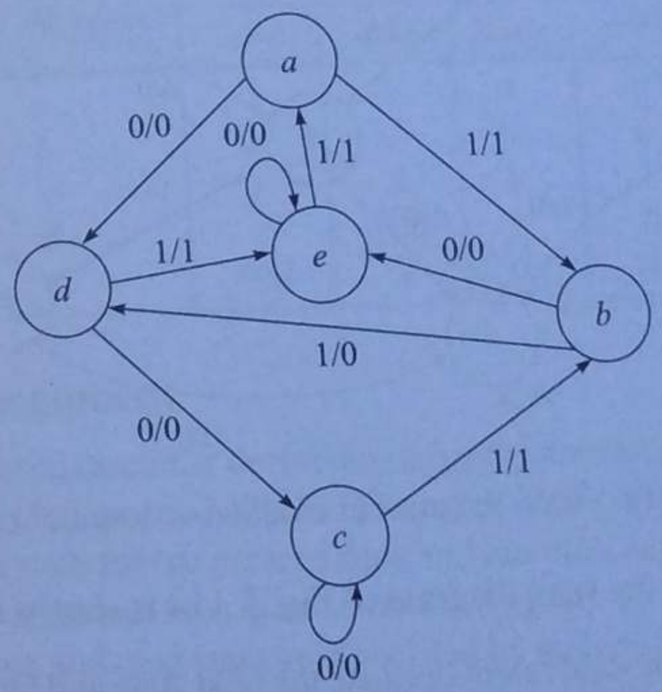


Fig. 5.113 State diagram

Table 5.67 State table for state diagram (Fig. 5.113)

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

5.21.3 State Reduction

The design procedure starts with determining the number of flip-flops required. It depends on the number of states. Using the state reduction techniques, we can reduce the number of states. The state reduction technique avoids the redundancy of states. To reduce the number of states, we have to identify equivalent states. Two states are said to be equivalent, if the output(s) and the next state are same for each input condition. Once the equivalent states are found, one of them may be eliminated without altering the input and output relations.

Consider the state diagram of a clocked sequential circuit, as shown in Fig. 5.114.

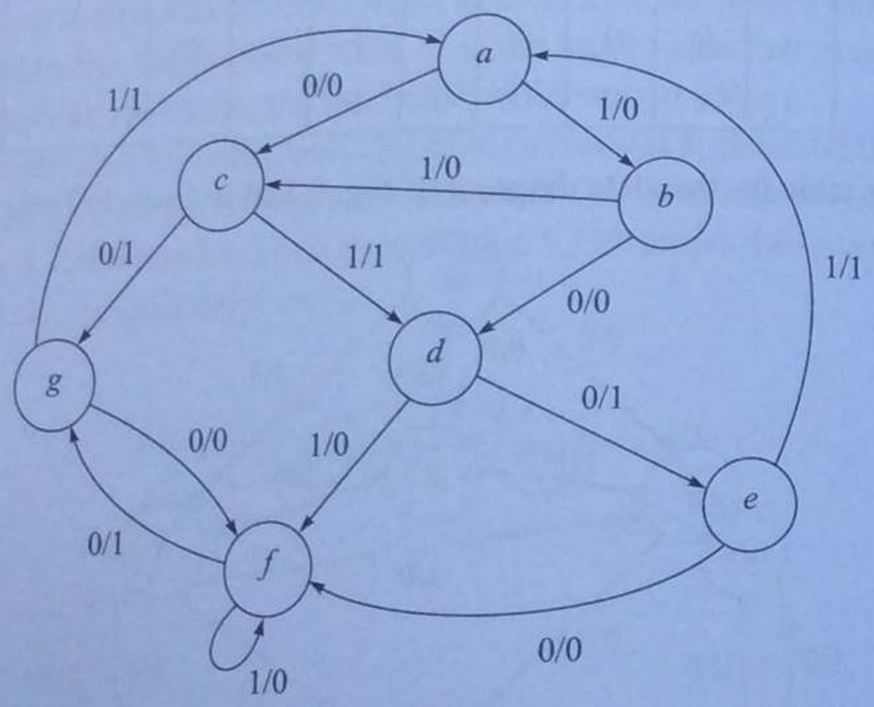


Fig. 5.114 State diagram of clocked sequential circuit

The state table for the state diagram of Fig. 5.114 is shown in Table 5.68.

The states *e* and *g* are equivalent because the next state and the outputs are same for the input $X = 0$ as well as $X = 1$ and hence, *g* is replaced by *e* and the state *g* is removed from the table as shown in Table 5.69.

Table 5.68 State table for state diagram (Fig. 5.115)

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

Table 5.69 Modified 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	(g)e	d	1	1
→d	e	f	1	0
e	f	a	0	1
→f	(g)e	f	1	0

Similarly, the states *d* and *f* are equivalent and hence, *f* is replaced by *d* and the state *f* is removed from the table as shown in Table 5.70.

Table 5.70 State table with *f* replaced by *d*

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)d	1	0
e	(f)d	a	0	1

5.21.4 State Assignment

The clocked sequential circuit is defined in terms of present state, next state and the output for different input conditions. The input for flip-flops are calculated to get the desired next state for the present state and input(s), hence there is a need to specify the present state and next state in binary format.

If the present state and next state are specified by alphabets, then binary values are assigned for the alphabets. The number of bits used to assign binary values to alphabets depends on the number of alphabets used. If the number of alphabets is *M*, then *n*-bits are required to represent the alphabets in binary, such that $2^n \geq M$. The selection of value of *n* is as minimum as possible.

For example, the state table shown in Table 5.70 includes five alphabets and hence, the number of bits required to represent the alphabets in binary is three. The assigned values for the alphabets using 3-bits are given below.

Variable	*Assigned value*
a	000
b	001
c	010
d	011
e	100

The values assigned to alphabets may be any combination for a desired length. Only condition is that a value should not be assigned to more than one variable.

Example 5.20 Design a sequential circuit using J-K flip-flops for the state diagram shown in Fig. 5.115.

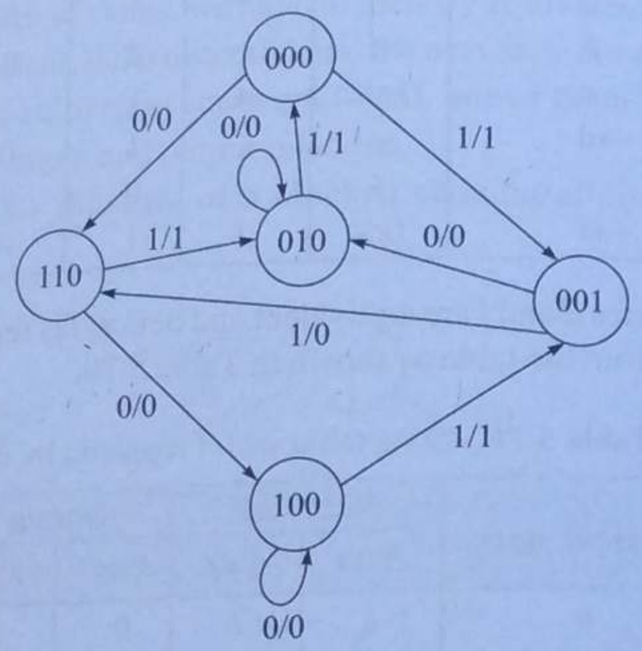


Fig. 5.115 State diagram of sequential circuit

Solution

The state table is given in Table 5.71.

Table 5.71 State table for state diagram (Fig. 5.115)

Present state			*Next state*						*Output*	
			X=0			*X=1*			*X=0*	*X=1*
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	*Y*	*Y*
0	0	0	1	1	0	0	0	1	0	1
0	0	1	0	1	0	1	1	0	0	1
0	1	0	0	1	0	0	0	0	0	1
1	0	0	1	0	0	0	0	1	0	1
1	1	0	1	0	0	0	1	0	0	1

State reduction

The excitation table for the given state diagram is given in Table 5.72.

Table 5.72 Excitation table for the state diagram using J-K flip-flop

Present state			*Input*	*Next state*			*Output*	*Input of the flip-flops*					
Q_2	Q_1	Q_0	*X*	Q_2	Q_1	Q_0	*Y*	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	0	1	1	0	0	0	X	1	X	1	X
0	0	1	0	0	1	0	0	X	1	1	X	0	X
0	1	0	0	0	1	0	0	0	X	X	0	0	X
0	1	0	0	1	0	0	0	0	X	0	X	X	0
1	0	0	0	1	0	0	0	0	X	X	1	X	0
1	1	0	0	0	0	1	1	1	X	0	X	0	X
0	0	1	1	1	1	0	1	X	1	1	X	1	X
0	0	1	1	0	0	0	1	0	X	X	1	0	X
0	1	0	1	0	0	1	1	1	X	0	X	X	1
1	0	0	1	0	1	0	1	0	X	X	0	X	1
1	1	0	1										

K-map for J_0

Q_0X \ Q_2Q_1	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	X	X	X	X
10	X	X	X	X

$J_0 = \overline{Q}_1 X$

K-map for K_0

Q_0X \ Q_2Q_1	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	1	X	X	X
10	1	X	X	X

$K_0 = 1$

K-map for J_1

Q_0X \ Q_2Q_1	00	01	11	10
00	1	X	X	0
01	1	X	X	0
11	1	X	X	X
10	1	X	X	X

$J_1 = \overline{Q}_2\overline{X} + \overline{Q}_2 Q_0$

K-map for K_1

Q_0X \ Q_2Q_1	00	01	11	10
00	X	0	1	X
01	X	1	0	X
11	X	X	X	X
10	X	X	X	X

$K_1 = \overline{Q}_2 X + Q_2\overline{X}$
$= Q_2 \oplus X$

K-map for J_2

Q_0X \ Q_2Q_1	00	01	11	10
00	1	0	X	X
01	0	0	X	X
11	1	X	X	X
10	0	X	X	X

$J_2 = Q_0X + \overline{Q}_1\overline{Q}_0\overline{X}$

K-map for K_2

Q_0X \ Q_2Q_1	00	01	11	10
00	X	X	0	0
01	X	X	1	1
11	X	X	X	X
10	X	X	X	X

$K_2 = X$

K-map for Y

Q_0X \ Q_2Q_1	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	X	X	X
10	0	X	X	X

$Y = X$

The logic diagram for the given state table is shown in Fig. 5.116.

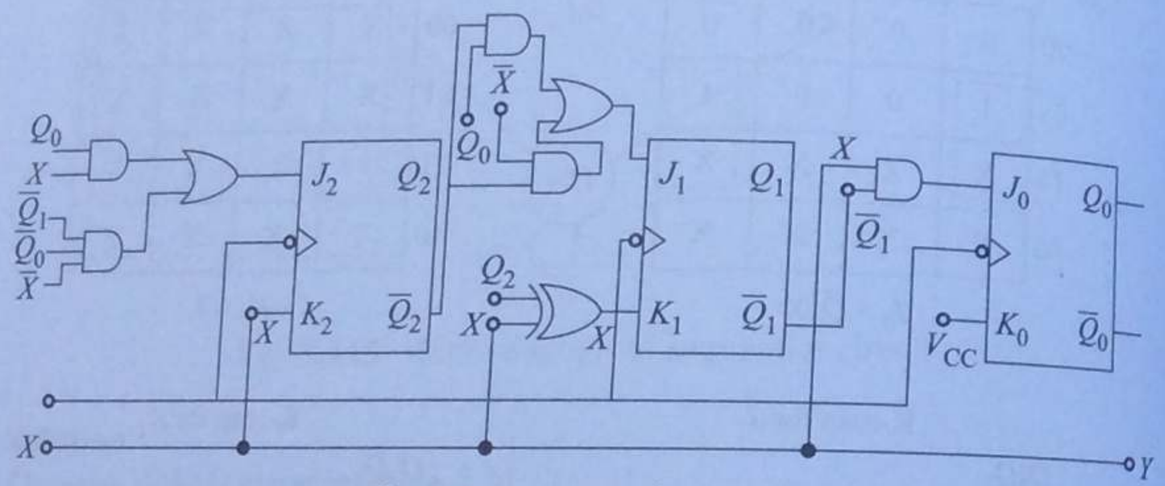


Fig. 5.116 Logic diagram

Example 5.21 With the help of a state table and state diagram, design a Mod-5 up/down counter.

Solution

A Mod-5 counter counts 0-1-2-3-4 during up counting and 4-3-2-1-0 during down counting. Let us consider X as the input signal, which controls the direction of up/down counter. The circuit acts as an up counter for $X = 0$. And for $X = 1$, the circuit acts as a down counter. The state diagram is shown in Fig. 5.117.

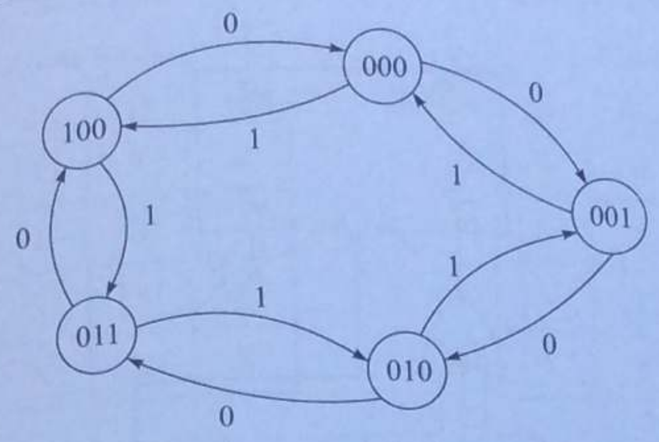


Fig. 5.117 State diagram of Mod-5 up/down counter

The state table which shows the present state and next state for $X = 0$ and $X = 1$ is given in Table 5.73.

Table 5.73 State table of Mod-5 up/down counter

Present state			*Next state*					
			X=0			*X=1*		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1	1	0	0
0	0	1	0	1	0	0	0	0
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	0	1	0
1	0	0	0	0	0	0	1	1

The excitation table for the given state diagram is given in Table 5.74.

Table 5.74 Excitation table for the state table (5.73) using J-K flip-flop

Present state			*Input*	*Next state*			*Input of the flip-flops*					
Q_2	Q_1	Q_0	X	Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	0	0	0	1	1	X	0	X	0	X
0	0	1	0	0	1	0	X	1	1	X	0	X
0	1	0	0	0	1	1	1	X	X	0	0	X
0	1	1	0	1	0	0	X	1	X	1	1	X
1	0	0	0	0	0	0	0	X	0	X	X	1
0	0	0	1	1	0	0	0	X	0	X	1	X
0	0	1	1	0	0	0	X	1	0	X	0	X
0	1	0	1	0	0	1	1	X	X	1	0	X
0	1	1	1	0	1	0	X	1	X	0	0	X
1	0	0	1	0	1	1	1	X	1	X	X	1

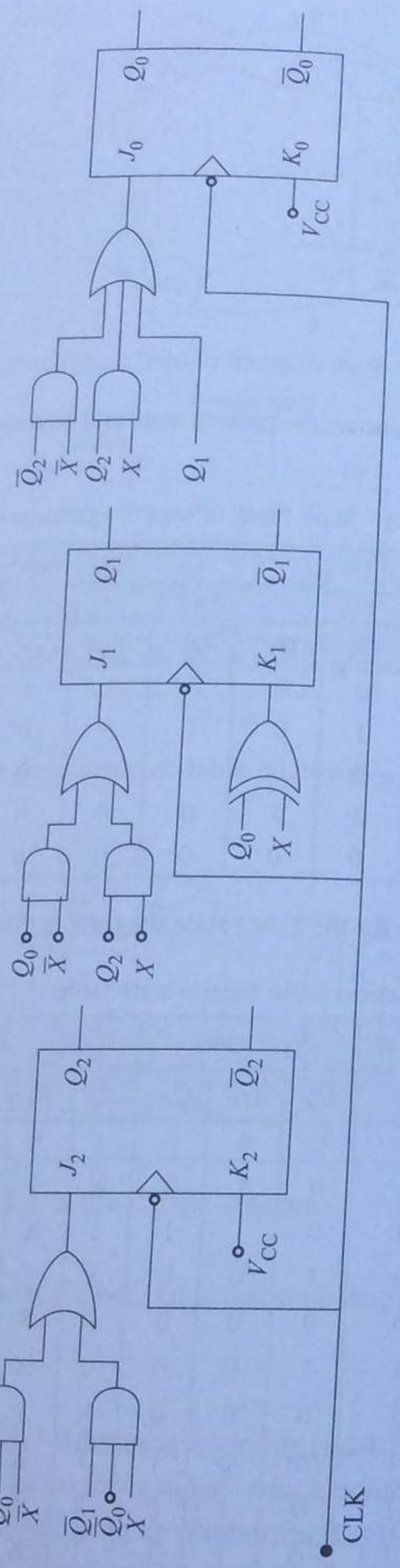


Fig. 5.118 Logic diagram of Mod-5 up/down counter using J-K flip-flop

K-map for J_0

Q_0X \ Q_2Q_1	00	01	11	10
00	1	1	X	0
01	0	1	X	1
11	X	X	X	X
10	X	X	X	X

$J_0 = Q_1 + Q_2X + \overline{Q}_2\overline{X}$

K-map for K_0

Q_0X \ Q_2Q_1	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	1	1	X	X
10	1	1	X	X

$K_0 = 1$

K-map for J_1

Q_0X \ Q_2Q_1	00	01	11	10
00	0	X	X	0
01	0	X	X	1
11	0	X	X	X
10	1	X	X	X

$J_1 = Q_0\overline{X} + Q_2X$

K-map for K_1

Q_0X \ Q_2Q_1	00	01	11	10
00	X	0	X	X
01	X	1	X	X
11	X	0	X	X
10	X	1	X	X

$K_1 = Q_0\overline{X} + \overline{Q}_0X = Q_0 \oplus X$

K-map for J_2

Q_0X \ Q_2Q_1	00	01	11	10
00	0	0	X	X
01	1	0	X	X
11	0	0	X	X
10	0	1	X	X

$J_2 = Q_1Q_0\overline{X} + \overline{Q}_1\overline{Q}_0X$

K-map for K_2

Q_0X \ Q_2Q_1	00	01	11	10
00	X	X	X	1
01	X	X	X	1
11	X	X	X	X
10	X	X	X	X

$K_2 = 1$

The logic diagram for the given state diagram is shown in Fig. 5.118.

Example 5.22 Design a clocked sequential circuit using J-K flip-flop for the state diagram shown in Fig. 5.119.

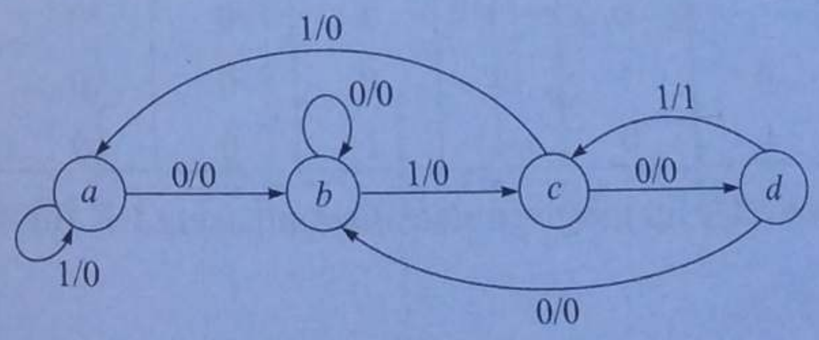


Fig 5.119 State diagram of clocked sequential circuit

Solution

The state table for the state diagram of Fig. 5.119 is shown in Table 5.75.

Table 5.75 State table for the state diagram (Fig 5.119)

Present state	Next state		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	b	a	0	0
b	b	c	0	0
c	d	a	0	0
d	b	c	0	1

State reduction

The states b and d are not equivalent because the outputs are not matching for $X=0$ and $X=1$. From the state table, it is observed that all the states are different and there is no possibility to reduce the state table.

State assignments

The state table includes four alphabets and hence, the number of bits required to represent the alphabets in binary is two. The assigned values for the alphabets using 2-bits are given below.

Variable	Assigned value
a	00
b	01
c	10
d	11

The state table for the state diagram of Fig. 5.119 is shown in Table 5.76.

Table 5.76 State table for the state diagram (Fig 5.119)

Present state		Next state				Output	
		$X=0$		$X=1$		$X=0$	$X=1$
Q_1	Q_0	Q_1	Q_0	Q_1	Q_0	Y	Y
0	0	0	1	0	0	0	0
0	1	0	1	1	0	0	0
1	0	1	1	0	0	0	0
1	1	0	1	1	0	0	1

The excitation table for the given state diagram using J-K flip-flop is given in Table 5.77.

Table 5.77 Excitation table for the state table (5.76) using J-K flip-flop

Present state		Input	Next state		Output	Input of the flip-flops			
Q_1	Q_0	X	Q_1	Q_0	Y	J_0	K_0	J_1	K_1
0	0	0	0	1	0	1	X	0	X
0	1	0	0	1	0	X	0	0	X
0	0	0	1	1	0	1	X	X	0
1	1	0	0	1	0	X	0	X	1
1	0	1	0	0	0	0	X	0	X
0	1	1	1	0	0	X	1	1	X
0	0	1	0	0	0	0	X	X	1
1	1	1	1	0	1	X	1	X	0

K-map for J_0

X \ Q_1Q_0	00	01	11	10
0	1	X	X	1
1	0	X	X	0

$J_0 = \overline{X}$

K-map for K_0

X \ Q_1Q_0	00	01	11	10
0	X	0	0	X
1	X	1	1	X

$K_0 = X$

K-map for J_1

X \ Q_1Q_0	00	01	11	10
0	0	0	X	X
1	0	1	X	X

$J_1 = Q_0X$

K-map for K_1

X \ Q_1Q_0	00	01	11	10
0	X	X	1	0
1	X	X	0	1

$K_1 = Q_0\overline{X} + \overline{Q}_0X = Q_0 \oplus X$

K-map for Y

X \ Q_1Q_0	00	01	11	10
0	0	0	0	0
1	0	0	1	0

$Y = Q_1Q_0X$

The logic diagram for the given state diagram is shown in Fig. 5.120.

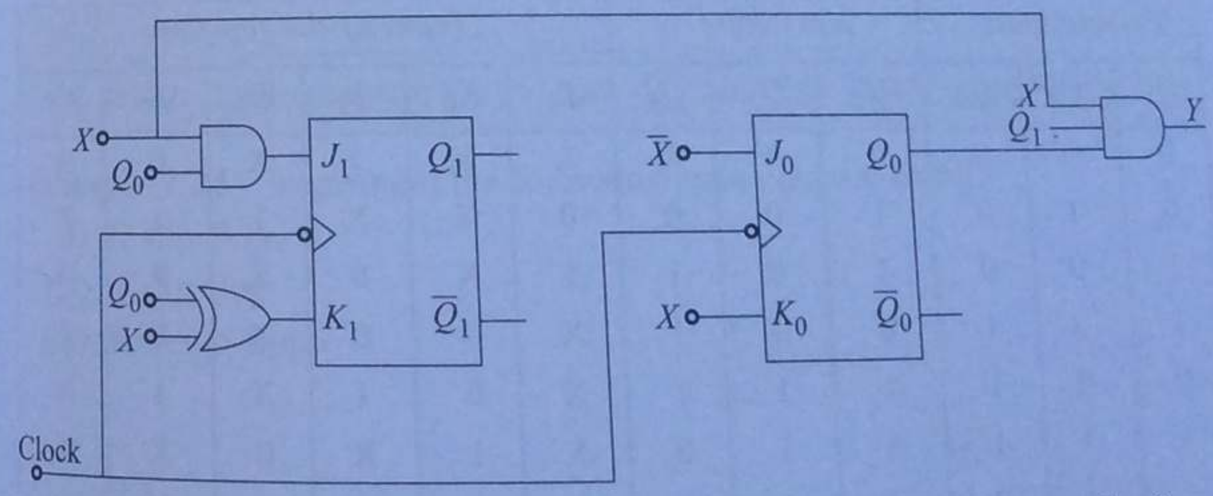


Fig. 5.120 Logic diagram for the state diagram (Fig. 5.119) using J-K flip-flop

Example 5.23 Design a sequential generator using J-K flip-flop to generate the sequence.

$$0 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 7 \rightarrow 6$$

Solution

The state diagram for the given sequence is shown in Fig 5.121.

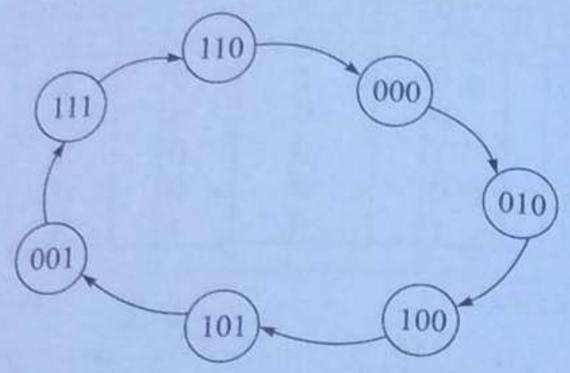


Fig. 5.121 State diagram for the given sequence

The state table is given in Table 5.78.

Table 5.78 State table for the state diagram (Fig 5.121)

Present state			*Next state*		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	1	0
0	1	0	1	0	0
1	0	0	1	0	1
1	0	1	0	0	1
0	0	1	1	1	1
1	1	1	1	1	0
1	1	0	0	0	0

The excitation table for the given state diagram using J-K flip-flop is given in Table 5.79.

Table 5.79 Excitation table for the state table (5.78) using J-K flip-flop

Present state			*Next state*			*Input of the flip-flops*					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	0	1	0	0	X	1	X	0	X
0	1	0	1	0	0	0	X	X	1	1	X
1	0	0	1	0	1	1	X	0	X	X	0
1	0	1	0	0	1	X	0	0	X	X	1
0	0	1	1	1	1	X	0	1	X	1	X
1	1	1	1	1	0	X	1	X	0	X	0
1	1	0	0	0	0	0	X	X	1	X	1

K-map for J_0

Q_2Q_1 / Q_0	00	01	11	10
0	0	0	0	1
1	X	X	X	X

$J_0 = Q_2\overline{Q}_1$

K-map for K_0

Q_2Q_1 / Q_0	00	01	11	10
0	X	X	X	X
1	0	X	1	0

$K_0 = Q_1$

K-map for J_1

Q_2Q_1 / Q_0	00	01	11	10
0	1	X	X	0
1	1	X	X	0

$J_1 = \overline{Q}_2$

K-map for K_1

Q_2Q_1 / Q_0	00	01	11	10
0	X	1	1	X
1	X	X	0	X

$K_1 = \overline{Q}_0$

K-map for J_2

Q_2Q_1 / Q_0	00	01	11	10
0	0	1	X	X
1	1	X	X	X

$J_2 = Q_0 + Q_1$

K-map for K_2

Q_2Q_1 / Q_0	00	01	11	10
0	X	X	1	0
1	X	X	0	1

$K_2 = Q_1\overline{Q}_0 + \overline{Q}_1Q_0$
$= Q_1 \oplus Q_0$

The logic diagram for the given sequence is shown in Fig. 5.122.

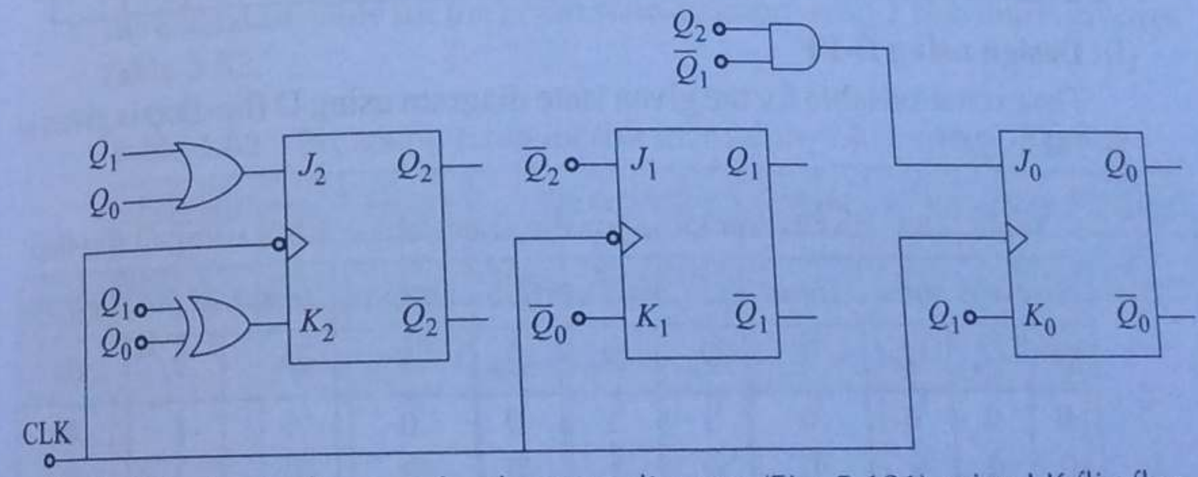


Fig. 5.122 Logic diagram for the state diagram (Fig. 5.121) using J-K flip-flop

Example 5.24 Implement the following state diagram using

(i) D flip-flop
(ii) T flip-flop
(iii) R-S flip-flop

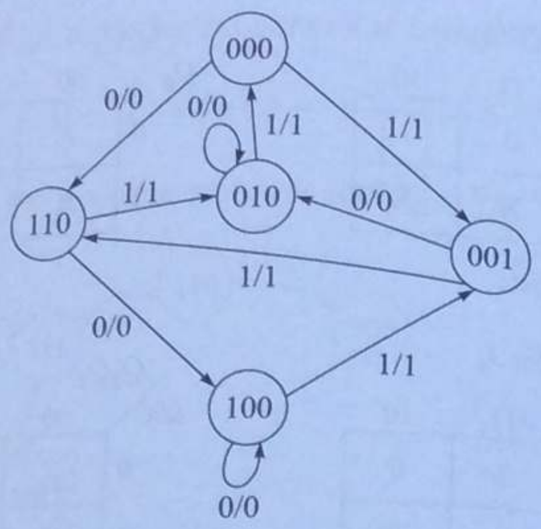


Fig. 5.123 State diagram

Solution

The state table for the state diagram of Fig. 5.123 is given in Table 5.80.

Table 5.80 State table for the state diagram (Fig 5.123)

Present state			*Next state*						*Output*	
			X=0			*X=1*			*X=0*	*X=1*
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	*Y*	*Y*
0	0	0	1	1	0	0	0	1	0	1
0	0	1	0	1	0	1	1	0	0	1
0	1	0	0	1	0	0	0	0	0	1
1	0	0	1	0	0	0	0	1	0	1
1	1	0	1	0	0	0	1	0	0	1

(i) Design using D-FF

The excitation table for the given state diagram using D flip-flop is given in Table 5.81.

Table 5.81 Excitation table for the state table (5.80) using D flip-flop

Present state			*Input*	*Next state*			*Output*	*Input of the flip-flops*		
Q_2	Q_1	Q_0	*X*	Q_2	Q_1	Q_0	*Y*	D_2	D_1	D_0
0	0	0	0	1	1	0	0	1	1	0
0	0	1	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	0	1	0
1	0	0	0	1	0	0	0	1	0	0
1	1	0	0	1	0	0	0	1	0	0
0	0	0	1	0	0	1	1	0	0	1
0	0	1	1	1	1	0	1	1	1	0
0	1	0	1	0	0	0	1	0	0	0
1	0	0	1	0	0	1	1	0	0	1
1	1	0	1	0	1	0	1	0	1	0

K-map for D_2

Q_2Q_1 \ Q_0X	00	01	11	10
00	1	0	1	1
01	0	0	0	0
11	1	X	X	X
10	0	X	X	X

$D_2 = Q_0X + Q_2\overline{X} + \overline{Q}_1\overline{Q}_0\overline{X}$

K-map for D_1

Q_2Q_1 \ Q_0X	00	01	11	10
00	1	1	0	0
01	0	0	1	0
11	1	X	X	X
10	1	X	X	X

$D_1 = \overline{Q}_2\overline{X} + Q_2Q_1X + \overline{Q}_2\overline{Q}_1Q_0$

K-map for D_0

Q_2Q_1 \ Q_0X	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	0	X	X	X
10	0	X	X	X

$D_0 = \overline{Q}_1\overline{Q}_0X$

K-map for Y

Q_2Q_1 \ Q_0X	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	X	X	X
10	0	X	X	X

$Y = X$

The logic diagram for the given state table using D flip-flop is shown in Fig. 5.124.

(ii) Design using T-FF

The excitation table for the given state diagram using T flip-flop is given in Table 5.82.

Table 5.82 Excitation table for the state table (5.81) using T flip-flop

Present state			*Input*	*Next state*			*Output*	*Input of the flip-flops*		
Q_2	Q_1	Q_0	*X*	Q_2	Q_1	Q_0	*Y*	T_0	T_1	T_2
0	0	0	0	1	1	0	0	0	1	1
0	0	1	0	0	1	0	0	1	1	0
0	1	0	0	0	1	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	1	0
0	0	0	1	0	0	1	1	1	0	0
0	0	1	1	1	1	0	1	1	0	1
0	1	0	1	0	0	0	1	0	1	0
1	0	0	1	0	0	1	1	1	0	1
1	1	0	1	0	1	0	1	0	0	1

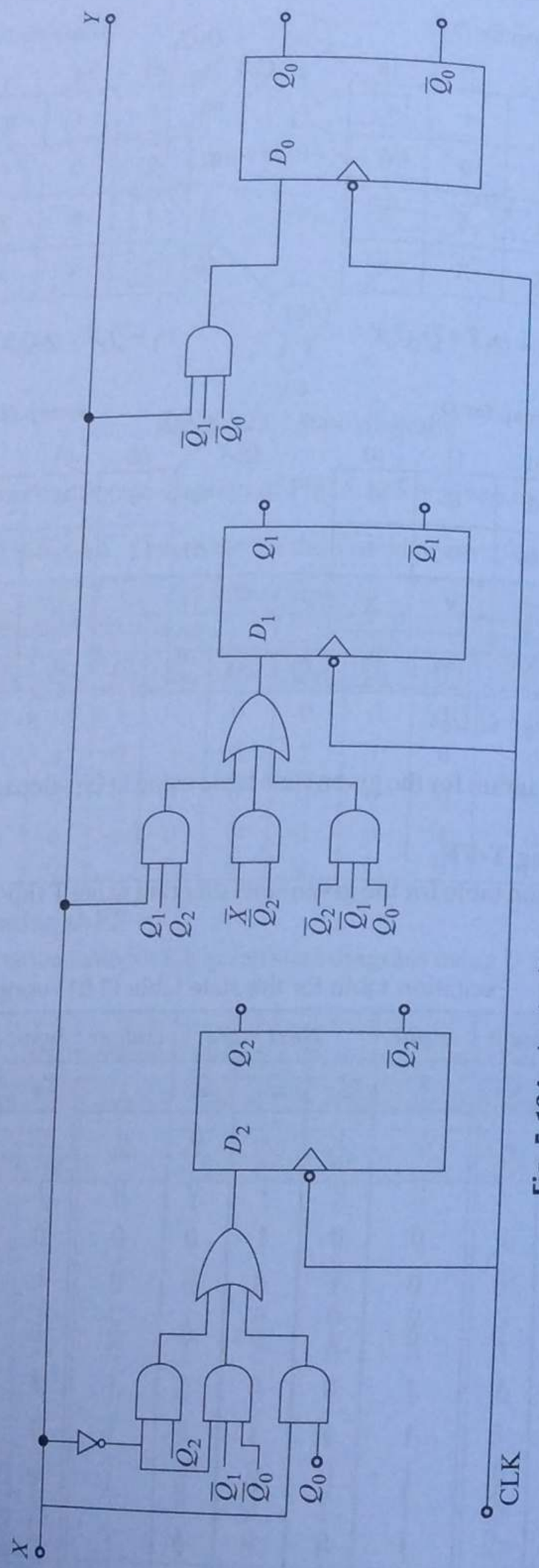


Fig. 5.124 Logic diagram for state diagram (Fig. 5.123) using D flip-flop

K-map for T_0

Q_0X \ Q_2Q_1	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	1	X	X	X
10	1	X	X	X

$T_0 = Q_0\overline{X} + \overline{Q}_1X$

K-map for T_1

Q_0X \ Q_2Q_1	00	01	11	10
00	1	0	1	0
01	0	1	0	0
11	0	X	X	X
10	1	X	X	X

$T_1 = Q_2Q_1\overline{X} + \overline{Q}_2\overline{Q}_1\overline{X} + \overline{Q}_2Q_1Q_0X$

K-map for T_2

Q_0X \ Q_2Q_1	00	01	11	10
00	1	0	0	0
01	0	0	1	1
11	1	X	X	X
10	0	X	X	X

$T_2 = Q_2X + Q_0X + \overline{Q}_2\overline{Q}_1\overline{Q}_0\overline{X}$

K-map for Y

Q_0X \ Q_2Q_1	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	X	X	X
10	0	X	X	X

$Y = X$

The logic diagram for the given state table using T flip-flop is shown in Fig. 5.125.

(iii) Design using S-R FF

The excitation table for the given state diagram using S-R flip-flop is given in Table 5.83.

Table 5.83 Excitation table for the state table (5.81) using S-R flip-flop

Present state			*Input*	*Next state*			*Output*	*Input of the flip-flops*					
Q_2	Q_1	Q_0	X	Q_2	Q_1	Q_0	Y	S_0	R_0	S_1	R_1	S_2	R_2
0	0	0	0	1	1	0	0	0	X	1	0	1	0
0	0	1	0	0	1	0	0	0	1	1	0	0	X
0	1	0	0	0	1	0	0	0	X	X	0	0	X
1	0	0	0	1	0	0	0	0	X	0	X	X	0
1	1	0	0	1	0	0	0	0	X	0	1	X	0
0	0	0	1	0	0	1	1	1	0	0	X	0	X
0	0	1	1	1	1	0	1	0	1	1	0	1	0
0	1	0	1	0	0	0	1	0	X	0	1	0	X
1	0	0	1	0	0	1	1	1	0	0	X	0	1
1	1	0	1	0	1	0	1	0	X	X	0	0	1

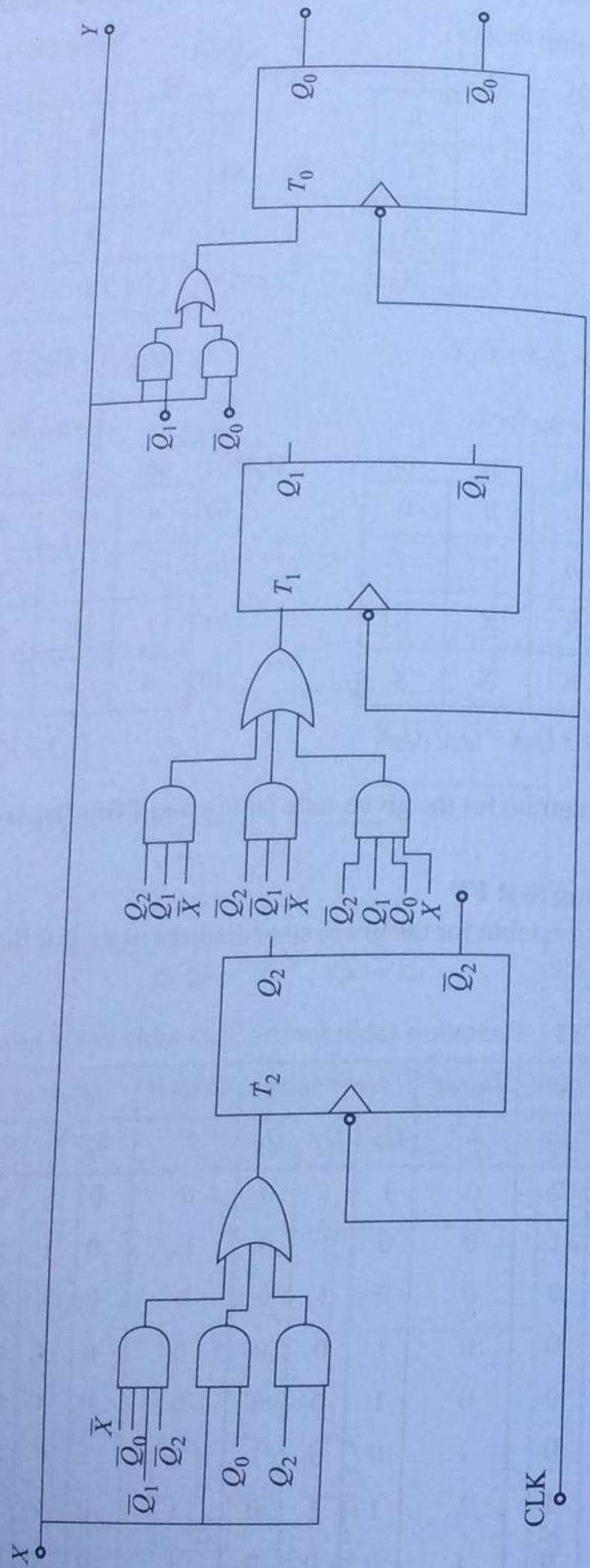


Fig. 5.125 Logic diagram for state diagram (Fig. 5.124) using T flip-flop

K-map for S_0

Q_0X \ Q_2Q_1	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	0	X	X	X
10	0	X	X	X

$S_0 = \overline{Q}_0\overline{Q}_1X$

K-map for R_0

Q_0X \ Q_2Q_1	00	01	11	10
00	X	X	X	X
01	0	X	X	0
11	1	X	X	X
10	1	X	X	X

$R_0 = Q_0$

K-map for S_1

Q_0X \ Q_2Q_1	00	01	11	10
00	1	X	0	0
01	0	0	X	0
11	1	X	X	X
10	1	X	X	X

$S_1 = \overline{Q}_2\overline{X} + \overline{Q}_2Q_0$

K-map for R_1

Q_0X \ Q_2Q_1	00	01	11	10
00	0	0	1	X
01	X	1	0	X
11	0	X	X	X
10	0	X	X	X

$R_1 = Q_2\overline{X} + \overline{Q}_2Q_1X$

K-map for S_2

Q_0X \ Q_2Q_1	00	01	11	10
00	1	0	X	X
01	0	0	0	0
11	1	X	X	X
10	0	X	X	X

$S_2 = Q_0X + \overline{Q}_2\overline{Q}_1\overline{Q}_0\overline{X}$

K-map for R_2

Q_0X \ Q_2Q_1	00	01	11	10
00	0	X	0	0
01	X	X	1	1
11	0	X	X	X
10	X	X	X	X

$K_2 = Q_2X$

K-map for Y

Q_0X \ Q_2Q_1	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	X	X	X
10	0	X	X	X

$Y = X$

The logic diagram for the given state table using S-R flip-flop is shown in Fig. 5.126.

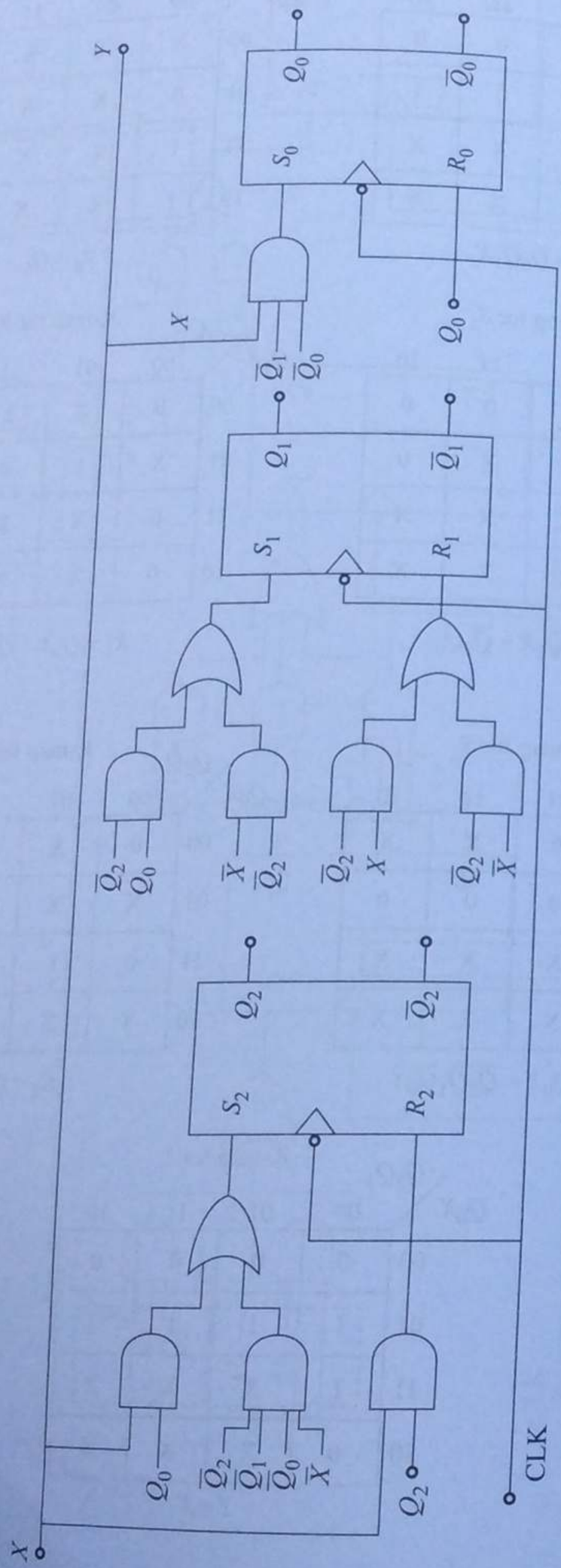


Fig. 5.126 Logic diagram for state diagram (Fig. 5.123) using S-R flip-flop

Example 5.25 Design a circuit to generate the sequence

$$0 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 7$$

Solution

The state diagram for the given sequence is shown in Fig. 5.127.

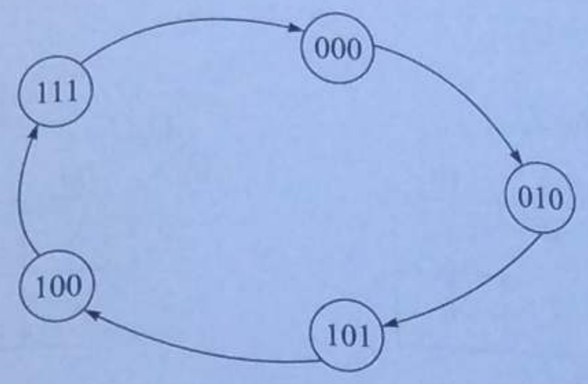


Fig. 5.127 State diagram for the given sequence

The state table for the state diagram of Fig. 5.127 is given in Table 5.84.

Table 5.84 State table for the state diagram (Fig 5.127)

Present state			*Next state*		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	1	0
0	1	0	1	0	1
1	0	1	1	0	0
1	0	0	1	1	1
1	1	1	0	0	0

The excitation table for the given state diagram is given in Table 5.85.

Table 5.85 Excitation table for the state table (5.84) using J-K flip-flop

Present state			*Next state*			*Input of the flip-flops*					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	0	1	0	0	X	1	X	0	X
0	1	0	1	0	1	1	X	X	1	1	X
1	0	1	1	0	0	X	1	0	X	X	0
1	0	0	1	1	1	1	X	1	X	X	0
1	1	1	0	0	0	X	1	X	1	X	1

K-map for J_0

Q_0 \ Q_2Q_1	00	01	11	10
0	0	1	X	1
1	X	X	X	X

$J_0 = Q_1 + Q_2$

K-map for K_0

Q_0 \ Q_2Q_1	00	01	11	10
0	X	X	X	X
1	X	X	1	1

$K_0 = 1$

K-map for J_1

Q_0 \ Q_2Q_1	00	01	11	10
0	1	X	X	1
1	X	X	X	0

$J_1 = \overline{Q}_0$

K-map for K_1

Q_0 \ Q_2Q_1	00	01	11	10
0	X	1	X	X
1	X	X	1	X

$K_1 = 1$

K-map for J_2

Q_0 \ Q_2Q_1	00	01	11	10
0	0	1	X	X
1	X	X	X	X

$J_2 = Q_1$

K-map for K_2

Q_0 \ Q_2Q_1	00	01	11	10
0	X	X	X	0
1	X	X	1	0

$K_2 = Q_1$

The logic diagram for the given sequence using J-K flip-flop is shown in Fig. 5.128.

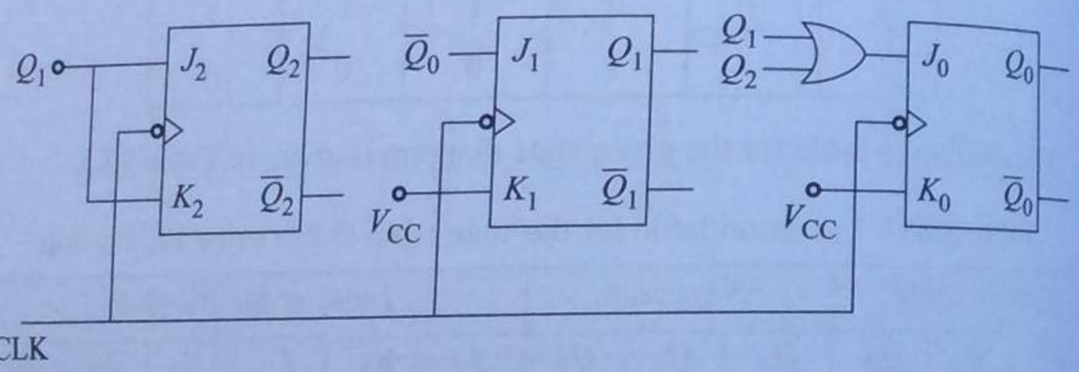


Fig. 5.128 Logic diagram for the state diagram (Fig. 5.127) using J-K flip-flop

5.22 LOCKOUT CONDITION

In Example 5.25, the clocked circuit is designed to generate the sequence 0→2→5→4→7→0, the states 1, 3, and 6 are unused states. If the sequence generator finds itself in an unused state, the next state is unknown. It may be possible that the sequence generator go from one unused state to another unused state, but it never arrives at a used state. The circuit is said to be *locked*. To

avoid the condition of lockout, there is a need to design the circuit such that when the circuit is found in an unused state, the next state should be known and it must be a used state. The state diagrams of the sequence generator to generate the sequence 0→2→5→4→7→0 with lockout condition is shown in Figs. 5.129 and 5.130.

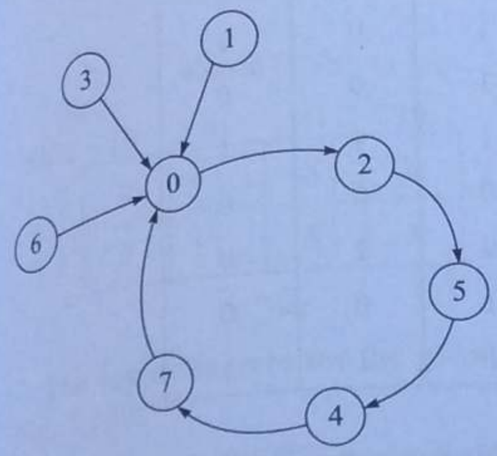


Fig. 5.129 State diagrams to generate the sequence 0→2→5→4→7 →0 with lockout condition

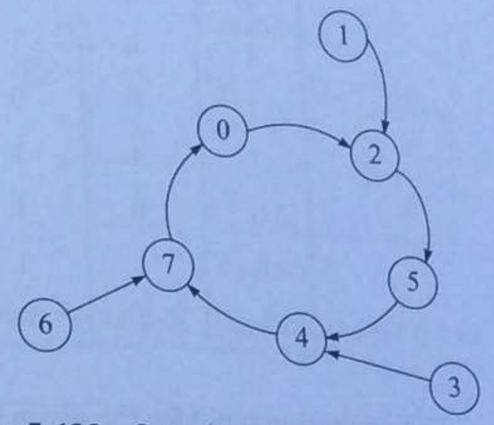


Fig. 5.130 State diagrams to generate the sequence 0→2→5 →4→7→ 0 with lockout condition

In Fig. 5.129, when the sequence generator is found in an unused state, its next state is a used state 0. When the sequence generator is found in an unused state, the next state is the nearest used state in Fig. 5.130. For example, for the unused state 1, the next state is 2; and for the unused state 3, the next state is 4.

***Example* 5.26** Design a sequence generator to generate the sequence 0→2→5→4→7 and avoid the lockout condition using J-K flip-flops.

Solution

The state diagram to generate the sequence 0→2→5→4→7 while avoiding the lockout condition using J-K flip-flops is shown in Fig. 5.131.

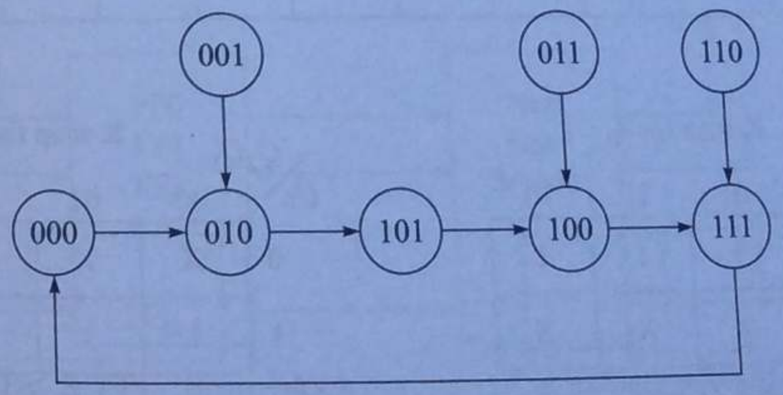


Fig. 5.131 State diagram to generate the sequence 0→2→5→4→7

The state table for the state diagram (Fig. 5.131) is given in Table 5.86.

Table 5.86 State table for the state diagram (Fig. 5.131)

Present state			*Next state*		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	1	0
0	1	0	1	0	1
1	0	1	1	0	0
1	0	0	1	1	1
1	1	1	0	0	0
0	0	1	0	1	0
0	1	1	1	0	0
1	1	0	1	1	1

The excitation table for the given state table is given in Table 5.87.

Table 5.87 Excitation table for the state table (5.86) using J-K flip-flop

Present state			*Next state*			*Input of the flip-flops*					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	0	1	0	0	X	1	X	0	X
0	1	0	1	0	1	1	X	X	1	1	X
1	0	1	1	0	0	X	1	0	X	X	0
1	0	0	1	1	1	1	X	1	X	X	0
1	1	1	0	0	0	X	1	X	1	X	1
0	0	1	0	1	0	X	1	1	X	0	X
0	1	1	1	0	0	X	1	X	1	1	X
1	1	0	1	1	1	1	X	X	0	X	0

K-map for J_0

Q_2Q_1 / Q_0	00	01	11	10
0	0	1	1	1
1	X	X	X	X

$J_0 = Q_1 + Q_2$

K-map for K_0

Q_2Q_1 / Q_0	00	01	11	10
0	X	X	X	X
1	1	1	1	1

$K_0 = 1$

K-map for J_1

Q_2Q_1 / Q_0	00	01	11	10
0	1	X	X	1
1	1	X	X	0

$J_1 = \overline{Q}_0 + \overline{Q}_2$

K-map for K_1

Q_2Q_1 / Q_0	00	01	11	10
0	X	1	0	X
1	X	1	1	X

$K_1 = Q_0 + \overline{Q}_2$

K-map for J_2

Q_2Q_1 / Q_0	00	01	11	10
0	0	1	X	X
1	0	1	X	X

$J_2 = Q_1$

K-map for K_2

Q_2Q_1 / Q_0	00	01	11	10
0	X	X	0	0
1	X	X	1	0

$K_2 = Q_1Q_0$

The logic diagram for the given sequence using J-K flip-flop is shown in Fig. 5.132.

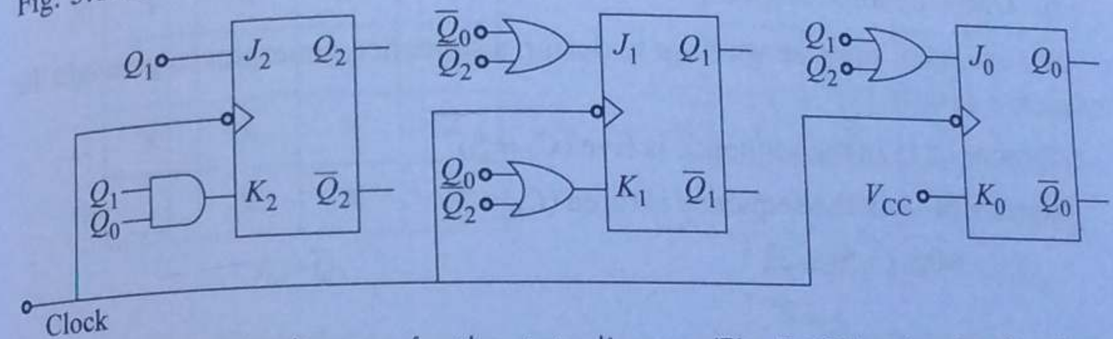


Fig. 5.132 Logic diagram for the state diagram (Fig. 5.131) using J-K flip-flop

5.23 SEQUENCE GENERATOR

The circuit which generates a desired sequence of bits in synchronous mode with a clock is known as a *sequence generator*. The basic block diagram of a sequence generator is shown in Fig. 5.133.

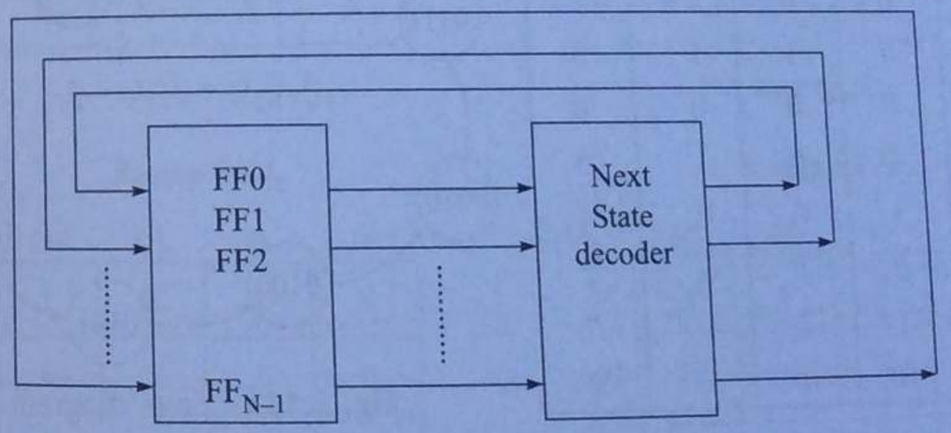


Fig. 5.133 Basic block diagram of sequence generator

For designing a sequence generator, first calculate the number of flip-flops required and then, design the next state decoder to generate the desired sequence.

The procedure to design a sequence generator is as follows:

1. Find the minimum number of flip-flops required.
   The minimum number of flip-flops required depends on the nature of sequence. The required number of flip-flops is calculated from the equation

   $$\text{Max}\ (C_0, C_1) \le 2^{n-1}$$

   where n is the number of flip-flops required, C_0 is the count of 0 in the given sequence, and C_1 is the count of 1 in the given sequence.
2. Draw the state table as per the following procedure.
   (a) Assign the desired sequence (start from LSB) to the output of the flip-flops, which produces the output.
   (b) Assign the output to the other flip-flops 0 or 1, such that all the states are different.
3. Draw the state diagram from the state table.
4. Prepare the excitation table on the basis of present state and next state.
5. Draw the K-map and simplify the same.
6. Draw the logic diagram.

For example, suppose we have to design a sequence generator to generate the sequence 11001011.

Number of 1s in the sequence is five ($C_1 = 5$)

Number of 0s in the sequence is three ($C_0 = 3$)

$$\text{Max}\ (3,5) \le 2^{n-1}$$

$$5 \le 2^{n-1}$$

$$n = 4$$

The state table to generate the desired sequence is given in Table 5.88.

Table 5.88 State table to generate the sequence 11001011

Q_3	Q_2	Q_1	Q_0	State
0	0	0	1	1
0	0	1	1	3
0	0	0	0	0
0	1	0	1	5
0	1	0	0	4
0	0	1	0	2
0	1	1	1	7
1	0	0	1	9

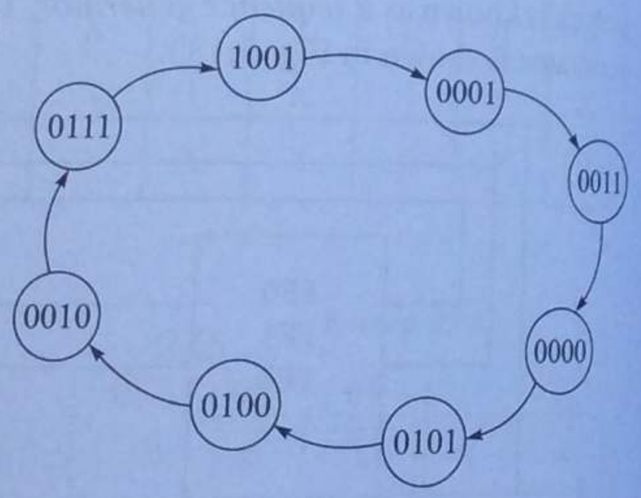


Fig. 5.134 State diagram

The desired sequence (start from LSB) is assigned under the column of Q_0 and the values of Q_3, Q_2, and Q_1 are assigned as 0 or 1 such that all the states get different values.

For the above state table, the state diagram is shown in Fig. 5.134.

The excitation table for the state diagram of Fig. 5.134 is shown in Table 5.89.

Table 5.89 Excitation table for the state diagram (Fig. 5.134) using J-K flip-flop

Present state				Next state				Input of Flip-flops							
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2	J_3	K_3
0	0	0	1	0	0	1	1	X	0	1	X	0	X	0	X
0	0	1	1	0	0	0	0	X	1	X	1	0	X	0	X
0	0	0	0	0	1	0	1	1	X	0	X	1	X	0	X
0	0	0	1	0	1	0	0	X	1	0	X	X	0	0	X
0	1	0	0	0	0	1	0	0	X	1	X	X	1	0	X
0	1	0	0	0	1	1	1	1	X	X	0	1	X	0	X
0	0	1	0	1	0	0	1	X	0	X	1	X	1	0	X
0	1	1	1	0	0	0	0	X	1	0	X	0	X	1	X
1	0	0	1	0	0	0	0	X	1	0	X	0	X	X	1

K-map for J_0

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	1	0	X	X
01	X	X	X	X
11	X	X	X	X
10	1	X	X	X

$J_0 = \overline{Q}_2$

K-map for K_0

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	X	X	X	X
01	0	1	X	1
11	1	0	X	X
10	X	X	X	X

$K_0 = Q_3 + Q_2\overline{Q}_1 + Q_1\overline{Q}_2$

K-map for J_1

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	0	1	X	X
01	1	0	X	0
11	X	X	X	X
10	X	X	X	X

$J_1 = Q_2\overline{Q}_0 + Q_0\overline{Q}_2\overline{Q}_3$

K-map for K_1

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	1	1	X	X
10	0	X	X	X

$K_1 = Q_1Q_0$

K-map for J_2

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	1	X	X	X
01	0	X	X	0
11	0	X	X	X
10	1	X	X	X

$J_2 = \overline{Q}_0$

K-map for K_2

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	X	1	X	X
01	X	0	X	X
11	X	1	X	X
10	X	X	X	X

$K_2 = Q_1 + \overline{Q}_0$

K-map for J_3

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	0	0	X	X
01	0	0	X	X
11	0	1	X	X
10	0	X	X	X

$J_3 = Q_2Q_1$

K-map for K_3

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	X	X	X	X
01	X	X	X	1
11	X	X	X	X
10	X	X	X	X

$K_3 = 1$

The logic diagram to generate the given sequence is shown in Fig. 5.135.

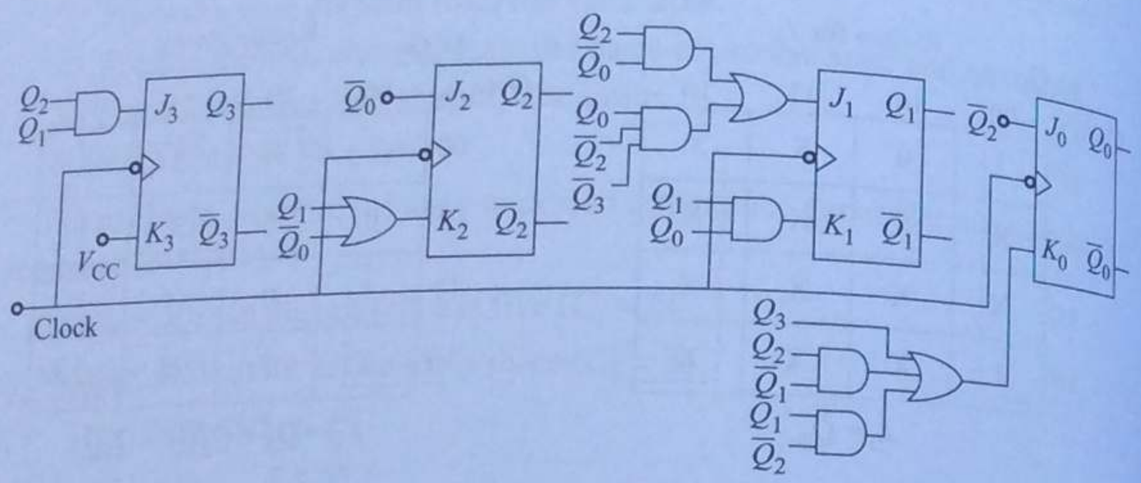


Fig. 5.135 Logic diagram to generate the sequence 11001011

Example 5.27 Design a sequence generator using D flip-flop to generate the sequence 101100110.

Solution

Number of 1s in the sequence is five ($C_1 = 5$)

Number of 0s in the sequence is four ($C_0 = 4$)

$$\text{Max } (4, 5) \leq 2^{n-1}$$

$$5 \leq 2^{n-1}$$

$$n = 4$$

The state table to generate the desired sequence is given in Table 5. 91.

The desired sequence (start from LSB) is assigned under the column of Q_0 and the values of Q_3, Q_2, and Q_1 are assigned as 0 or 1 such that all the states have different values.

Table 5.91 State table to generate the sequence 101100110

Q_3	Q_2	Q_1	Q_0	State
0	0	0	0	0
0	0	1	1	3
0	0	0	1	1
0	1	0	0	4
0	1	1	0	6
0	1	1	1	7
1	0	0	1	9
1	0	0	0	8
0	1	0	1	5

The state diagram for the above state table is given in Fig. 5.136.

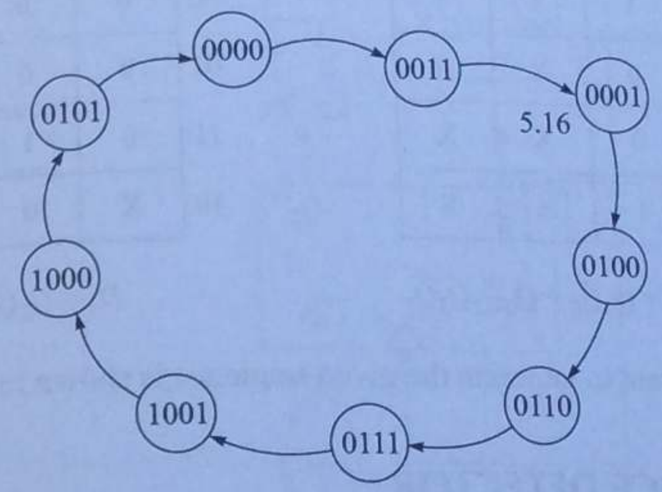


Fig. 5.136 State diagram for state table (Table 5.91)

The excitation table for the state diagram is shown in Table 5.92.

Table 5.92 Excitation table for the state diagram (Fig. 5.136) using D flip-flop

Present state				*Next state*				*Input Of Flip-flops*			
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	D_0	D_1	D_2	D_3
0	0	0	0	0	0	1	1	1	1	0	0
0	0	1	1	0	0	0	1	1	0	0	0
0	0	0	1	0	1	0	0	0	0	1	0
0	1	0	0	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1	1	1	1	0
0	1	1	1	1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	0	0	0	0	1
1	0	0	0	0	1	0	1	1	0	1	0
0	1	0	1	0	0	0	0	0	0	0	0

K-map for D_0

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	1	0	X	1
01	0	0	X	0
11	1	1	X	X
10	X	1	X	X

$D_0 = \overline{Q}_2\overline{Q}_0 + Q_1$

K-map for D_1

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	1	1	X	0
01	0	0	X	0
11	0	0	X	X
10	X	1	X	X

$D_1 = \overline{Q}_3\overline{Q}_2$

K-map for D_2

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	0	1	X	1
01	1	0	X	0
11	0	0	X	X
10	X	1	X	X

$D_2 = Q_2\overline{Q}_0 + Q_3\overline{Q}_0 + Q_0\overline{Q}_2\overline{Q}_3\overline{Q}_1$

K-map for D_3

Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	0	0	X	0
01	0	0	X	1
11	0	1	X	X
10	X	0	X	X

$D_3 = Q_3Q_0 + Q_1Q_0Q_2$

The logic diagram to generate the given sequence is shown in Fig. 5.137.

5.24 SEQUENCE DETECTOR

The sequence detector is a logic circuit that detects a desired sequence. The procedure to detect a desired sequence is as follows:

1. Draw the state diagram for the desired sequence. The procedure to draw the state diagram is explained in the example given below.
2. Draw the state table from the state diagram.
3. Find the required number of flip-flops.
4. Draw the excitation table for the state table.
5. Draw the K-map and simplify the same.
6. Draw the logic diagram.

Example 5.28 Design a sequence detector to detect the sequence 101011.

Solution

The number of states of a state diagram is equal to the number of bits in the desired sequence. In the given sequence, the number of bits is six and therefore, the number of states is six. The state diagram for the correct sequence is shown in Fig. 5.138. The states of the state diagram are defined by the alphabets *a*, *b*, *c*, *d*, *e*, and *f*. The output is considered as 0 when the complete correct sequence is not detected. Once the detector detects the correct sequence, the output becomes 1.

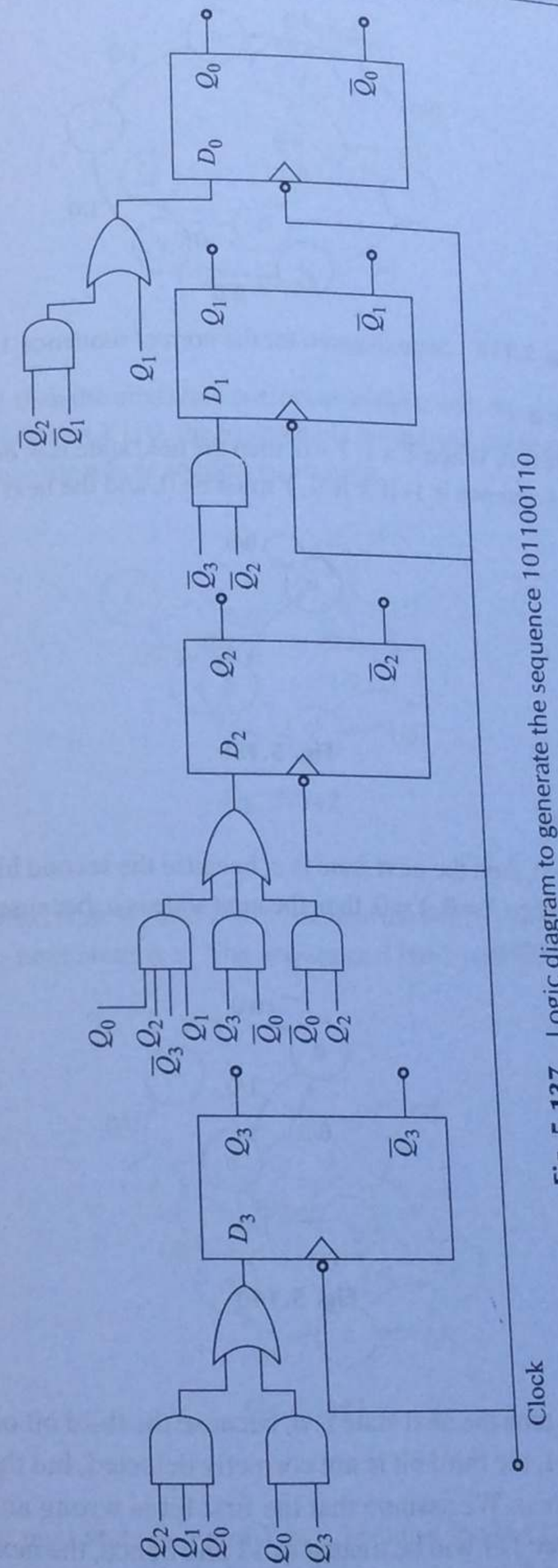


Fig. 5.137 Logic diagram to generate the sequence 101100110

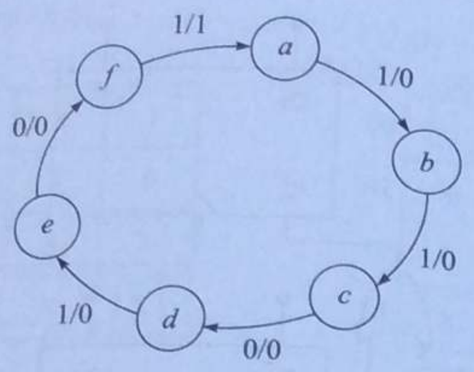


Fig. 5.138 State diagram for the correct sequence 101011

Present state a

The initial state is a. When $X = 1$, $Y = 0$, then the next state is b, because the first bit of the desired sequence is 1. If X is 0, Y must be 0, and the next state is a.

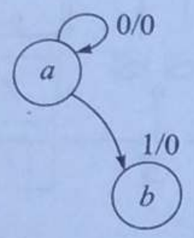


Fig. 5.139

Present state b

When $X = 1$, $Y = 0$, then the next state is c, because the second bit of the sequence is correct (11). When $X = 0$, $Y = 0$, then the next state is a, because the second bit of the sequence is wrong.

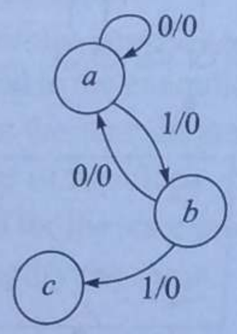


Fig. 5.140

Present state c

When $X = 0$, $Y = 0$, then the next state is d, because the third bit of the sequence is correct. When X is 1, the third bit is not correctly detected, but there is no need to go to the initial state a. We assume that the first bit is wrong and the third bit is correct. The sequence 11+ will be treated as 11 and hence, the next state is b and Y is 0.

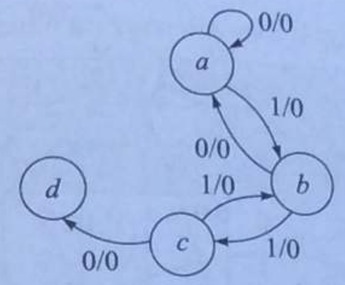


Fig. 5.141

Present state d

When $X = 1$, $Y = 0$, then the next state is e, because the fourth bit of the sequence is correctly detected. When X is 0, the fourth bit is wrong and the next state must be a and Y is 0, because the whole sequence is wrong.

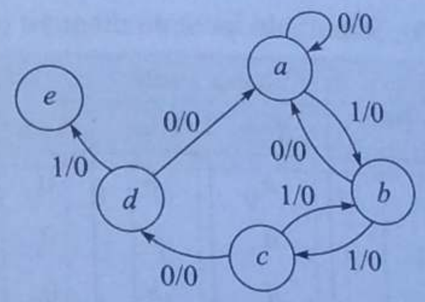


Fig. 5.142

Present state e

When X is 0, the next state is f and Y is 0, because the fifth bit is correctly detected. When X is 1, the next state is c. The sequence 11~~0~~1~~1~~ will be treated as 11 and Y is 0.

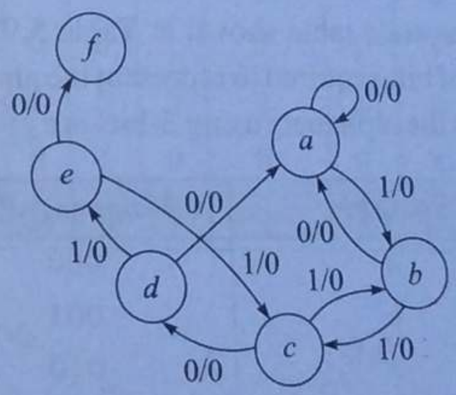


Fig. 5.143

Present state f

When X is 1, the next state is a and Y is 1, because the last bit of the sequence is correctly detected. When X is 0, the last bit is wrong and the next state must be a and Y is 0, because the whole sequence is wrong.

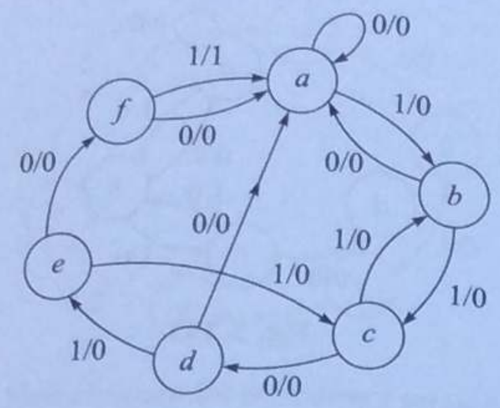


Fig. 5.144

State table The state table for the state diagram of Fig. 5.144 is given in Table 5.93.

Table 5.93 State table for state diagram (Fig. 5.144)

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

State reduction From the state table, it is observed that all the states are different and there is no possibility to reduce the state table.

State assignment The state table shown in Table 5.93 includes six alphabets and hence, the number of bits required to represent the alphabets in binary is three. The assigned values for the alphabets using 3-bits are given below.

Variable	Assigned value
a	000
b	001
c	010
d	011
e	100
f	101

The state table which shows the present state and the next state for input $X = 0$ and $X = 1$, and the output for $X = 0$ and $X = 1$ is given in Table 5.94.

Table 5.94 State table for state diagram (Fig. 5.144) in binary

Present state			Next state						Output	
			X=0			X=1			X=0	X=1
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Y	Y
0	0	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	0	0
0	1	0	0	1	1	0	0	1	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	1	0	1	0	1	0	0	0
1	0	1	0	0	0	0	0	0	0	1

The excitation table for the given state diagram using J-K flip-flop is given in Table 5.95

Table 5.95 Excitation table for state table (Table 5.94) using J-K flip-flop

Present state			Input	Next state			Output	Input of the flip-flops					
Q_2	Q_1	Q_0	X	Q_2	Q_1	Q_0	Y	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	0	0	0	0	0	0	X	0	X	0	X
0	0	1	0	0	0	0	0	X	1	0	X	0	X
0	1	0	0	0	1	1	0	1	X	X	0	0	X
0	1	1	0	0	0	0	0	X	1	X	1	0	X
1	0	0	0	1	0	1	0	1	X	0	X	X	0
1	0	1	0	0	0	0	0	X	1	0	X	X	1
0	0	0	1	0	0	1	0	1	X	0	X	0	X
0	0	1	1	0	1	0	0	X	1	1	X	0	X
0	1	0	1	0	0	1	0	1	X	X	1	0	X
0	1	1	1	1	0	0	0	X	1	X	1	1	X
1	0	0	1	0	1	0	0	0	X	1	X	X	1
1	0	1	1	0	0	0	1	X	1	0	X	X	1

K-map for J_0

Q_0X \ Q_2Q_1	00	01	11	10
00	0	1	X	1
01	1	1	X	0
11	X	X	X	X
10	X	X	X	X

$J_0 = Q_1 + \overline{Q}_2X + Q_2\overline{X}$

K-map for K_0

Q_0X \ Q_2Q_1	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	1	1	X	1
10	1	1	X	1

$K_0 = 1$

K-map for J_1

Q_0X \ Q_2Q_1	00	01	11	10
00	0	X	X	0
01	0	X	X	1
11	1	X	X	0
10	0	X	X	0

$J_1 = Q_2\overline{Q}_0X + Q_0\overline{Q}_2X$

K-map for K_1

Q_0X \ Q_2Q_1	00	01	11	10
00	X	0	X	X
01	X	1	X	X
11	X	1	X	X
10	X	1	X	X

$K_1 = Q_0 + X$

K-map for J_2

Q_0X \ Q_2Q_1	00	01	11	10
00	0	0	X	X
01	0	0	X	X
11	0	1	X	X
10	0	0	X	X

$J_2 = Q_1Q_0X$

K-map for K_2

Q_0X \ Q_2Q_1	00	01	11	10
00	X	X	X	0
01	X	X	X	1
11	X	X	X	1
10	X	X	X	1

$K_2 = Q_0 + X$

K-map for Y

Q_0X \ Q_2Q_1	00	01	11	10
00	0	0	X	0
01	0	0	X	0
11	0	0	X	1
10	0	0	X	0

$Y = Q_2Q_0X$

The logic diagram for the given state table using J-K flip-flop is shown in Fig. 5.145.

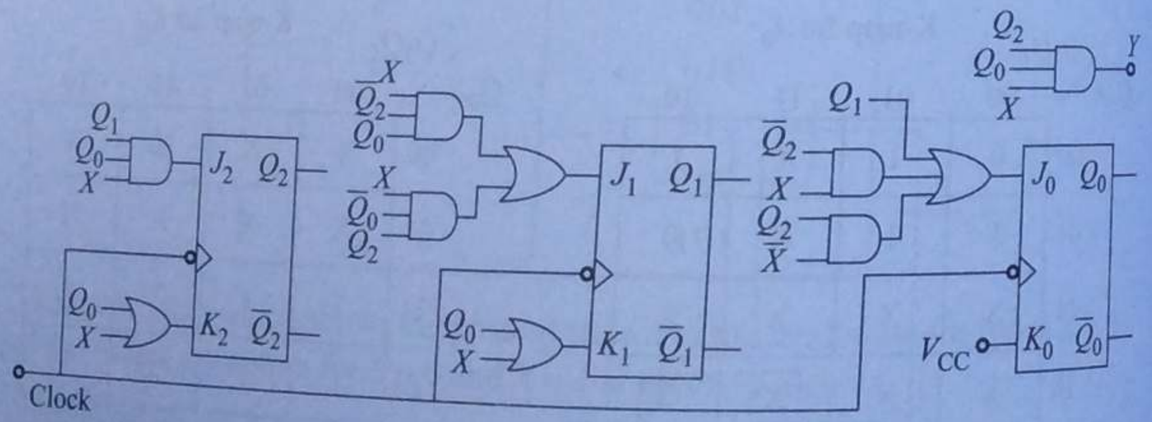


Fig. 5.145 Logic diagram for the sequence (101011) detector

Example 5.29 Design and implement a sequence detector to detect the sequence 1101011 using a clock sequence circuit and R-S FF.

Solution

In the given sequence, the number of bits is seven and therefore, the number of states is seven. The state diagram for the correct sequence is shown in Fig. 5.142.The states of the state diagram are defined by the alphabets a, b, c, d, e, f, and g. The output will be considered as 0 when the complete correct sequence is not detected. Once the detector detects the complete correct sequence, the output becomes 1.

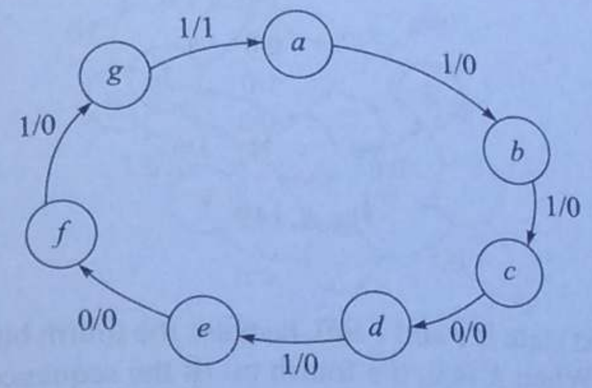


Fig. 5.146 State diagram for the correct sequence 1101011

Present state a

When X is 1, the next state is b and Y is 0, because the first bit of the desired sequence is 1. If X is 0, then Y must be 0 and the next state is a.

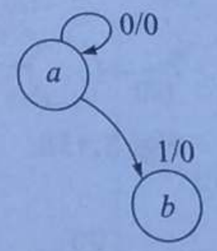


Fig. 5.147

Present state b

When X is 1, the next state is c and Y is 0, because the second bit (11) of the sequence is correct. When $X = 0$, $Y = 0$, then the next state is a, because the detected second bit of the sequence is wrong.

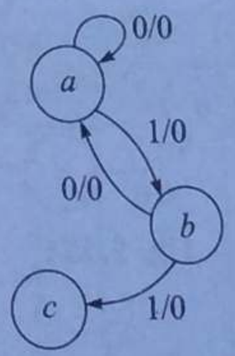


Fig. 5.148

Present state c

When X is 0, the next state is d and Y is 0, because the third bit of the sequence is correct. When X is 1, the third bit of the sequence is not correctly detected, but there is no need to go to the initial state a. We assume the first bit is wrong and the third bit is correct. The sequence 11+ will be treated as 11 and hence, the next state is b and Y is 0.

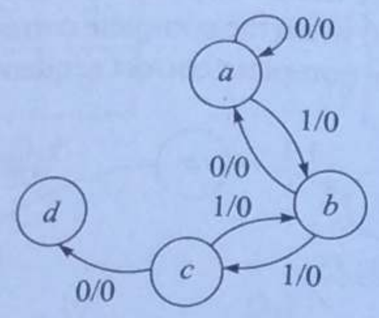


Fig. 5.149

Present state d

When X is 1, the next state is e and Y is 0, because the fourth bit of the sequence is correctly detected. When X is 0, the fourth bit of the sequence is wrong and the next state must be a and Y is 0, because the whole sequence is wrong.

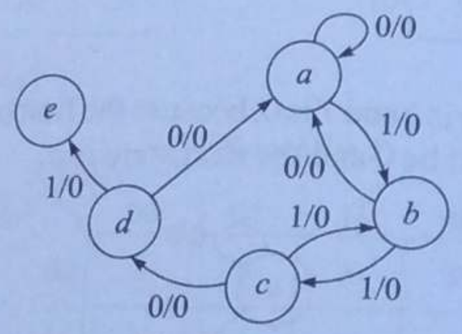


Fig. 5.150

Present state e

When X is 0, the next state is f and Y is 0, because the fifth bit is correctly detected. When X is 1, the next state is c. The sequence 11~~011~~ will be treated as 11 and Y is 0.

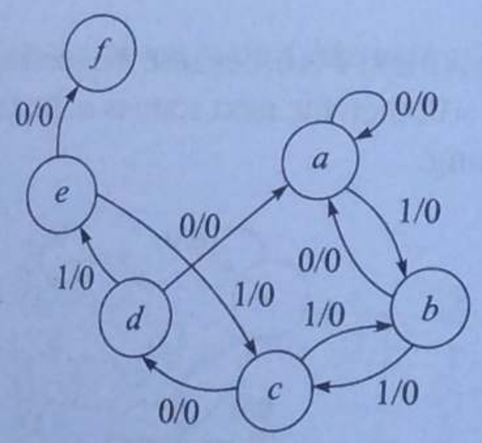


Fig. 5.151

Present state f

When X is 1, the next state is g and Y is 1, because the sixth bit of the sequence is correctly detected. When X is 0, the sixth bit of the sequence is wrong and the next state must be a, and Y is 0, because the whole sequence is wrong.

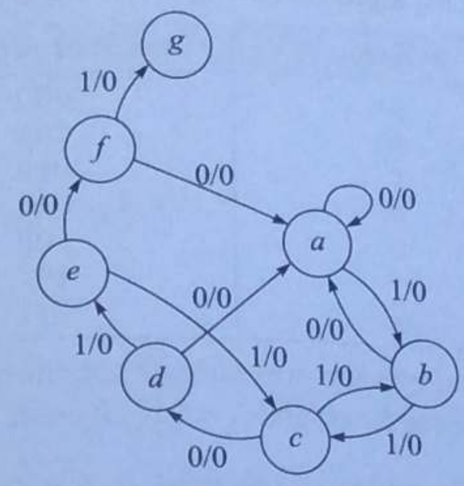


Fig. 5.152

Present state g

When X is 1, the next state is a and Y is 1, because the last bit of the sequence is correctly detected. When X is 0, the last bit of the sequence is wrong and the next state must be a and Y is 0, because the whole sequence is wrong.

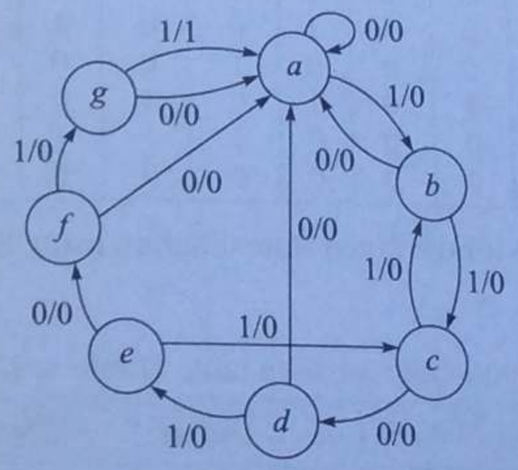


Fig. 5.153

State table: The state table for the state diagram of Fig. 5.153 is shown in Table 5.96.

Table 5.96 State table for state diagram (Fig. 5.153)

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

State reduction From the state table, it is observed that all the states are different and there is no possibility to reduce the state table.

State assignment The state table shown in Table 5.96 includes seven alphabets and hence, the number of bits required to represent the alphabets in binary is three. The assigned values for the alphabets using 3-bits are given below.

Variable	*Assigned value*
a	000
b	001
c	010
d	011
e	100
f	101
g	111

The state table which shows the present state and the next state for input $X=0$ and $X=1$, and the output for $X=0$ and $X=1$ is given in Table 5.97.

Table 5.97 State table for state diagram (Fig. 5.153) in binary

Present state			*Next state*						*Output*	
			X=0			*X=1*			*X=0*	*X=1*
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	*Y*	*Y*
0	0	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	0	0
0	1	0	0	1	1	0	0	1	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	1	0	1	0	1	0	0	0
1	0	1	0	0	0	1	1	1	0	0
1	1	1	0	0	0	0	0	0	0	1

The excitation table for the given state diagram using S-R flip-flop is given in the Table 5.98.

Table 5.98 Excitation table for state table (Table 5.97) using J-K flip-flop

Present state			*Input*	*Next state*			*Output*	*Input of the flip-flops*					
Q_2	Q_1	Q_0	*X*	Q_2	Q_1	Q_0	*Y*	S_0	R_0	S_1	R_1	S_2	R_2
0	0	0	0	0	0	0	0	0	X	0	X	0	X
0	0	1	0	0	0	0	0	0	1	0	X	0	X
0	1	0	0	0	1	1	0	1	0	X	0	0	X
0	1	1	0	0	0	0	0	0	1	0	1	0	X
1	0	0	0	1	0	1	0	1	0	0	X	X	0
1	0	1	0	0	0	0	0	0	1	0	X	0	1
1	1	1	0	0	0	0	0	0	1	0	1	0	1
0	0	0	1	0	0	1	0	1	0	0	X	0	X
0	0	1	1	0	1	0	0	0	1	1	0	0	X
0	1	0	1	0	0	1	0	1	0	0	1	0	X
0	1	1	1	1	0	0	0	0	1	0	1	1	0
1	0	0	1	0	1	0	0	0	X	1	0	0	1
1	0	1	1	1	1	1	0	X	0	1	0	X	0
1	1	1	1	0	0	0	1	0	1	0	1	0	1

K-map for S_0

Q_2Q_1 \ Q_0X	00	01	11	10
00	0	1	X	1
01	1	1	X	0
11	0	0	0	X
10	0	0	0	0

$S_0 = Q_1\overline{Q}_0 + \overline{Q}_2\overline{Q}_0X + Q_2\overline{Q}_0\overline{X}$

K-map for R_0

Q_2Q_1 \ Q_0X	00	01	11	10
00	X	0	X	0
01	0	0	X	X
11	1	1	1	0
10	1	1	1	1

$R_0 = Q_0\overline{X} + \overline{Q}_2Q_0 + Q_1Q_0$

K-map for S_1

Q_2Q_1 \ Q_0X	00	01	11	10
00	0	X	X	0
01	0	0	X	1
11	1	0	0	1
10	0	0	0	0

$S_1 = Q_2\overline{Q}_1X + Q_0\overline{Q}_1X$

K-map for R_1

Q_2Q_1 \ Q_0X	00	01	11	10
00	X	0	X	X
01	X	1	X	0
11	0	1	1	0
10	X	1	1	X

$R_1 = Q_0\overline{X} + Q_1X$

K-map for S_2

Q_2Q_1 \ Q_0X	00	01	11	10
00	0	0	X	X
01	0	0	X	0
11	0	1	0	X
10	0	0	0	0

$S_2 = \overline{Q}_2Q_1Q_0X$

K-map for R_2

Q_2Q_1 \ Q_0X	00	01	11	10
00	X	X	X	0
01	X	X	X	1
11	X	0	1	0
10	X	X	1	1

$R_2 = Q_0\overline{X} + Q_2Q_1 + \overline{Q}_0X$

K-map for Y

Q_2Q_1 \ Q_0X	00	01	11	10
00	0	0	X	0
01	0	0	X	0
11	0	0	1	0
10	0	0	0	0

$Y = Q_2Q_1X$

The logic diagram for the given state table using S-R flip-flop is shown in Fig. 5.154.

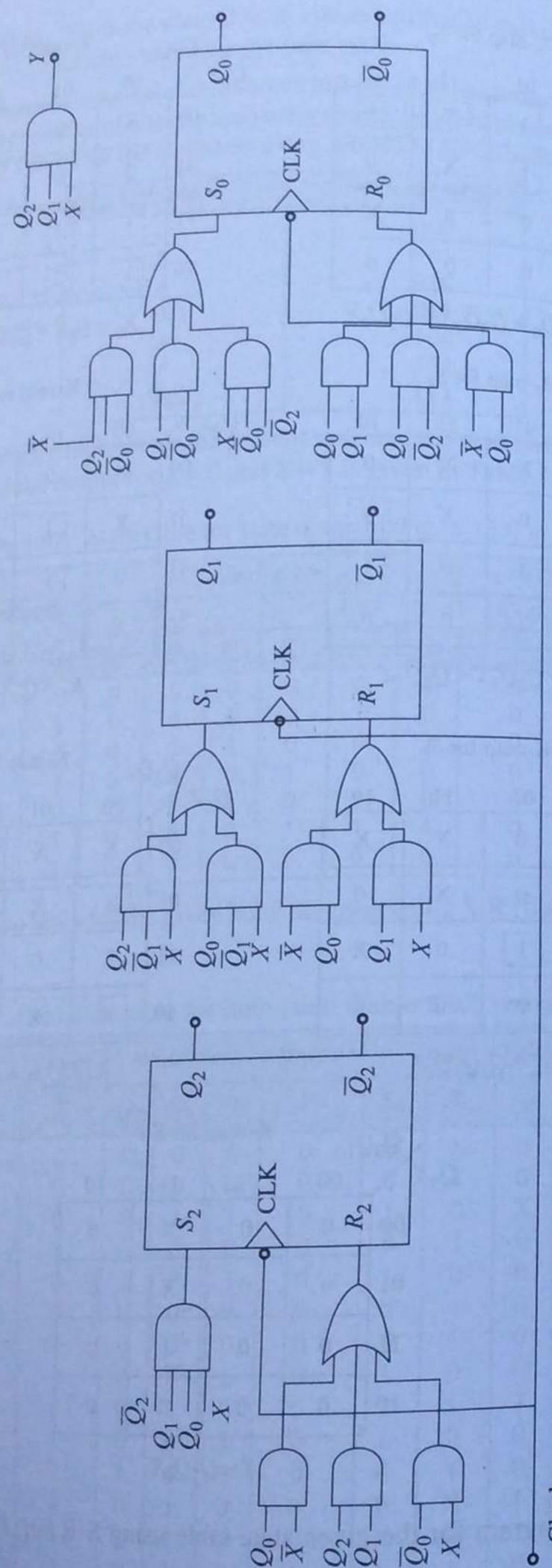


Fig. 5.154 Logic diagram for the sequence (101011) detector

Example 5.29 Design and implement a sequence detector 1101001 using a clock sequence circuit and D flip-flop.

Solution

In the given sequence, the number of bits is seven and therefore, the number of states is seven. The state diagram for the correct sequence is shown in Fig. 5.155. The states of the state diagram are defined by the alphabets a, b, c, d, e, f, and g. The output will be considered as 0, when the complete correct sequence is not detected. Once the detector detects the complete correct sequence, the output becomes 1.

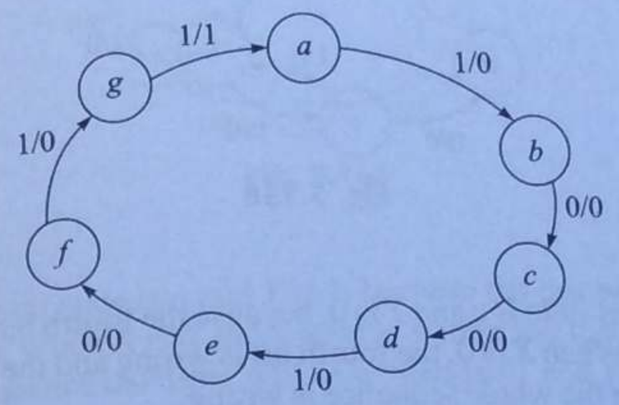


Fig. 5.155 State diagram for the correct sequence

Present State a

When X is 1, the next state is b and Y is 0, because the first bit of the desired sequence is 1. If X is 0, then Y must be 0 and the next state is a.

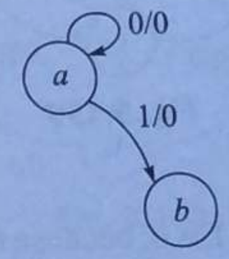


Fig. 5.156

Present State b

When X is 0, the next state is c and Y is 0, because the second bit of the sequence is correct. When X is 1, the next state is b, and Y is 0, because the second bit is not correctly detected, but there is no need to go to the initial state a. We assume that the first bit is wrong and the second bit is correct, 1+ will be considered as 1 and hence, the next state is b and Y is 0.

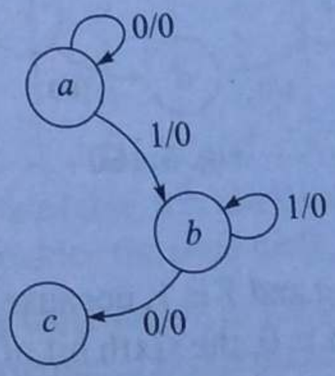


Fig. 5.157

Present state c

When X is 0, the next state is d and Y is 0, because the third bit of the sequence is correct. When X is 1, the third bit is not correctly detected, but there is no need to go to the initial state a. We assume that the first bit is wrong and the third bit is correct, ~~101~~ is considered as 1 and hence, the next state is b and Y is 0.

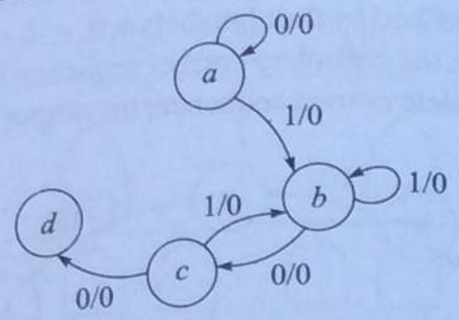


Fig. 5.158

Present state d

When X is 1, the next state is e and Y is 0, because the fourth bit of the sequence is correctly detected. When X is 0, the fourth bit is wrong and the next state must be a and Y is 0, because the whole sequence is wrong.

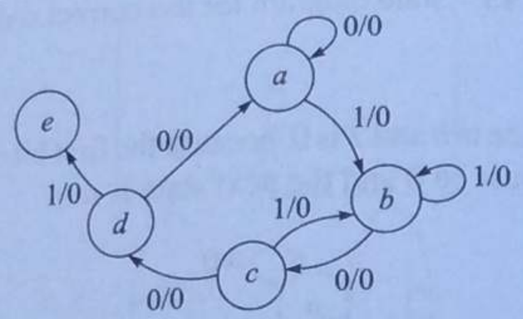


Fig. 5.159

Present state e

When X is 0, the next state is f and Y is 0, because the fifth bit is correctly detected. When X is 1, the next state is b. The sequence 1~~1001~~ will be treated as 1 and Y is 0.

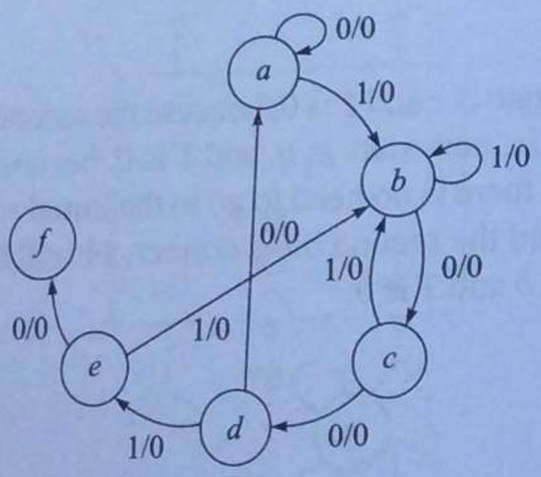


Fig. 5.160

Present state f

When X is 1, the next state is g and Y is 1, because the sixth bit of the sequence is correctly detected. When X is 0, the sixth bit of the sequence is wrong. The

next state must be d and Y is 0. The sequence 001~~001~~ will be treated as 001 and Y is 0.

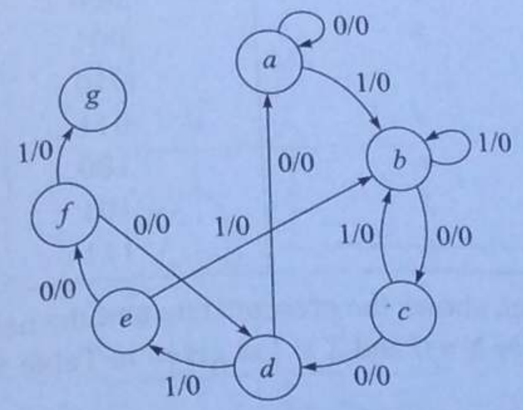


Fig. 5.161

Present state g

When X is 1, the next state is a and Y is 1, because the last bit of the sequence is correctly detected. When X is 0, the last bit is wrong and the next state must be c and Y is 0. The sequence 01~~01001~~ will be treated as 01 and Y is 0.

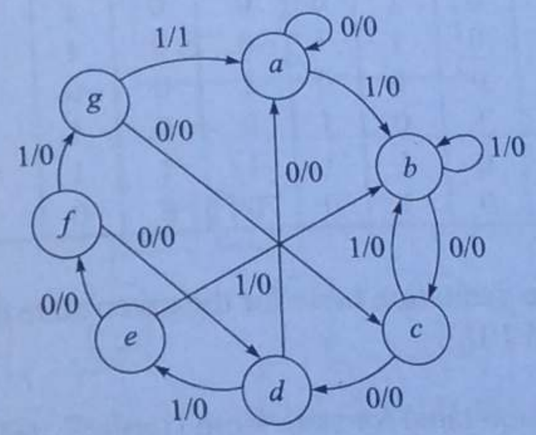


Fig. 5.162

State table The state table for the state diagram of Fig. 5.158 is shown in Table 5.99.

Table 5.99 State table for state diagram (Fig. 5.162)

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

State reduction From the state table, it is observed that all the states are different and there is no possibility to reduce the state table.

State assignment The state table shown in Table 5.99 includes seven alphabets and hence, the number of bits required to represent the alphabets in binary is three. The assigned values for the alphabets using 3-bits are given in the following table.

Variable	Assigned value
a	000
b	001
c	010
d	011
e	100
f	101
g	111

The state table which shows the present state and the next state for $X = 0$ and $X = 1$, and the output for $X = 0$ and $X = 1$ is given in Table 5.100.

Table 5.100 State table for state diagram (Fig. 5.159) in binary

Present state			Next state						Output	
			X=0			X=1			X=0	X=1
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Y	Y
0	0	0	0	0	0	0	0	1	0	0
0	0	1	0	1	0	0	0	1	0	0
0	1	0	0	1	1	0	0	1	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	1	0	1	0	0	1	0	0
1	0	1	0	1	1	1	1	1	0	0
1	1	1	0	1	0	0	0	0	0	1

Excitation table The excitation table for the given state diagram using D flip-flop is given in Table 5.101.

Table 5.101 Excitation table for state table (Table 5.100) using D flip-flop

Present state			Input	Next state			Output	Input of the flip-flops		
Q_2	Q_1	Q_0	X	Q_2	Q_1	Q_0	Y	D_0	D_1	D_2
0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	0	1	0
0	1	0	0	0	1	1	0	1	1	0
0	1	1	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	C	1	0	1
1	0	1	0	0	1	1	0	1	1	0
1	1	1	0	0	1	0	0	0	1	0
0	0	0	1	0	0	1	0	1	0	0
0	0	1	1	0	0	1	0	1	0	0
0	1	0	1	0	0	1	0	1	0	0
0	1	1	1	1	0	0	0	0	0	1
1	0	0	1	0	0	1	0	1	0	0
1	0	1	1	1	1	1	0	1	1	1
1	1	1	1	0	0	0	1	0	0	0

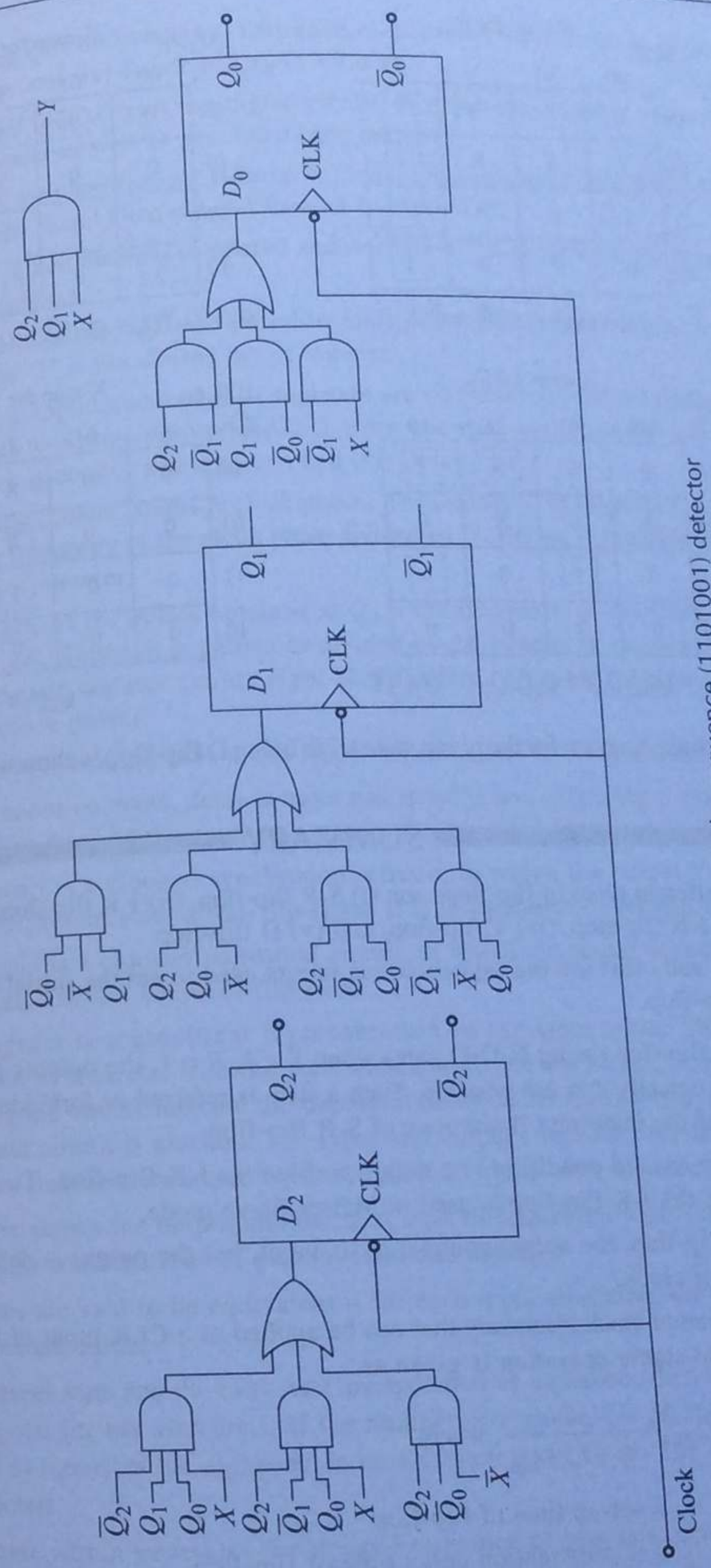


Fig. 5.165 Logic diagram for the sequence (1101001) detector

K-map for D_0

Q_0X \ Q_2Q_1	00	01	11	10
00	0	1	X	1
01	1	1	X	1
11	1	0	0	1
10	0	0	0	1

$D_0 = Q_2\overline{Q}_1 + Q_1\overline{Q}_0 + \overline{Q}_1X$

K-map for D_1

Q_0X \ Q_2Q_1	00	01	11	10
00	0	1	X	0
01	0	0	X	0
11	0	0	0	1
10	1	0	1	1

$D_1 = \overline{Q}_0\overline{X}Q_1 + Q_2Q_0\overline{X} + Q_2\overline{Q}_1Q_0 + \overline{Q}_1\overline{X}Q_0$

K-map for D_2

Q_0X \ Q_2Q_1	00	01	11	10
00	0	0	X	1
01	0	0	X	0
11	0	1	0	1
10	0	0	0	0

$D_2 = \overline{Q}_2Q_1Q_0X + Q_2\overline{Q}_1Q_0X + Q_2\overline{Q}_0\overline{X}$

K-map for Y

Q_0X \ Q_2Q_1	00	01	11	10
00	0	0	X	0
01	0	0	X	0
11	0	0	1	0
10	0	0	0	0

$Y = Q_2Q_1X$

The logic diagram for the given state table using D flip-flop is shown in Fig. 5.163.

SUMMARY

- The different types of flip-flops are: (i) S-R flip-flop, (ii) J-K flip-flop, (iii) Master-Slave J-K flip-flop, (iv) T flip-flop, and (v) D flip-flop.
- Preset and clear are the asynchronous inputs used to set the initial condition of the flip-flop.
- In S-R flip-flop (using NAND gate) when $S = 1$, $R = 1$, the outputs Q and $\overline{Q}$ are same. Logically it is not possible. Such a state is referred as forbidden state. This is one of the important limitations of S-R flip-flop.
- The race-around condition is a major problem in J-K flip-flop. To resolve this problem, the J-K flip-flop is used in Master-Slave mode.
- In a D flip-flop, the output is equal to its input, but the output is delayed by the amount of clock.
- The maximum clock frequency that can be applied as a CLK input of flip-flop for proper and stable operation is given as

$$f_{max} = \frac{1}{t_s + t_p + t_{np}}, \text{ where}$$

t_s = set-up time of flip-flop
t_p = is propagation delay time of flip-flop
t_{np} = is propagation of next stage decoder

- The flip-flop can be used in (i) Bounce elimination switch (ii) Registers (iii) Counters, and (iv) Random Access Memory.
- The sequential logic circuit is a digital circuit, in which the output is a function of present inputs as well as past inputs and outputs.
- The registers are classified as (i) Serial In Serial Out, (ii) Serial In Parallel Out, (iii) Parallel In Serial Out, and (iv) Parallel In Parallel Out.
- Registers in which the data is entered and/or retrieved in serial form are referred to as shift registers.
- A bidirectional shift register is used to multiply or divide the number by 2^n, provided that '1' is not shifted out of register.
- The common applications of shift registers are (i) Serial-to-Parallel converter, (ii) Parallel-to-Serial converter, (iii) Ring counter, (iv) Johnson counter, (v) Sequence generator and detector, and (vi) Delay line.
- The N-bit ring counter counts N clock pulses. The frequency of the pulses obtained at Q_0 is the frequency of the clock pulse divided by N. Hence, it is also known as divided-by-N counter.
- The frequency of the pulses obtained at Q_0 is the frequency of the clock pulses divided by $2N$. Hence, it is known as divided-by-$2N$ counter or twisted counter. A 4-bit Johnson counter counts eight-clock pulses. An n-bit Johnson counter counts 2^n clock pulses.
- The n-bit ripple counter uses n flip-flops, which counts 2^n clock pulses.
- In asynchronous counters, delay is more and speed is less. Applying a clock pulse to all the flip-flops of the counter increases the speed of operation.
- Moore circuit is a clocked synchronous circuit, in which the output Y depends only on the present state of the flip-flops. It is independent of present input(s).
- Mealy circuit is a clocked sequential circuit, in which the output depends on the present state and the inputs of flip-flops.
- State diagram is a graphical representation of the state table. The state is represented by a circle, with the state indicated inside the circle. Directed lines connecting the circles indicate the transition between the states when the input is applied and circuit is clocked. The input and output conditions are labeled with two binary numbers separated by the symbol '/' along with directed lines.
- State table shows the output and the next state of sequential logic circuit for all the possible combinations of present state and input(s).
- Two states are said to be equivalent if for each input condition, the outputs and next states are same.
- If the present state and the next state are specified by alphabets, then binary values are assigned for the alphabets. If the number of alphabets is M, then n-bits are required to represent the alphabets in binary, such that $2^n \geq M$. The least value of n is selected.
- The circuit which generates the desired sequence of bits in synchronous mode with a clock is known as a sequence generator.
- The sequence detector is a logic circuit which detects the desired sequence.

KEY TERMS

Sequential circuit In a sequential circuit, the output is dependent upon the present inputs as well as the past inputs and outputs. Sequential circuits include memory elements, which store the past inputs and outputs.

Flip-flop In digital systems, data is represented by two states, logic 0 state and logic 1 state. A 1-bit memory cell stores the data in terms of 0 or 1. This 1-bit memory cell is commonly referred as a flip-flop.

Clocked S-R flip-flop The simple S-R flip-flop will be set or reset by changing the inputs S and R. Most of the applications demand a memory cell, which sets or resets in synchronous with the clock. The clock is nothing but a train of pulses.

D flip-flop In a J-K flip-flop, $Q_{n+1} = J_n$, under the condition that K_n input is complement of J_n input. When K_n input is complement of J_n input; then the output is same as input J, it is delay by the amount of clock. Such a flip-flop is known as delay flip-flop or D flip-flop. The D flip-flop is obtained from the J-K flip-flop by connecting a NOT gate between the J and K input of J-K flip-flop.

T flip-flop It has a single input. The output of T flip-flop is either same or complement of the previous output.

Triggered flip-flop When the output of a flip-flop change in presence of a clock pulse, then it is known as a level-triggered flip-flop. In some of the applications, it is required that the output of flip-flop should respond, when the clock changes its state.

Registers The flip-flop is a 1-bit memory cell, which stores one bit of digital data. A group of flip-flops can store more than one bit of digital data. An array of flip-flops is known as a register. The data can be entered and retrieved in serial or parallel form.

EXERCISE

Review Questions

1. Explain the following flip-flops:
   (a) Clocked R-S
   (b) J-K with preset and clear
   (c) Master-Slave J-K
   (d) D-type and T-type
2. Design a 4-bit binary up/down ripple counter.
3. Explain the basic flip-flop circuit for R-S using
   (a) NAND gates
   (b) NOR gates
4. Design a 3-bit binary up/down counter. Draw its timing diagram.
5. Design the conversion logic to convert a J-K flip-flop to D flip-flop.
6. Design and implement a Mod-6 synchronous counter using J-K flip-flop.
7. Design a pulse train generator using shift register for the pulse train 111010.
8. Write a short note on race-around condition in J-K flip-flop.
9. Design and implement a Mod-5 synchronous counter using J-K flip-flop.
10. Design a pulse train generator using shift register for the pulse train 1000110.
11. Draw a neat circuit diagram of clocked J-K flip-flop using NAND gates. Give its truth table and explain race-around condition.

12. Design and implement a sequence generator using 1101011 shift register.
13. Draw a neat circuit diagram of a 3-bit bidirectional shift register using J-K flip-flops having right and left data inputs and mode control M such that $M = 0$ left shift, $M = 1$ right shift.
14. What is race-around condition? How does it get eliminated in a Master-slave J-K flip-flop? Explain.
15. Design and implement a Mod-5 synchronous counter using T flip-flops.
16. Design a Mod-4 up/down counter.
17. Design and implement a Mod-6 synchronous counter using D flip-flop.
18. Design a 4-bit synchronous counter using J-K flip-flops. Use K-maps.
19. Explain why a synchronous counter is faster than a ripple counter.
20. What is the difference between truth table and excitation table?
21. Explain how a J-K flip-flop is converted into D flip-flop and T flip-flop.
22. Draw the logical diagram of a 4-bit shift register. Explain how shift-left and shift-right operations are performed.
23. Explain with a neat diagram
    (i) 3-bit bidirectional shift register using D flip-flop
    (ii) 3-bit ripple down counter using T flip-flop
24. Assume that '1011' input data pattern is loaded into a 4-bit ring counter. Sketch the resulting flip-flop Q output waveforms. (Assume positive edge triggering.)
25. What is meant by universal shift register? Explain with the help of function table, the working of a 4-bit universal shift register IC.
26. Draw a neat circuit diagram of a 3-bit Johnson counter. Draw the relevant output waveforms.
27. Draw a neat circuit diagram of Master-Slave J-K flip-flop using NAND gates. Explain the working of the circuit for the following conditions of J and K inputs:

CLK	*J*	*K*	*O/P* (Q_{n+1})
0	1	0	1
1	1	1	$\overline{Q}_n$

28. Carry out the following flip-flop conversions (Conversion tables and K-maps expected)
    (i) S-R to D
    (ii) D to S-R
    (iii) J-K to S-R
29. With a suitable example, explain Moore and Mealy circuits.
30. Write notes on
    (a) State diagram
    (b) State table
    (c) State reduction
    (d) State assignment
31. Explain the procedure to design and analyse a clock sequential circuit.
32. Write and explain the steps to design a sequence generator.
33. Write and explain the steps to design a sequence detector.
34. Design a sequential circuit for the state diagram shown in Fig. 5.164 using J-K flip-flops.

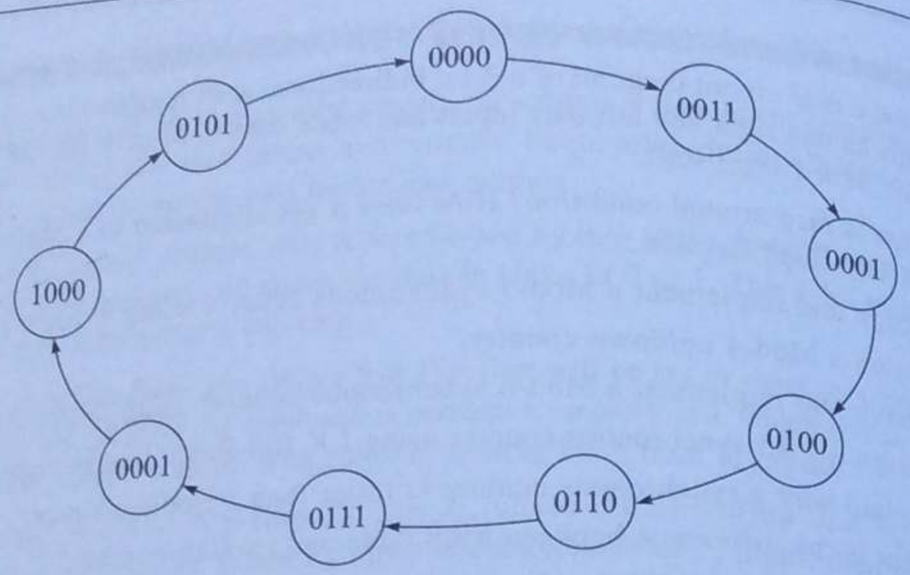


Fig. 5.164

35. With the help of state table and state diagram, design a Mod-9 up/down counter.
36. Design a clocked sequential circuit for the state diagram shown in Fig. 5.165 using J-K flip-flop.

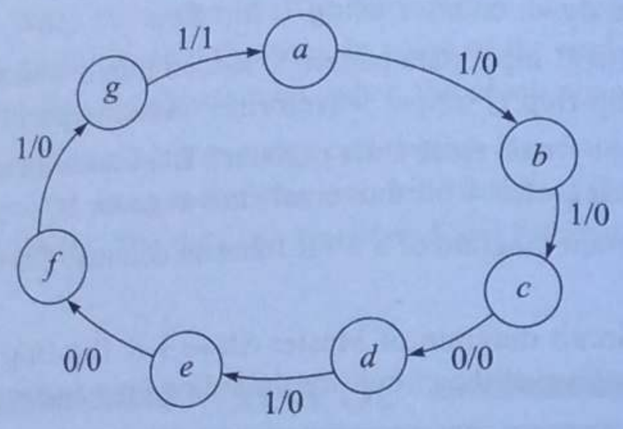


Fig. 5.165

37. Design a sequential generator to generate the sequence $0 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 1 \rightarrow 7 \rightarrow 5$ using T flip-flop.
38. Design a circuit to generate the sequence $0 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 7 \rightarrow 3$.
39. Design a sequence generator to generate the sequence $0 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$ and avoid the lockout condition using J-K flip-flops.
40. Design a sequence generator to generate the sequence 101101100 using D flip-flop.
41. Design and implement a sequence generator to generate the sequence 1101000 using a clock sequence circuit and R-S flip-flop.

Multiple-choice Questions

Select the correct alternative

1. Placing an inverter between the inputs of an S-R flip-flop makes it a ..... flip-flop.
   (a) J-K (b) D
   (c) T (d) Master slave J-K

2. The output Q_n of a J-K flip-flop is 0. It changes to 1 when a clock pulse is applied. The inputs J_n and K_n are respectively .....
   (a) 1 and X (b) 0 and X
   (c) X and 0 (d) X and 1
3. The characteristic equation of a T-flip-flop is given by .....
   (a) $Q^+ = \overline{T}Q + T\overline{Q}$ (b) $Q^+ = T\overline{Q} + Q\overline{T}$
   (c) $Q^+ = TQ$ (d) $Q^+ = T\overline{Q}$
4. An R-S latch is a .....
   (a) combinational circuit (b) synchronous sequential circuit
   (c) 1-bit memory element (d) one clock delay element
5. A Mod-2 counter followed by a Mod-5 counter is .....
   (a) same as a Mod-5 counter followed by a Mod-2 counter
   (b) a decade counter
   (c) Mod-7 counter
   (d) none of the above
6. A 4-bit Mod-16 ripple counter uses a J-K flip-flop. If the propagation delay of each flip-flop is 50 ms, the maximum clock frequency is equal to .....
   (a) 20 MHz (b) 10 MHz
   (c) 5 MHz (d) 4 MHz
7. A pulse train with a frequency of 1 MHz is counted using a Mod-1024 ripple-counter built with J-K flip-fop. For proper operation of the counter, the maximum permissible propagation delay per flip-flop stage is ..... ns.
   (a) 100 (b) 50
   (c) 20 (d) 10
8. The Q-output of J-K flip-flop is 1. The output does not change when a clock pulse is applied. The inputs J and K will be respectively .....
   (a) 0 and X (b) X and 0
   (c) 1 and 0 (d) 0 and 1
9. A divide-by-78 counter can be realized by using .....
   (a) 6 Mod-13 counters
   (b) 13 Mod-6 counters
   (c) 1 Mod-13 counter followed by 1 Mod-6 counter
   (d) 13 Mod-13 counters
10. In sequential circuits, the outputs at any instant of time depend .....
    (a) only on the inputs present at that instant of time
    (b) on past output as well as present inputs
    (c) only on the past inputs
    (d) only on the past outputs
11. A ring counter is same as .....
    (a) up-down counter (b) parallel counter
    (c) shift registers (d) none of these
12. The number of flip-flops required to build a Mod-15 counter is .....
    (a) 4 (b) 5
    (c) 6 (d) 7

13. How many flip-flops are required for dividing the frequency by 64?
    (a) 4 (b) 5
    (c) 6 (d) 8
14. An n-stage ripple counter can count upto .....
    (a) 2^n (b) $2^n - 1$
    (c) n (d) 2^{n-1}

Answers

1. (b)	2. (a)	3. (b)	4. (c)
5. (b)	6. (c)	7. (a)	8. (b)
9. (c)	10. (b)	11. (d)	12. (a)
13. (c)	14. (c)		

Chapter 6

ASYNCHRONOUS SEQUENTIAL CIRCUITS

Chapter Outline

- Basic concepts of asynchronous sequential circuits and their classification
- Design of fundamental mode and pulse mode asynchronous sequential circuits
- Problems of asynchronous sequential circuits: Cycles, Races, and Hazards

6.1 INTRODUCTION

Digital circuits are of two types, namely *combinational* and *sequential circuit*. The combination circuit has been discussed in Chapter 4. In sequential circuits, the output is dependent upon the present inputs as well as the past inputs and outputs. A sequential circuit can be classified into *synchronous* and *asynchronous* circuit.

In a synchronous sequential circuit, transitions from one state to next state are controlled by clock pulse; but in an asynchronous sequential circuit, transitions from one state to next state are not controlled by clock pulse—transitions occur whenever there is a change in input to the circuit.

Sequential circuits without clock pulses are called *asynchronous sequential circuits*. It can be classified into two types: *fundamental mode* asynchronous sequential circuits and *pulse mode* asynchronous sequential circuits. In fundamental mode circuits, inputs and outputs are represented by levels, and in pulse mode asynchronous sequential circuits, inputs and outputs are represented by pulses. In the design of synchronous circuits, it is assumed that a change occurs in only one of the input and no change occurs in any other inputs until the circuit enters a stable state. A block diagram of an asynchronous sequential circuit is shown in Fig. 6.1.

The level signals applied at the input lines $I_1, I_2, ..., I_m$, and the output of memory devices, namely $F_1, F_2, ..., F_K$ are called the state of the circuit. The input signals at $I_1, I_2, ..., I_m$ are the input state variables of the circuits. The level signals that appear at the outputs of the combinational circuit, namely $O_1, O_2, ..., O_n$ are the outputs of the entire circuits, and the signals that appear at the inputs of memory devices $Y_1, Y_2, ..., Y_k$ are called the excitation or next state variables.

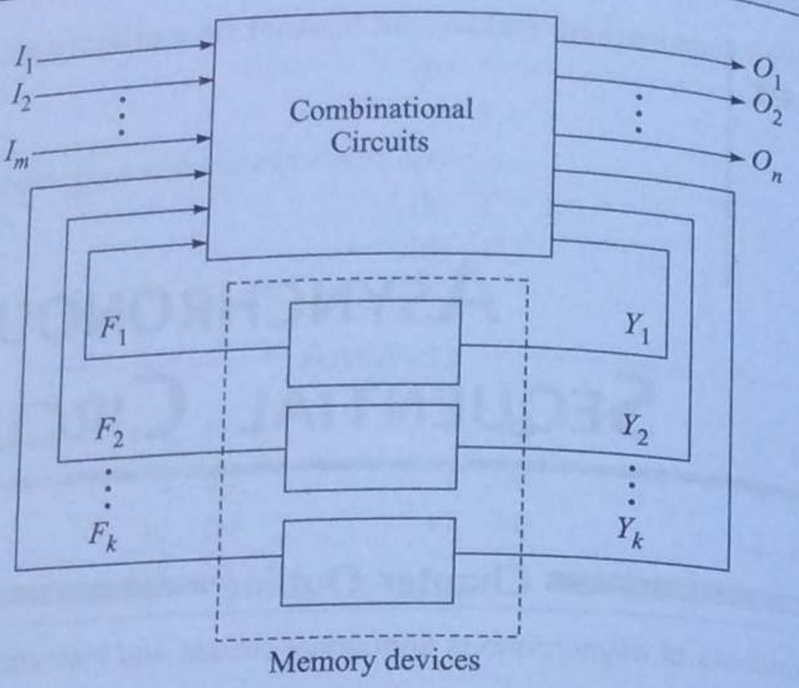


Fig.6.1 Block diagram of asynchronous sequential circuit

In an asynchronous sequential circuit, the circuit is said to be in stable state if the outputs of memory devices become equal to their inputs ($F_i = Y_i$ for $i = 1, 2, ..., k$ for a given set of inputs). If there is a change in the inputs, then the combinational circuit produces a new set of excitation variables, Y_1, Y_2, ..., Y_k, at the inputs of memory devices and the circuit enters into an unstable state. When the outputs of memory devices become equal to their inputs, the circuit is said to enter the next stable state.

6.2 FUNDAMENTAL MODE ASYNCHRONOUS SEQUENTIAL CIRCUITS

In fundamental mode asynchronous sequential circuits, the inputs and outputs are represented by their levels rather than pulses. Designing a fundamental mode asynchronous sequential circuit requires the following steps:

- Formulate what the circuit has to do, from the problem definition.
- Develop a primitive state table and specify the outputs that are associated with the stable state.
- Minimize the primitive state table by merge process. It is the process of identifying those primitive states that are related by input sequences and can be combined together into a single stable state, i.e. finding the states that can be put together in the same row of the excitation map. At the end of this merge process, the number of rows will be reduced.
- Assign state variables (i.e. secondary variables) to the rows of merged primitive state table and obtain the present state/next state and output table. The outputs assigned to the unstable states vary according to the design requirements.
- Decide the memory element to be used and obtain the excitation and output table. Obtain the simplified expression of the excitation and output function.
- Draw the schematic circuit diagram.

Let us consider the design example of fundamental mode asynchronous sequential circuits.

Example 6.1 Design an asynchronous sequential circuit with two input, I_1 and I_2, and one output, Z. Initially, both inputs are equal to zero. When I_1 or I_2 becomes 1, Z becomes 1. When the second input goes to 1, the output changes from 1 to 0. The output stays at 0 until the circuit goes back to (0, 0).

Solution

The primitive state table for the given requirement can be constructed as shown in Table 6.1.

Table 6.1 Primitive state table

Row	*Next state, Output*			
	$I_1I_2 = 0$	$I_1I_2 = 0\ 1$	$I_1I_2 = 1\ 11$	$I_1I_2 = 1\ 0$
1	①,0	2		3
2		②,1	4	
3			4	③,1
4			④,0	
5	1	⑤,0	4	
6	1		4	⑥,0

Initially, both the inputs are 0 (i.e. $I_1 = I_2 = 0$), and the circuit is said to be in a stable state 1, represented as ①, 0, where the circle represents the stable state and the output is 0.

When I_1 remains same at 0 and I_2 becomes 1, the circuit enters a stable state 2 and its output is 1. This is represented as ②, 1 in the second row. In the first row, an uncircled 2 is entered to indicate that a transition to ② state will occur as a result of the input change from 00 to 01. The uncircled 2 represents an unstable state.

Similarly, when I_1 becomes 1 and I_2 remains same at 0, the circuit enters a stable state 3 and its output is 1. This is represented as ③, 1 in the third row. In the first row, an uncircled 3 is entered to indicate that a transition to ③ state will occur as a result of the input change from 00 to 10. The uncircled 3 represents an unstable state.

When the circuit is in stable state ② or ③ (i.e. $I_1I_2 = 01$ or $I_1I_2 = 10$) and the other input also becomes 1 ($I_1I_2 = 11$), then the circuit is in stable state ④ with output 0. This is represented as ④, 0 in the fourth row. In second and third rows, an uncircled 4 is entered to indicate that a transition to ④ state will occur as a result of the input change from 01 to 11 or from 10 to 11. The uncircled 4 represents an unstable state. In an asynchronous circuit, both inputs cannot change simultaneously, hence stable states 5 and 6 are shown in Table 6.1.

When the circuit is in ④, and I_1 becomes 0, then the circuit enters ⑤ with output 0. When I_2 becomes 0 too, it will go to the initial state. Similarly, if the circuit is in ④, and I_2 becomes 0, the circuit enters ⑥ with output 0. From here, when I_1 becomes 0, it will go to ①.

To complete the partial primitive state table, the transitions from every stable state for each change in the input should be specified. Since two inputs are not allowed to change simultaneously, a dash is entered at such places.

A circled entry and an uncircled entry can be combined to minimize the primitive state table by merge process. From Table 6 .1, it is clear that the first three rows can be combined and similarly, the last three rows can be combined. Whenever a circled entry and an uncircled entry are combined as a circled entry, as shown in Table 6.2, the outputs associated with unstable states correspond to their stable states.

Table 6.2 Minimized state table

Row number	*Next state, Output*			
	$I_1 I_2 = 0\,0$	$I_1 I_2 = 0\,1$	$I_1 I_2 = 1\,1$	$I_1 I_2 = 1\,0$
1	①,0	②,1	4,0	③,1
2	1 0	⑤,0	④,0	⑥,0

Since there are only two rows in the merged state table, a 1-bit variable 0 and 1 can be assigned to row 1 and row 2, respectively. The present state/next state and output table is given in Table 6.3.

Table 6.3 Present state /next state and output table

Row number	*Next state, Output*			
	$I_1 I_2 = 0\,0$	$I_1 I_2 = 0\,1$	$I_1 I_2 = 1\,1$	$I_1 I_2 = 1\,0$
0	0 0	0 1	1,0	0,1
1	0 0	1 0	1,0	1,0

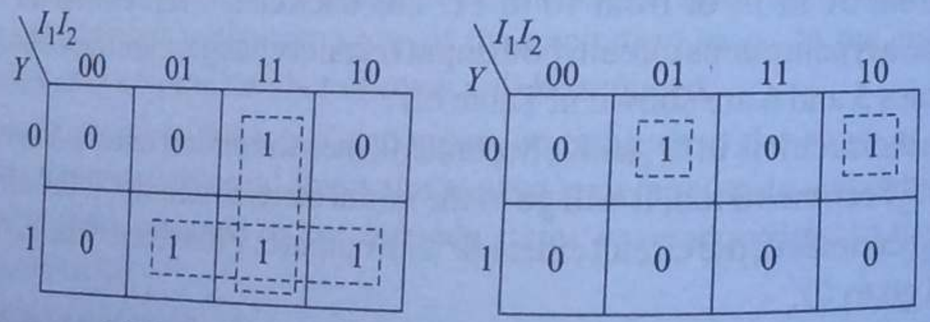
The present state/next state and output table can be obtained by replacing every circled entry in the first row by 0 and in the second row by 1. The unstable state 4 in the first row is assigned 1, since it must go to stable state 4 which belongs to second row. Similarly, the unstable state 1 in the second row is assigned 0.

If a delay flip-flop is used, then an excitation and output table is generated, as given in Table 6.4.

Table 6.4 Excitation and output table

Y	*D, Z*			
	$I_1 I_2 = 0\,0$	$I_1 I_2 = 0\,1$	$I_1 I_2 = 1\,1$	$I_1 I_2 = 1\,0$
0	0 0	0 1	1,0	0,1
1	0 0	1 0	1,0	1,0

The K-maps for the function (D) and output (Z) with inputs $I_1 I_2$ and Y are as shown in Fig. 6.2.



(a) K-map for D (b) K-map for Z

Fig. 6.2 K-map simplification for excitation and output functions

The simplified expressions for D and Z are:

$$D = I_1 I_2 + I_2 Y + Y I_1 \tag{6.1}$$

$$Z = \bar{I}_1 I_2 \bar{Y} + I_1 \bar{I}_2 \bar{Y}$$

$$= \bar{Y}(\bar{I}_1 I_2 + I_1 \bar{I}_2)$$

$$= \bar{Y}(I_1 \oplus I_2) \tag{6.2}$$

The circuit diagram for the given asynchronous sequential circuit is shown in Fig. 6.3.

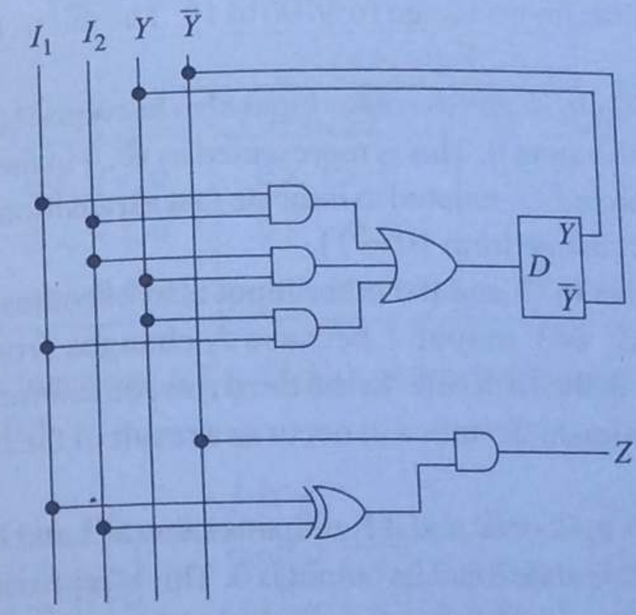


Fig. 6.3 Circuit diagram of Example 6.1

Example 6.2 Design an asynchronous sequential circuit with two inputs, I_1 and I_2 and one output, Z. Initially, both inputs are equal to 0. When I_1 changes from 0 to 1, Z becomes 1. When I_2 changes from 0 to 1, Z becomes 0. Otherwise, Z is 0. Realize the circuit using J-K flip flop.

Solution

The primitive state table for the given requirement is given in Tables 6.5.

Table 6.5 Primitive state table

Row number	*Next state, Output*			
	$I_1 I_2 = 0\,0$	$I_1 I_2 = 0\,1$	$I_1 I_2 = 1\,1$	$I_1 I_2 = 1\,0$
1	①,0	3		2
2	1		4	②,1
3	1	③,0	5	
4		3	④,0	6
5		3	⑤,1	6
6	1		4	⑥,0

Initially, both inputs are equal to 0 (i.e. $I_1 = I_2 = 0$), the circuit is said to be in a stable state 1 and the output is 0, represented as ①, 0.

When I_2 remains same at 0 and I_1 becomes 1, the circuit enters a stable state 2 and its output is 0. This is represented as ②, 1 in the second row. In the first row, an uncircled 2 is entered to indicate that a transition to ② state will occur as a result of the input change from 00 to 10. The uncircled 2 represents an unstable state.

Similarly, when I_1 remains same at 0 and I_2 becomes 1, the circuit enters a stable state 3 and its output is 0. This is represented as ③, 0 in the fourth row. In the first row, an uncircled 3 is entered to indicate that a transition to ③ state will occur as a result of the input change from 00 to 10. The uncircled 3 represents an unstable state.

When the circuit is in ② and the other input also becomes 1($I_1 I_2 = 11$), then the circuit enters ④ with output 0. This is represented as ④, 0 in the fourth row. In the second row, an uncircled 4 is entered to indicate that a transition to ④ will occur as a result of the input change from 10 to 11.

When the circuit is in ③ and the other input also becomes 1($I_1 I_2 = 11$), then the circuit enters ⑤ with output 1 because I_1 changes from 0 to 1. This is represented as ⑤, 1 in the fifth row. In the third row, an uncircled 5 is entered to indicate that a transition to ⑤ state will occur as a result of the input change from 01 to 11.

When the circuit is in ④ or ⑤ and if I_1 remains same at 1 and I_2 becomes 0, then the circuit enters a stable state 6 and its output is 0. This is represented as ⑥, 0 in the sixth row. In the fourth and fifth rows, an uncircled 6 is entered to indicate that a transition to ⑥ state will occur as a result of the input change from 11 to 10.

A circled entry and an uncircled entry can be combined to minimize the primitive state table by merge process. The outputs associated with unstable states correspond to their stable states. The minimized state table is given in Table 6.6.

Table 6.6 Minimized state table

Row	*Next state, Output*			
	$I_1 I_2 = 00$	$I_1 I_2 = 01$	$I_1 I_2 = 11$	$I_1 I_2 = 10$
1	①,0	3,0	4,0	②,1
2	1,0	③,0	⑤,1	6,0
3	1,0	3,0	④,0	⑥,0

Since there are three rows in the merged state table, a 2-bit variable can be used; 00 assigned to row 1, 01 assigned to row 2, and 10 assigned to row 3. The present state/next state and output table is given in Table 6.7. The unstable states are assigned 00/01/10 according to their stable state.

Table. 6.7 Present state/next state and output table

$Y_1 Y_2$	*Next state, Output*			
	$I_1 I_2 = 00$	$I_1 I_2 = 01$	$I_1 I_2 = 11$	$I_1 I_2 = 10$
0 0	00, 0	01, 0	10, 0	00, 1
0 1	00, 0	01, 0	01, 1	10, 0
1 0	00, 0	01, 0	10, 0	10, 0

If J-K flip-flop is used, then an excitation and output table is required, as given in Table 6.8.

Table. 6.8 Excitation and output table

$Y_1 Y_2$	$J_1 K_1, J_2 K_2, Z$			
	$I_1 I_2 = 00$	$I_1 I_2 = 01$	$I_1 I_2 = 11$	$I_1 I_2 = 10$
0 0	0×, 0×, 0	0×, 1×, 0	1×, 0×, 0	0×, 0×, 1
0 1	0×, ×1, 0	0×, ×0, 0	0×, ×0, 1	1×, ×1, 0
1 0	×1, 0×, 0	×1, 1×, 0	×0, 0×, 0	×0, 0×, 0

The excitation function $(J_1 K_1)$, $(J_2 K_2)$, and Z can be simplified using K-maps, as shown in Fig. 6.4.

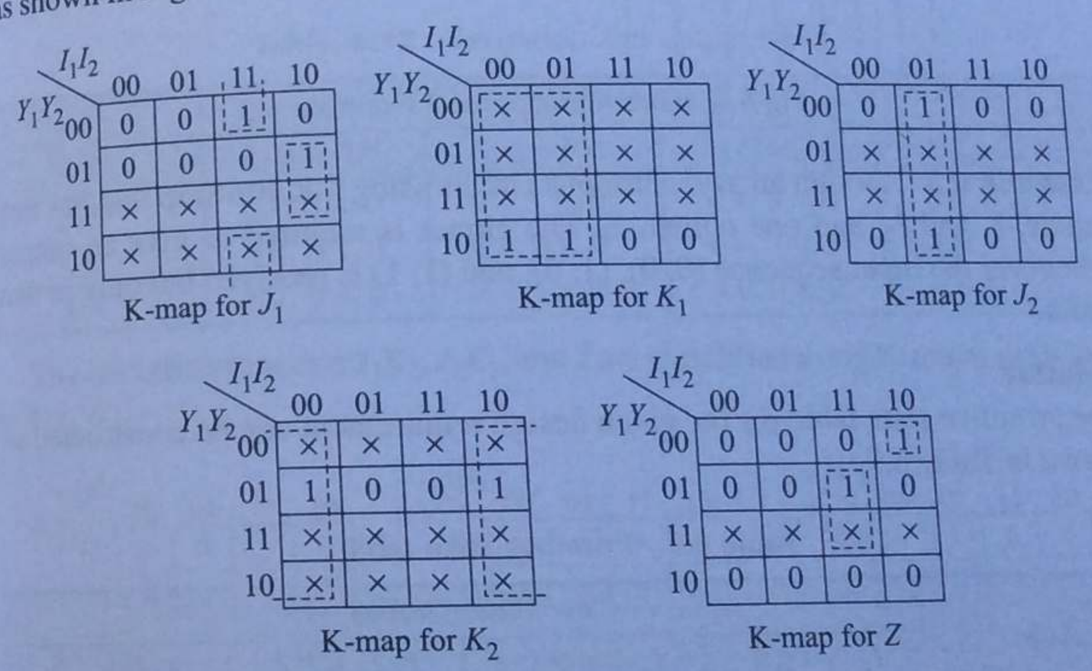


Fig. 6.4 K-map for J_1, K_1, J_2, K_2, and Z

$$J_1 = I_1 I_2 \bar{Y}_2 + I_1 \bar{I}_2 Y_2 \tag{6.3}$$

$$K_1 = \bar{I}_1 \tag{6.4}$$

$$J_2 = \bar{I}_1 I_2 \tag{6.5}$$

$$K_2 = \bar{I}_2 \tag{6.6}$$

$$Z = I_1 \bar{I}_2 \bar{Y}_1 \bar{Y}_2 + I_1 I_2 Y_2 \tag{6.7}$$

Using the above equations, the circuit diagram for the given asynchronous circuit can be drawn, as shown in Fig. 6.5.

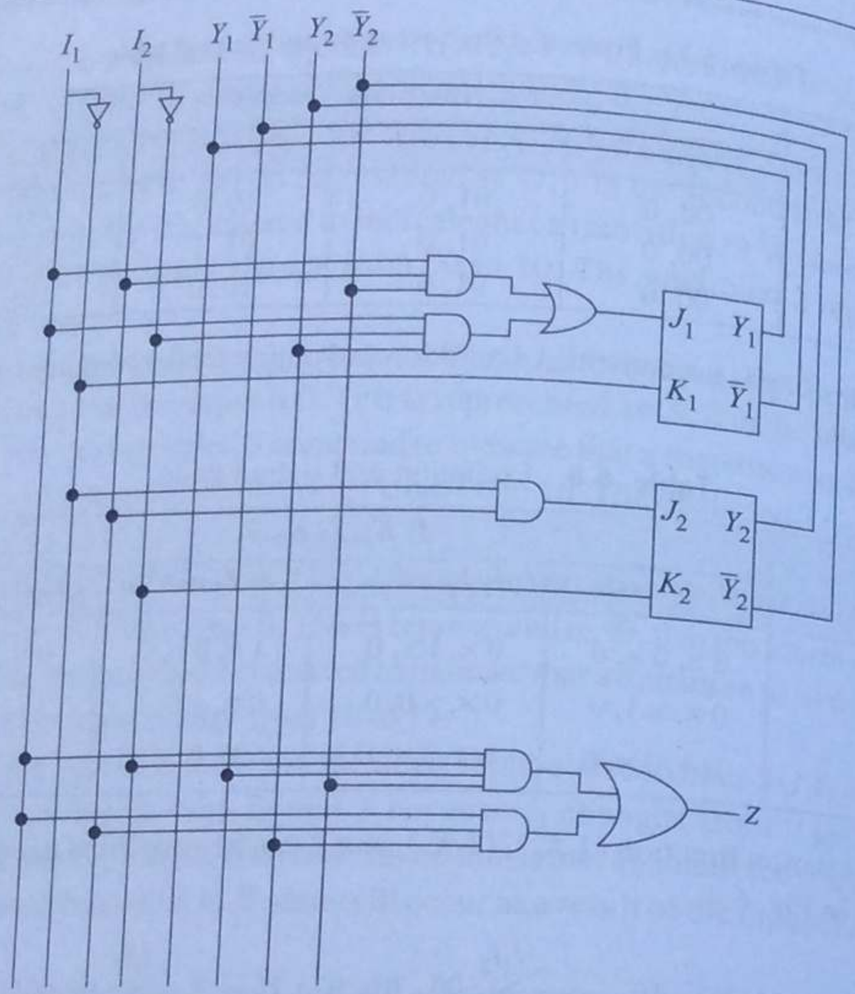


Fig 6.5 Circuit diagram of Example 6.2

Example 6.3 Design an asynchronous circuit using J-K flip-flop that has two inputs, I_1 and I_2, and one output, Z. The circuit is required to give an output whenever the input sequence (0, 0), (1, 0), and (1, 1) is received but only in that order.

Solution

The primitive state table for the given design requirement can be constructed as shown in Table 6.9.

Table 6.9 Primitive state table

Row	*Next state, Output*			
	$I_1 I_2 = 0\,0$	$I_1 I_2 = 0\,1$	$I_1 I_2 = 1\,1$	$I_1 I_2 = 1\,0$
1	①, 0	3		2
2	1		4	②, 0
3	1	③, 0	5	
4		3	④, 1	6
5		3	⑤, 0	6
6	1		5	⑥, 0

The primitive state table shown in Table 6.9 can be minimized by merging the rows, 1 to 6, as shown in Table 6.10.

Table 6.10 Minimized state table

	Next state, Output		
$I_1 I_2 = 0\,0$	$I_1 I_2 = 0\,1$	$I_1 I_2 = 1\,1$	$I_1 I_2 = 1\,0$
①, 0	3, 0	4, 1	②, 0
1, 0	③, 0	⑤, 0	⑥, 0
	3, 0	④, 1	6, 0

Since there are three rows in the merged state table, a 2-bit variable can be used, (0 0) assigned to row 1, (0 1) assigned to row 2, and (1 0) assigned to row 3, and the present state/next state and output table is given in Table 6.11. The unstable state is assigned 00/01/10 according to their stable state.

Table. 6.11 Present state/next state and output table

$Y_1 Y_2$	$Y_1 Y_2, Z$			
	$I_1 I_2 = 0\,0$	$I_1 I_2 = 0\,1$	$I_1 I_2 = 1\,1$	$I_1 I_2 = 1\,0$
0 0	00, 0	01, 0	10, 1	00, 0
0 1	00, 0	01, 0	01, 0	10, 0
1 0	00, 0	01, 0	10, 1	10, 0

If J-K flip-flop is used, then an excitation and output table is generated, as shown in Table 6.12.

Table. 6.12 Excitation and output table

$Y_1 Y_2$	$J_1 K_1, J_2 K_2, Z$			
	$I_1 I_2 = 0\,0$	$I_1 I_2 = 0\,1$	$I_1 I_2 = 1\,1$	$I_1 I_2 = 1\,0$
0 0	0×, 0×, 0	0×, 1×, 0	1×, 0×, 0	0×, 0×, 1
0 1	0×, ×1, 0	0×, 0×, 0	0×, ×0, 1	1×, ×1, 0
10	×1, 0×, 0	×1, 1×, 0	×0, 0×, 0	×0, 0×, 0

The excitation function J_1K_1, J_2K_2, and Z are simplified using K-maps, as shown in Fig. 6.6.

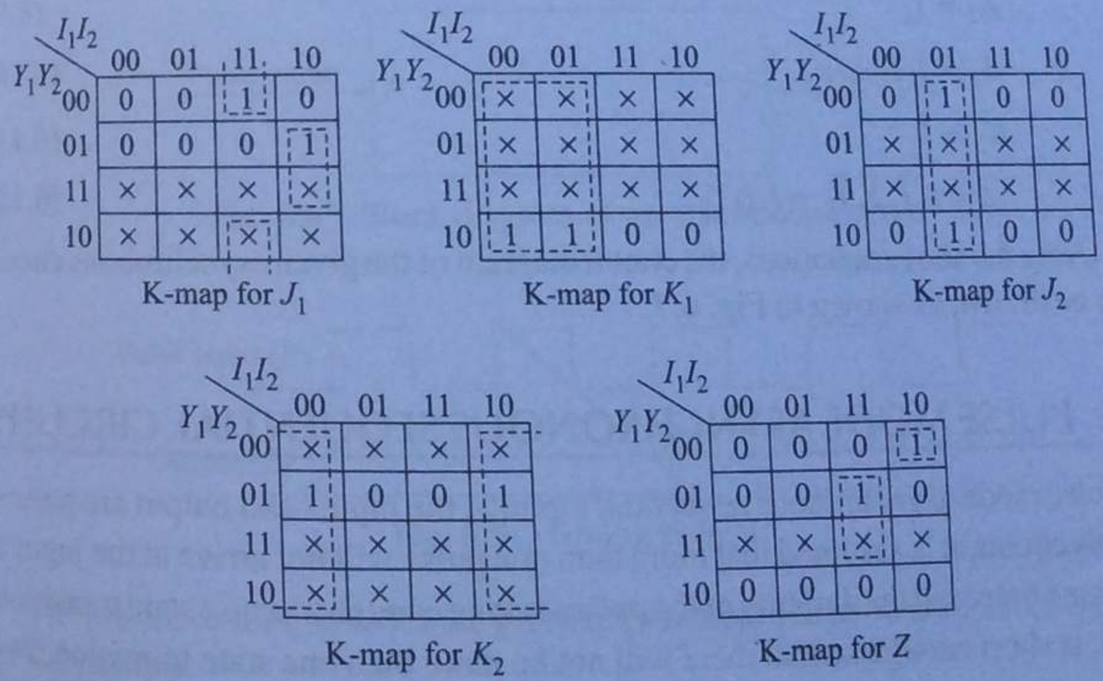


Fig. 6.6 K-map for J_1, K_1, J_2, K_2 and Z

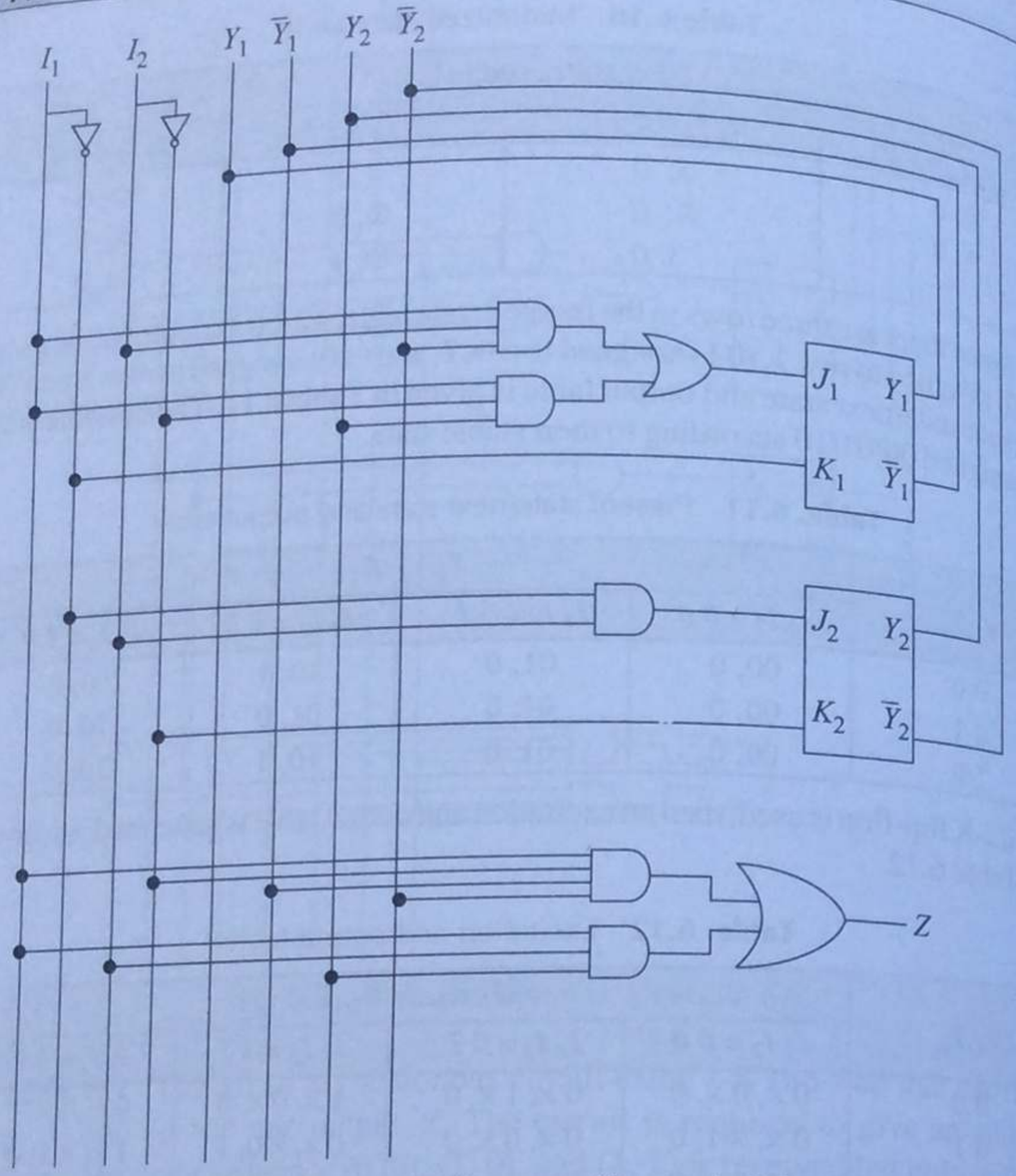


Fig. 6.7 Circuit diagram of Example 6.3

$$J_1 = I_1 I_2 \bar{Y}_2 + I_1 \bar{I}_2 Y_2 \tag{6.8}$$

$$K_1 = \bar{I}_1 \tag{6.9}$$

$$J_2 = \bar{I}_1 I_2 \tag{6.10}$$

$$K_2 = \bar{I}_2 \tag{6.11}$$

$$Z = I_1 \bar{I}_2 \bar{Y}_1 \bar{Y}_2 + I_1 I_2 Y_2 \tag{6.12}$$

Using the above equations, the circuit diagram of the given asynchronous circuit can be drawn, as shown in Fig. 6.7.

6.3 PULSE MODE ASYNCHRONOUS SEQUENTIAL CIRCUITS

In pulse mode asynchronous sequential circuits, the inputs and output are pulses. In this circuit, it is assumed that more than one pulse will not arrive at the input at the same time, and the duration of the pulses is long enough to cause state transition, and it is short enough so that there will not be more than one state transition for a single pulse.

The design procedure used for fundamental mode asynchronous sequential circuits is also applicable to pulse mode asynchronous sequential circuits. Designing a pulse mode asynchronous sequential circuits requires the following steps:

- Formulate what the circuit has to do, from the problem definition.
- Develop a primitive state table and specify the outputs that are associated with the stable state.
- Minimize the primitive state table by merge process. It is the process of identifying those primitive states that are related by input sequences and can be combined together into a single stable state, i.e. finding the states that can be put together in the same row of the excitation map. At the end of this merge process, the number of rows will be reduced.
- Assign state variables (i.e. secondary variables) to the rows of merged primitive state table and obtain present state/next state and output table. The outputs assigned to the unstable states vary according to the design requirements.
- Decide the memory element to be used and obtain the excitation and output table. Obtain the simplified expression for the excitation and output function.
- Draw the schematic circuit diagram.

Let us consider the design example of pulse mode asynchronous circuits.

Example 6.4 Design an asynchronous circuit that will output only the second pulse received and ignore other pulses.

Solution

The block diagram of the asynchronous circuit is shown in Fig. 6.8. It has a pulse input, and one output. As per the given design requirement, only the second pulse received in the pulse input should be available at the output, as shown in Fig. 6.9.

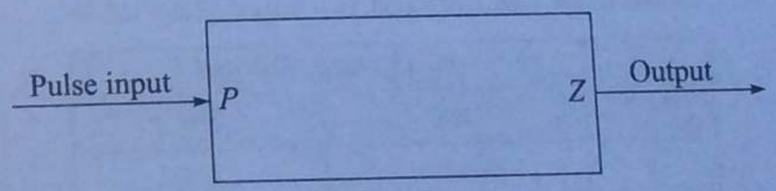


Fig. 6.8 Block diagram of asynchronous circuit

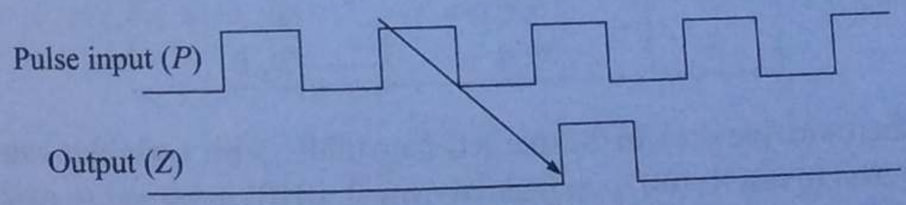


Fig. 6.9 Timing diagram

The primitive state table for the given requirement can be constructed, as shown in Table 6.13.

Table 6.13 Primitive state table

Row	Next state, Output	
	P = 0	P = 1
1	①,0	2
2	3	②,0
3	③,0	4
4		④,1
5	⑤,0	6
6	5	⑥,0

Initially, the input is 0 (i.e. $P = 0$), the circuit is in a stable state 1 and the output is 0, represented as ①, 0.

When P becomes 1, the circuit enters a stable state 2 and its output is 0. This is represented as ②, 0 in the second row. In the first row, an uncircled 2 is entered to indicate that a transition to ② state will occur as a result of the input change from 0 to 1. The uncircled 2 represents an unstable state.

When P becomes 0 and the circuit enters a stable state 3, its output is 0. This is represented as ③, 0 in the third row. In the second row, an uncircled 3 is entered to indicate that a transition to ③ state will occur as a result of the input change from 1 to 0. The uncircled 3 represents an unstable state.

When the circuit is in stable state ③ and the pulse input becomes 1, then the circuit enters a stable state ④ with output 1. This is represented as ④, 1 in the fourth row. It indicates that the second pulse is available at the output.

A circled entry and an uncircled entry can be combined to minimize the primitive state table by merge process. The primitive state table shown in Table 6.13 can be minimized by merging row 3 and row 4, as shown in Table 6.14. The stable state and the corresponding unstable states are combined as stable states, and the outputs associated with the unstable states correspond to their stable states.

Table 6.14 Minimized primitive state table

Row	Next state, Output	
	P = 0	P = 1
1	①, 0	2, 0
2	3, 0	②, 0
3	③, 0	4, 1
4	5, 0	④, 1
5	⑤, 0	⑥, 0

Since there are five rows in the merged state table, 3-bit variables can be used; (000) assigned to row 1, (001) assigned to row 2, (010) assigned to row 3, (011) assigned to row 4, and (100) assigned to row 5. The present state/next state and output table is given in Table 6.15. The unstable state is assigned 000/001/010/011/100 according to their stable state.

Table 6.15 Present state/next state and output table

$Y_1Y_2Y_3$	$Y_1Y_2Y_3$, Z	
	P = 0	P = 1
000	000, 0	001, 0
001	010, 0	001, 0
010	010, 0	011, 1
011	100, 0	011, 1
100	100, 0	100, 0

If J-K flip-flop is used, then an excitation and output table is generated, as shown in Table 6.16.

Table 6.16 Excitation and output table

$Y_1Y_2Y_3$	J_1K_1, J_2K_2, J_3K_3, Z	
	P = 0	P = 1
000	0×, 0×, 0×, 0	0×, 0×, 1×, 0
001	0×, 1×, ×1, 0	0×, 0×, ×0, 0
010	0×, ×0, 0×, 0	0×, ×0, 1×, 1
011	1×, ×1, ×1, 0	0×, ×0, ×0, 1
100	×0, 0×, 0×, 0	×0, 0×, 0×, 0

The excitation function J_1K_1, J_2K_2, J_3K_3, and Z can be simplified using the K-maps, as shown in Fig. 6.10.

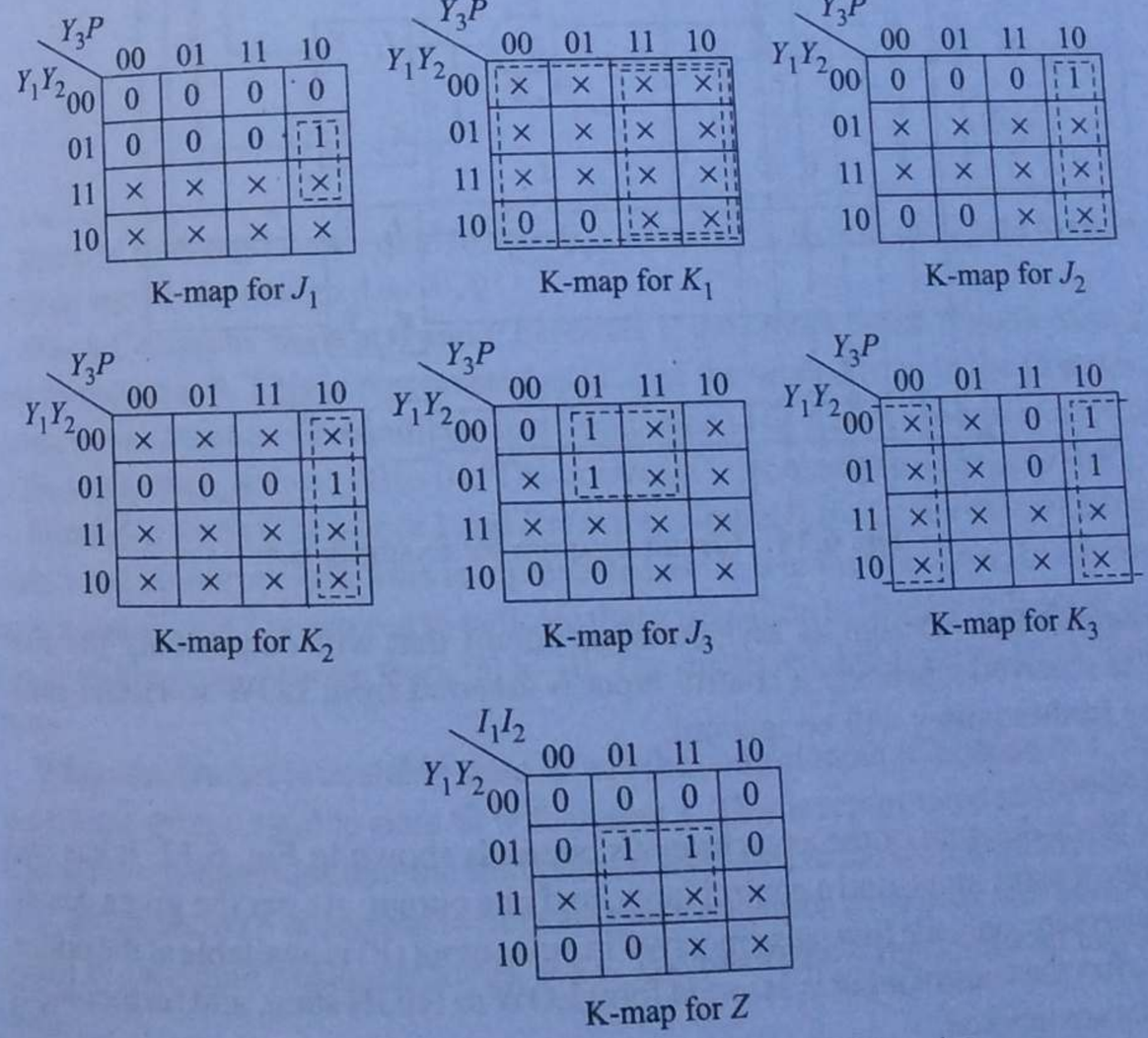


Fig. 6.10 K-map for J_1, K_1, J_2, K_2 and Z

$$J_1 = Y_2 Y_3 \overline{P} \quad (6.13)$$
$$K_1 = Y_3 \quad (6.14)$$
$$J_2 = Y_3 \overline{P} \quad (6.15)$$
$$K_2 = Y_3 \overline{P} \quad (6.16)$$
$$J_3 = \overline{Y}_1 P \quad (6.17)$$
$$K_3 = \overline{P} \quad (6.18)$$
$$Z = Y_2 P \quad (6.19)$$

Using the above equations, the circuit diagram of the given asynchronous circuit can be drawn, as shown in Fig. 6.11.

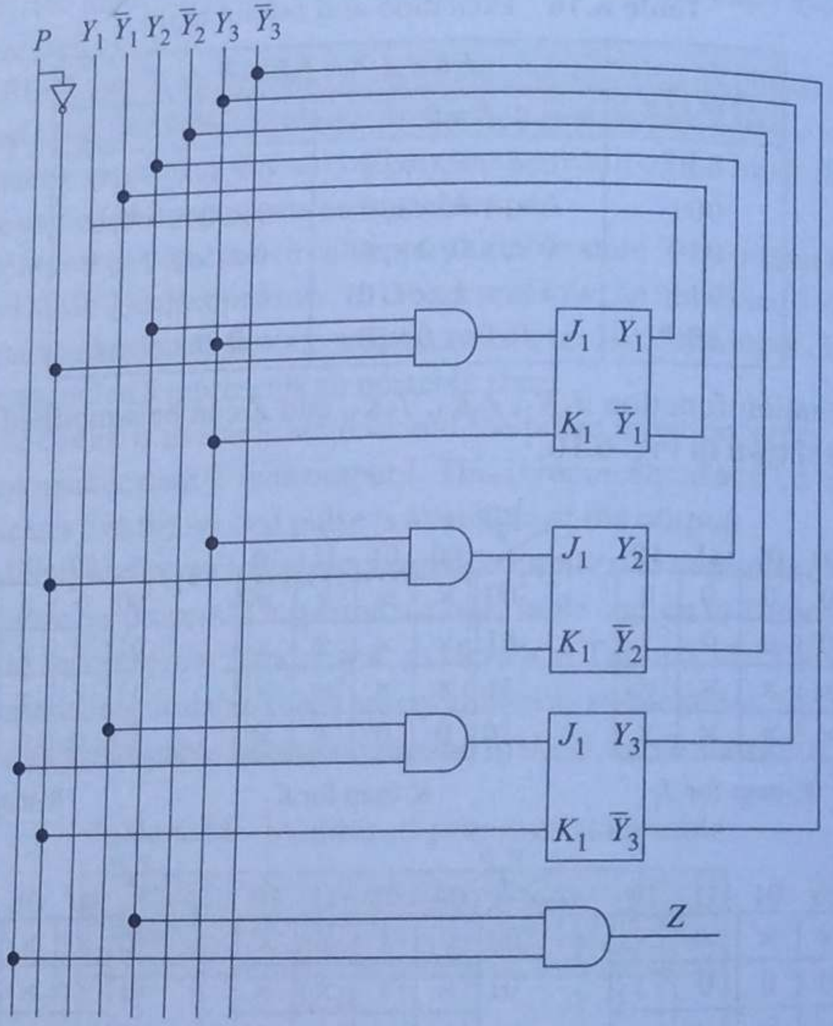


Fig. 6.11 Circuit diagram for Example 6.4

Example 6.5 Design as asynchronous circuit that will output only the first pulse received whenever a control input is asserted from LOW to HIGH state. Any further pulses will be ignored.

Solution

The block diagram of the asynchronous circuit is shown in Fig. 6.12. It has two inputs: a pulse input, and a control input, and one output. As per the given design requirement, only the first pulse received in pulse input (P) is available at the output (Z), after the control input is asserted form LOW to HIGH state, and further input pulses are ignored.

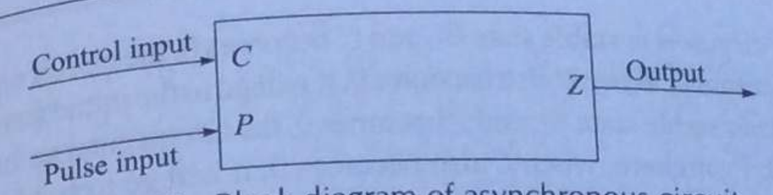


Fig. 6.12 Block diagram of asynchronous circuit

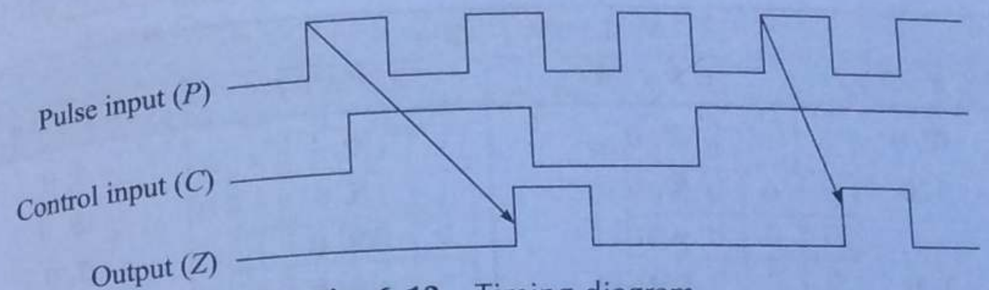


Fig. 6.13 Timing diagram

The primitive state table for the given requirement can be constructed, as shown in Table 6.17.

Table 6.17 Primitive state table

Row	*Next state, Output*			
	CP = 0 0	*CP = 0 1*	*CP = 1 1*	*CP = 1 0*
1	①, 0	2		3
2	1	②, 0	4	
3	1		5	③, 0
4		2	④, 1	3
5		6	⑤, 0	7
6	1	⑥, 0	5	
7	1		8	⑦, 0
8		2	⑧, 0	7

Initially both inputs are equal to zero (i.e. $C = P = 0$), the circuit is said to be in a stable state 1, represented as ①, 0 .

When C remains same at 0 and P becomes 1, the circuit enters a stable state 2 and its output is 0. This is represented as ②, 0 in the second row. In the first row, an uncircled 2 is entered to indicate that a transition to ② state will occur as a result of the input change from 00 to 01. The uncircled 2 represents an unstable state.

Similarly, when C becomes 1 and P remains same at 0, the circuit enters a stable state 3 and its output is 0. This is represented as ③, 0 in the third row. In the first row, an uncircled 3 is entered to indicate that a transition to ③ state will occur as a result of the input change from 00 to 10. The uncircled 3 represents an unstable state.

When the circuit is in stable state ② and the control input (C) becomes 1, then the circuit enters a stable state ④ with output 1. This is represented as ④, 1 in the fourth row. It indicates that the first pulse is available at the output. In the second row, an uncircled 4 is entered to indicate that a transition to ④ state will occur as a result of the input change from 01 to 11. The uncircled 4 represents an unstable state.

When the circuit is in stable state ④, and C becomes 0, the circuit enters a stable state ⑤ with output 0. When P also becomes 0, it will go to the initial state. Similarly, if the circuit is in stable state ④, and P becomes 0, the circuit enters a stable state ⑥ with output 0. From here, when C also becomes 0, it will go to stable state 1.

Table 6.18 Minimized primitive state table

Next state, Output			
CP=00	***CP=01***	***CP=11***	***CP=10***
①, 0	②, 0	④, 1	3, 0
1, 0	⑥, 0	5, 0	③, 0
	6, 0	⑤, 0	7, 0
1, 0	2, 0	⑧, 0	⑦, 0

The primitive state table shown in Table 6.17 can be minimized by merging rows 1, 2, and 4; rows 3, 6; and rows 5, 7, and 8, as shown in Table 6.18. The stable state and the corresponding unstable states are combined as the stable state, and the outputs associated with the unstable states correspond to their stable states.

Table 6.19 PS/NS and output table

Y_1Y_2	$Y_1\,Y_2, Z$			
	CP=00	***CP=01***	***CP=11***	***CP=10***
0 0	00, 0	00, 0	00, 1	10, 0
0 1	00, 0	01, 0	10, 0	10, 0
1 0	00, 0	01, 0	10, 0	11, 0
1 1	00, 0	00,0	11,0	11,0

If J-K flip-flop is used, then an excitation and output table is generated, as shown in Table 6.20.

Table. 6.20 Excitation and output table

Y_1Y_2	$J_1\,K_1, J_2\,K_2, Z$			
	CP=00	***CP=01***	***CP=11***	***CP=10***
0 0	0×, 0×, 0	0×, 0×, 0	0×, 0×, 1	1×, 0×, 0
0 1	0×, ×1, 0	0×, ×0, 0	1×, ×1, 0	1×, ×1, 0
1 0	×1, 0×, 0	×1, 1×, 0	×0, 0×, 0	×0, 1×, 0
1 1	×1, ×1, 0	×1, ×1, 0	×0, ×0, 0	×0, ×0, 0

The excitation function J_1K_1, J_2K_2, and Z can be simplified using K-maps, as shown in Fig. 6.14.

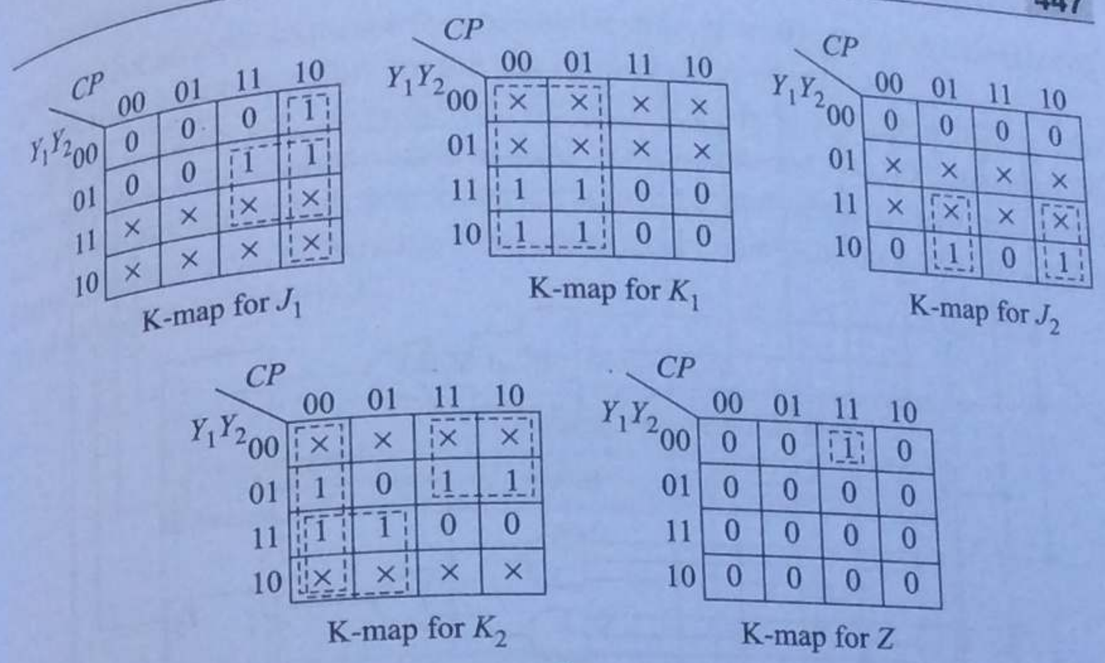


Fig. 6.14 K-map for J_1, K_1, J_2, K_2, and Z

$$J_1 = C\overline{P} + CY_2 \tag{6.20}$$

$$K_1 = \overline{C} \tag{6.21}$$

$$J_2 = C\overline{P}\,Y_1 + \overline{C}\,PY_1 \tag{6.22}$$

$$K_2 = C\overline{Y}_1 + \overline{C}\,Y_1 + \overline{C}\,\overline{P} \tag{6.23}$$

$$Z = CP\overline{Y}_1\,\overline{Y}_2 \tag{6.24}$$

Using the above equations, the circuit diagram of the given asynchronous circuit can be drawn, as shown in Fig. 6.15.

6.4 INCOMPLETELY SPECIFIED STATE MACHINES

In sequential circuits, some of the states are not specified. Such sequential circuits are known as incompletely specified state machines. In sequential circuits, not all combinations of states and inputs are possible. For example, when a machine is in a state C, it will never receive a '1' as input and consequently, the corresponding transition and output map may be left unspecified, as shown in Table 6.21, by a dash in row C and column $I = 1$.

Table 6.21 State table

Present state	***Next state and output***	
	I=0	***I=1***
A	*B*,0	*D*,1
B	*C*, 0	*A*,0
C	A,0	—
D	*C*,0	*B*,0

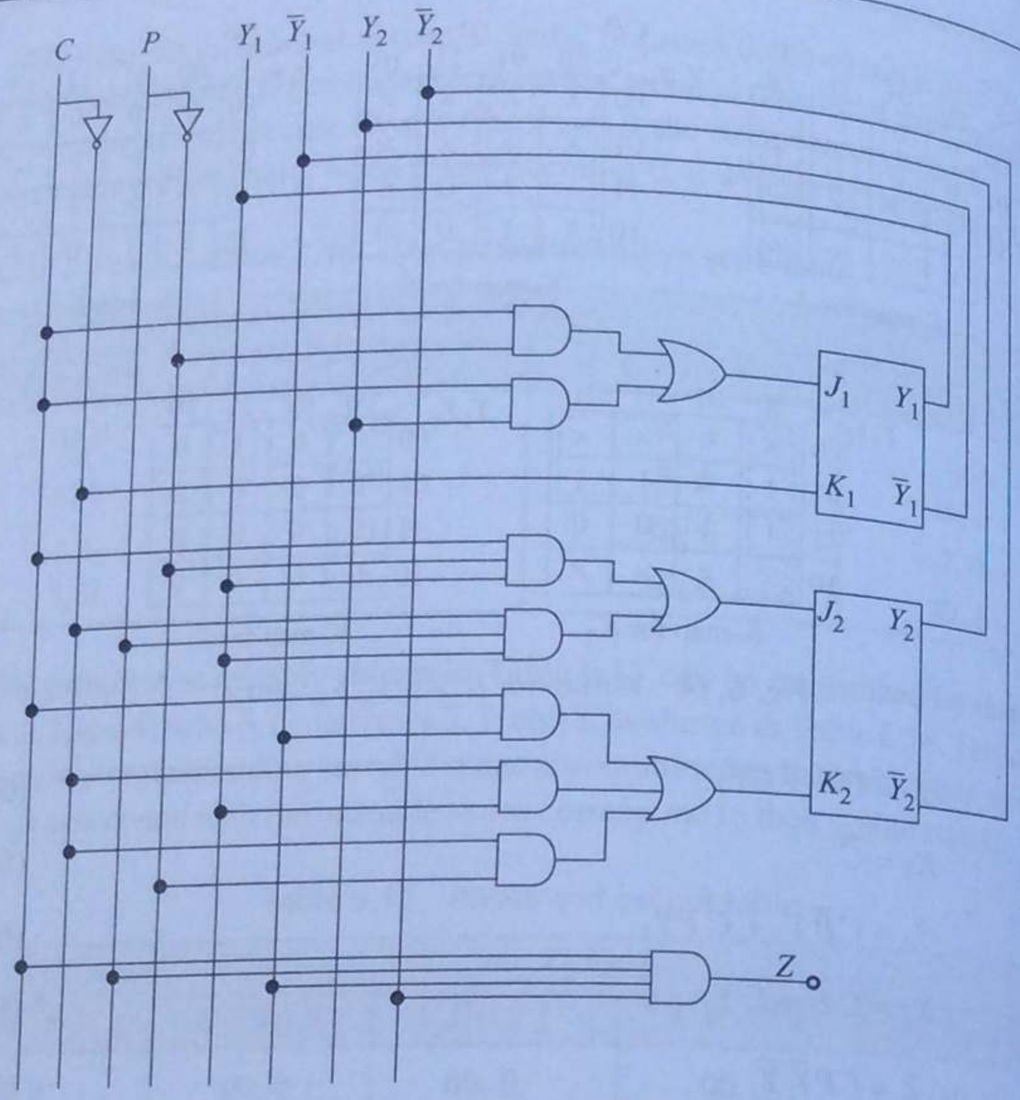


Fig. 6.15 Circuit diagram

In some sequential circuits, the state transitions are completely defined but for some combinations of states and inputs, the output values may not be critical and thus, left unspecified. For example, when a machine is in a state *B*, it will receive a '0' as input and consequently, the corresponding transition *C* occurs, but the output values are not critical and thus, left unspecified, as shown in Table 6.22, by a dash in row *B* and column $I = 0$.

Table 6.22 State table

Present state	*Next state and output*	
	I=0	*I=1*
A	*B*, 0	*D*, 1
B	*C*, –	*B*, 0
C	*A*, 0	*D*, 0
D	*C*, 0	*B*, 0

When a state transition is not specified, the behaviour of the machine may become unpredictable. To avoid such a situation, it is assumed that when the machine is in any of its possible starting states, the input sequences are applied in such a way that no unspecified state is encountered except at the final step. The output need not be

specified for an input sequence for a particular state; however, the next states should be specified except possibly for the last symbol of the sequence.

When outputs are unspecified, we may specify them according to our convenience. If the dashed output in Table 6.22 is replaced with '0', then states *B* and *D* become equivalent. The states are said to be equivalent, if their next states and outputs are identical. Now these two states can be combined and state *D* is rippled by *B* as shown in Table 6.23.

Table 6.23 State table

Present state	*Next state and output*	
	I=0	*I=1*
A	*B*, 0	*B*, 1
B	*C*, –	*B*, 0
C	*A*, 0	*B*, 0

6.5 PROBLEMS IN ASYNCHRONOUS CIRCUITS

The asynchronous sequential circuits have three major problems: *Cycles*, *Races*, and *Hazards*.

6.5.1 Cycles

Cycles are created due to the feedback in asynchronous sequential circuits. When the input changes, it induces a feedback transition through more than one unstable state, and such a situation is called a *cycle*. The circuit goes through a unique sequence of unstable states because of an input change. When a cycle exists in a state table of an asynchronous circuit, care must be taken to ensure that the circuit terminates in a stable state. Otherwise, the circuit goes form one unstable state to another until a new change occurs in the input.

6.5.2 Races

In case of cycles, only one feedback variable is unstable at any time, due to the change in an input variable. When two or more feedback variables change value in response to a change in an input variable with unequal propagation delay, then a RACE condition exists in an asynchronous sequential circuit. Races are of two types: *critical races* and *non-critical races*. Critical races should be eliminated in a circuit, whereas non-critical races may be tolerated.

Critical races

Critical races may always go to some wrong state. It is because of different propagation delays. Identical circuits with different propagation delays may go to different wrong states. Consider the PS/NS table shown in Table 6.24.

Table 6.24 PS/NS state table

Present state (Y_1Y_2)	*Next state* (Y_1Y_2) Z			
	$I_1I_2 = 0\,0$	$I_1I_2 = 0\,1$	$I_1I_2 = 1\,1$	$I_1I_2 = 1\,0$
00	11(1)	00(3)	11(9)	11(13)
01	01(4)	11(6)	11(10)	00(15)
10	10 (2)	00(7)	11(11)	10(16)
11	10(5)	11(8)	11(12)	10(14)

Assume that the circuit is in stable state 3 (i.e. $Y_1Y_2 = 00$ corresponding to $X_1X_2 = 01$) and input I_2 changes from 1 to 0. Now the next state variable Y_1Y_2 begins to switch from 00 to 11. But due to an unequal propagation delay, one of the next state variables becomes 1 and other variables do not change from the value 0, and hence, $Y_1Y_2 = 01$. Then the circuit goes to stable state 4. If $Y_1 = Y_2 = 10$, then the circuit goes to stable state 2. Therefore, for an input change, i.e. from $I_1I_2 = 01$ to 00, depending on the relative switching speed of the next state variables Y_1Y_2, the final state will be either stable state 2 or 4. But both final states, 2 and 4, are wrong for the given input change. This situation is called a *critical race*, which must be avoided in an asynchronous circuit.

Non-critical races

A non-critical race problem may always go to a correct final stable state, after its transition through unstable states. It is also because of different propagation delays. Consider Table 6.24 and assume that the circuit is in stable state 3 (i.e. $Y_1Y_2 = 00$). Now, if the input changes from $I_1I_2 = 01$ to 11, then the next state variables (Y_1Y_2) of the circuit are supposed to switch from 00 to 11. Again, due to propagation delay, Y_1Y_2 becomes either 01 or 10. Therefore, the circuit will either go to state (10) or (11) and finally reach the stable state 12. Such a situation where the correct final stable state is reached after a transition through unstable states is called a *non-critical race*. Non-critical races can be tolerated in an asynchronous circuit.

6.5.3 Hazards

Combinational circuits used in asynchronous sequential circuits may have unequal propagation delays. Hazard is an unwanted transient, i.e. spike or glitch that occurs due to unequal propagation delays through a combinational circuit. There are two types of hazards: *static hazard* and *dynamic hazard*, as shown Fig. 6.16.

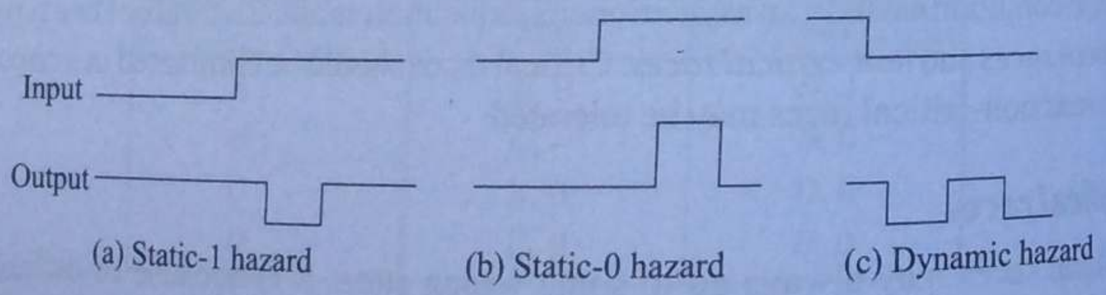


Fig. 6.16 Types of hazards

Static hazard is a condition which results in a single momentary incorrect output due to the change in an input variable when the output is expected to remain in the same state. Static hazard is of two types: *Static-1 hazard* and *Static-0* hazard.

If the output momentarily goes to state 0 due to a change in input, when the output is expected to remain in state 1 as per the steady state analysis, such hazard is known as static-1 hazard, as shown in Fig. 6.16(a).

Similarly, if the output momentarily goes to state 1 due to change in input, when the output is expected to remain in state 0 as per the steady state analysis, such hazard is known as static-0 hazard, as shown in Fig. 6.16(b).

Consider the K-map for the function $Y = \Sigma(0, 1, 2, 6)$ as shown in Fig. 6.17, and the corresponding circuit as shown in Fig. 6.18.

$$Y = B\overline{C} + \overline{A}\,\overline{B} \tag{6.25}$$

AB / C	00	01	11	10
0	1	1	1	0
1	1	0	0	0

Fig. 6.17 $Y = \Sigma(0, 1, 2, 6)$

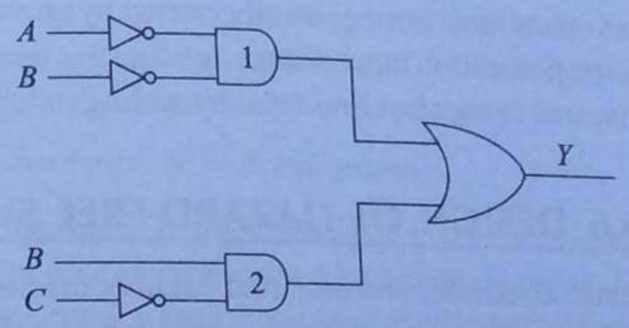


Fig. 6.18 Circuit diagram

When the input $ABC = 000$, the output $Y = 1$ due to the HIGH output of AND_1 gate. Now, when the input ABC changes to 010, Y should remain in the 1 state due to the HIGH output of AND_2 gate. Due to a change in the value of input variable B, switching of high output from AND_1 to AND_2 gate takes place and hence, Y is supposed to remain in HIGH state. But, due to an unequal propagation delay, if the upper AND gate output changes to 0 shortly before the lower AND gate output becomes 1, then during this brief period, $Y = 0$ momentarily. This situation is called a *static hazard*.

A static hazard can be removed by forming a group of adjacent cells, as shown in Fig. 6.19. This leads to a redundant grouping that overlaps both AB and BC grouping. The redundant term is $\overline{A}\,\overline{C}$ and the modified circuit is shown in Fig. 6.20. Now, when the input (ABC) changes from 000 to 010, the output will remain at 1 state (for both 010 and 000 inputs) due to the HIGH output at the AND_3 gate.

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

AB / C	00	01	11	10
0	1	1	1	0
1	1	0	0	0

Fig. 6.19 $Y = \Sigma(0, 1, 2, 6)$

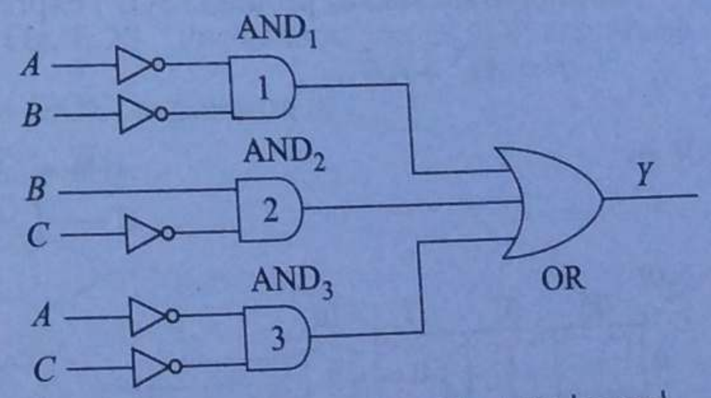


Fig. 6.20 Logic circuit with static hazard

Dynamic hazard

Multiple glitch situations at the output due a change from 0 to 1 (or form 1 to 0) is known as *dynamic hazard*. It can be removed by forming a group of adjacent cells, as in static hazard.

$$Y = \overline{A}\,\overline{C} + B\overline{C} + \overline{A}\,\overline{B} \tag{6.26}$$

Essential hazard

It 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 operational 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. Essential hazards cannot be eliminated by adding redundant gates, same as static hazards.

6.6 DESIGN OF HAZARD-FREE SWITCHING CIRCUITS

Static, dynamic, and essential hazards discussed in the previous section can be eliminated by using different procedures as discussed below.

6.6.1 Static Hazards Elimination

As discussed in the previous section, hazard is an unwanted transient, i.e. spike or glitch that occurs due to unequal propagation delays through a combinational circuit. Static hazard is a condition which results in a single momentary incorrect output due to change in an input variable, when the output is expected to remain in the same state. Such hazards occur whenever there exists a pair of adjacent input combinations that produces the same output.

Therefore, to design a static hazard-free switching circuit, we need to consider all adjacent input combinations of every pair of adjacent 1 cells and every pair of adjacent 0 cells in the K-map of a switching function at least one sub-cube.

Consider the K-map for the function, $Y(A, B, C) = \Sigma(0, 2, 4, 5)$, as shown in Fig. 6.21. The corresponding circuits for the simplified sum of products (SOP) and the product of sums (POS) expression are shown in Figs 6.22 and 6.23, respectively.

The simplified sum of product (SOP) expression is

$$F = \overline{A}\overline{C} + A\overline{B} \tag{6.27}$$

AB / C	00	01	11	10
0	1	1	0	1
1	0	0	0	1

Fig. 6.21 K-map with hazard

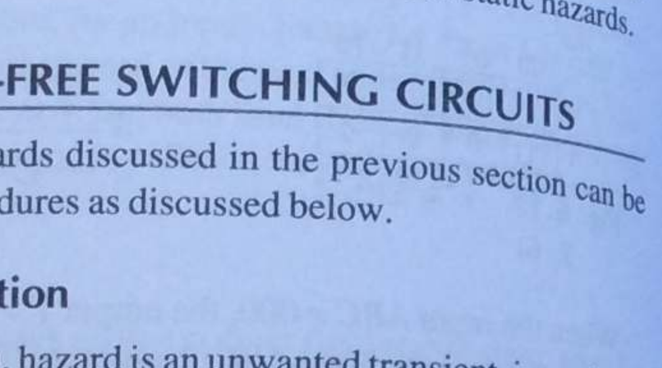


Fig. 6.22 Implementation of SOP expression

In the circuit (Fig. 6.23), when the input (ABC) changes from 100 to 000 or from 000 to 111, the output may momentarily go to 0 state due to the unequal propagation delay in the AND gates, rather than remaining constant at 1. As a result, static-1 hazard is present.

The simplified product of sums (POS) expression is

$$F = (A + \overline{C})(\overline{A} + \overline{B}) \tag{6.28}$$

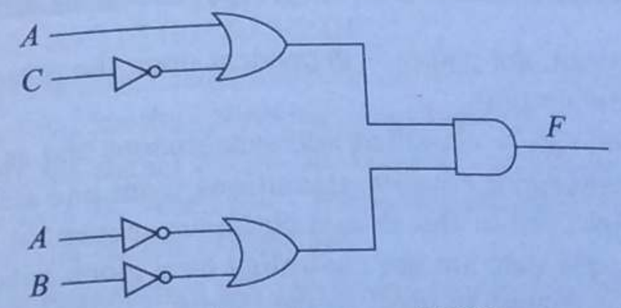


Fig. 6.23 Implementation of POS expression

In the circuit (Fig. 6.23), when the input changes from 111 to 011 or from 011 to 111, the output may momentarily go to 1 state due to the unequal propagation delay in the OR gates, rather than remaining constant at 0. As a result, static-0 hazard is present.

To make the switching circuits shown in Figs 6.22 and 6.23 hazard-free, form at least one sub-cube pair of adjacent 1 cells or adjacent 0 cells. To remove the static-1 hazard present in Fig. 6.22, form at least one sub-cube pair of adjacent 1 cells, as shown in Fig. 6.24 by dotted sub-cubes. To remove the static-0 hazard present in Fig. 6.23, form at least one sub-cube pair of adjacent 0 cells, as shown in Fig. 6.24 by dotted sub-cubes.

The simplified sum of product (SOP) expression is

$$F = \overline{A}\,\overline{C} + A\overline{B} + \overline{B}\,\overline{C} \tag{6.29}$$

AB / C	00	01	11	10
0	1	1	0	1
1	0	0	0	1

Fig. 6.24 K-map for hazard-free circuit

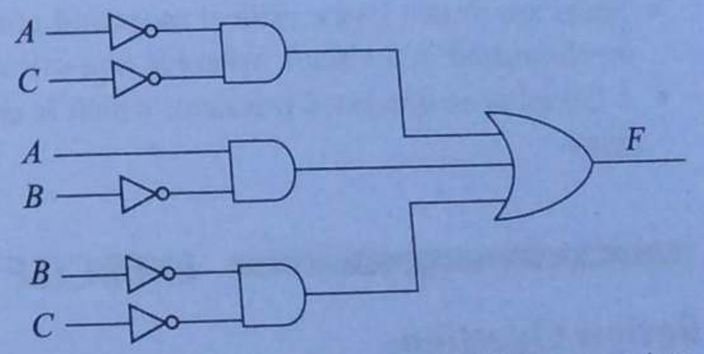


Fig. 6.25 Implementation of SOP expression

The simplified product of sum (POS) expression is

$$F = (A + \overline{C})(\overline{A} + \overline{B})(\overline{C} + \overline{B}) \tag{6.30}$$

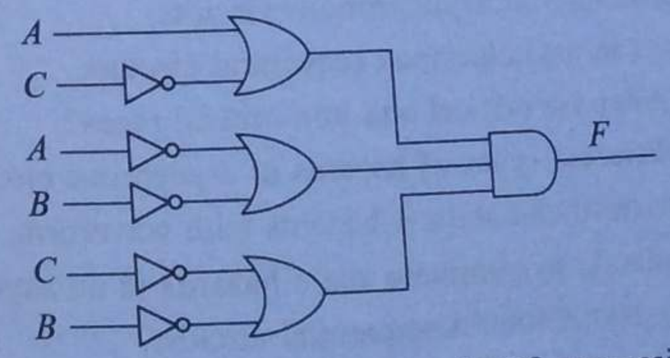


Fig. 6.26 Implementation of POS expression

6.6.2 Dynamic Hazards Elimination

Dynamic hazards can also be eliminated, following the similar procedure as that of static hazard elimination, by covering every pair of 1 cells or every pair of 0 cells in the K-map by at least one sub-cube.

SUMMARY

- In a sequential circuit, the output is dependent upon the present inputs as well as the past inputs and outputs.
- A sequential circuit can be classified into synchronous and asynchronous circuits.
- In synchronous sequential circuits, transitions from one state to next state are controlled by a clock pulse. But in asynchronous sequential circuits, transitions from one state to next state are not controlled by a clock pulse; transitions occur whenever there is a change in input to the circuit.
- Asynchronous sequential circuits can be classified into two types: fundamental mode and pulse mode.
- In fundamental mode circuits, inputs and outputs are represented by levels, whereas in pulse mode asynchronous sequential circuits, inputs and outputs are represented by pulses.
- In sequential circuits, some of the states are not specified. Such sequential circuits are known as incompletely specified state machines.
- The asynchronous sequential circuits have three major problems, namely cycles, races, and hazards.
- The problem of cycle is created due to the feedback used in the asynchronous sequential circuits.
- When two or more feedback variables change value in response to a change in an input variable, then a RACE condition exists in an asynchronous sequential circuit.
- Races are of two types: critical races and non-critical races. Critical races should be eliminated in a circuit, whereas non-critical races may be tolerated.
- A hazard is an unwanted transient, which is created due to an unequal propagation delay.

EXERCISE

Review Questions

1. Differentiate synchronous and asynchronous sequential circuits.
2. Explain the fundamental and pulse mode asynchronous sequential circuits.
3. Describe the design procedure for asynchronous sequential circuits.
4. Explain the problems in asynchronous circuits.
5. Describe cycles in asynchronous sequential circuits.
6. What do you mean by critical and non-critical races?
7. What are the different types of hazards in asynchrous circuits?
8. Differentiate static-0 and static-1 hazards with waveform.
9. Explain the methods to eliminate static hazards in an asynchronous circuit.
10. Define races in asynchronous sequential circuits.

11. Describe incompletely specified state machines with example. Give the general model of asynchronous sequential circuits.
12. Design a clocked synchronous sequential circuit that checks a serial data line for odd parity. The circuit has two inputs, SYNC and DATA. Output ERROR using D flip-flops.
13. Design a sequential circuit using D flip-flops to find each of the following sequence in a serial input signal (I). Whenever the sequence is found, the output (Z) should be 1. (a) 110 (b) 0011 (c)1010
14. Design an asynchronous sequential circuit with two inputs, I_1 and I_2, and two outputs, Z_1 and Z_2, for the following specifications:
    When $I_1I_2 = 00$, the output $Z_1Z_2 = 00$. When $I_1 = 0$ and the I_2 changes form 0 to 1 the output $Z_1Z_2 = 01$. When $I_2 = 1$ and I_1 changes form 0 to 1, the output $Z_1Z_2 = 10$. Otherwise, the output is constant.
15. Obtain a static hazard-free asynchronous circuit for the following switching function

$$f = E(0, 2, 4, 5, 8, 10, 14)$$

Chapter 7

ALGORITHMIC STATE MACHINES

Chapter Outline

- Basic concepts of construction of ASM charts for sequential circuits
- Methods of realization of ASM charts
- Use of RTL notation for describing sequential circuits
- Realization of data and control sections from RTL code
- Importance of VHDL and its basic constructs

7.1 INTRODUCTION

Any digital system design can be divided into two distinct parts. The part that deals with data processing operations is referred to as *data section* and the other that deals with control signals generation and supervision is referred to as the *control section.*

The data section and the control section of a digital system is shown in Fig. 7.1. The data section manipulates the data in the registers according to the system's requirements, while the control logic generates properly sequenced commands to the data section.

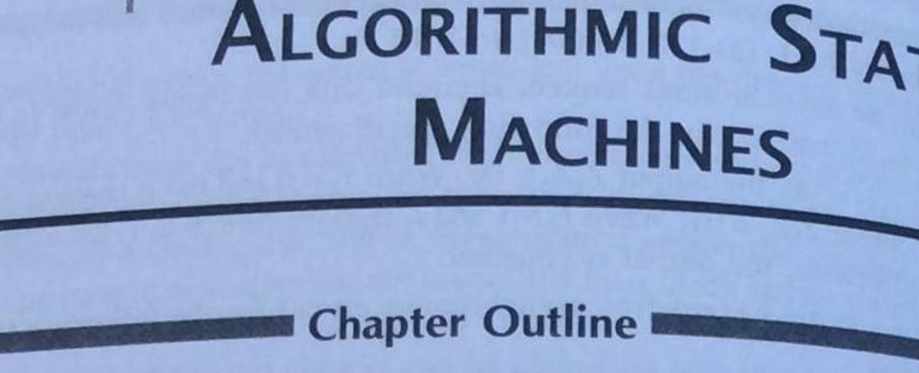


Fig. 7.1 Data section and control section of a digital system

The control section is a sequential circuit which generates control signals (commands) for data processing by considering the status signals and external input signals at any given time. Thus, depending upon the status and external input signals, the sequential circuit goes to the next state to initiate further operations.

The control section sequence plays an important role in digital design. We need some good means for specifying its hardware algorithm. The notation should help the designer in expressing abstract algorithms and implementing those in the hardware.

In this chapter, we discuss the various ways of representing the hardware algorithms of control sections and data sections. It includes the use of Algorithmic State Machine (ASM) charts, Register Transfer Language (RTL), and an introduction to the hardware description language like VHDL.

7.2 ALGORITHMIC STATE MACHINES

Algorithmic state machine (ASM) or simply *state machine* is another name used for a sequential circuit. These names are used when the sequential circuit is used to control a digital system that carries out a step-by-step procedure or algorithm.

A special type of flowchart is called a *state machine flowchart* or *SM chart* or *ASM chart*. The ASM chart technique uses a superior notation, especially while designing synchronous circuits. As the name suggests, all the controllers are the state machines into which the algorithms are translated. ASM chart is basically composed of three basic elements: (i) state box, (ii) decision box, and (iii) conditional box.

7.2.1 State Box

Any state in the control sequence is indicated by a state box. The state box is rectangular in shape within which the output signals, which are generated or the register operation which should be carried out in that state, are written. The name of the state and the binary assignment of the step are written at the upper left and right corner of the state box. Figure 7.2 shows a state S_0 with binary assignment 010 which generates an output $Z = 1$.

Each active edge of the clock causes a change of state from the previous state to the next state in a synchronous sequential circuit. A change in clock from 0 to 1 is active edge for positive trigger logic, and from 1 to 0 is active edge for negative trigger logic. The state changes, as shown in the state diagram (Fig. 7.3), are represented using ASM chart state boxes (Fig. 7.4).

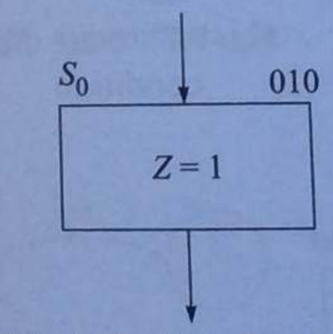


Fig. 7.2 State box of S_0

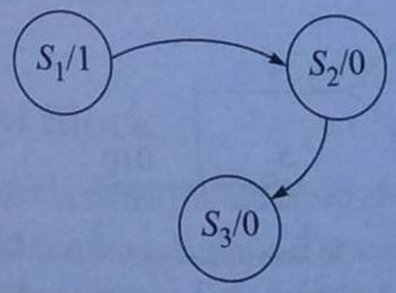


Fig. 7.3 State diagram

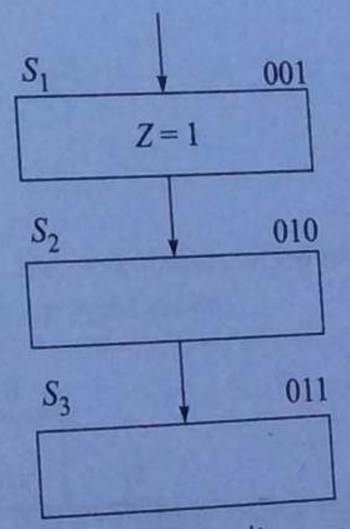


Fig. 7.4 State diagram using ASM chart state box

Figures 7.3 and 7.4 represent a state change $S_1(001)$ to $S_2(010)$ and $S_2(010)$ to $S_3(011)$, producing an output $Z = 1$ in state S_1, and $Z = 0$ in state S_2 and S_3. The state changes are represented on the time axis, as shown in Fig. 7.5.

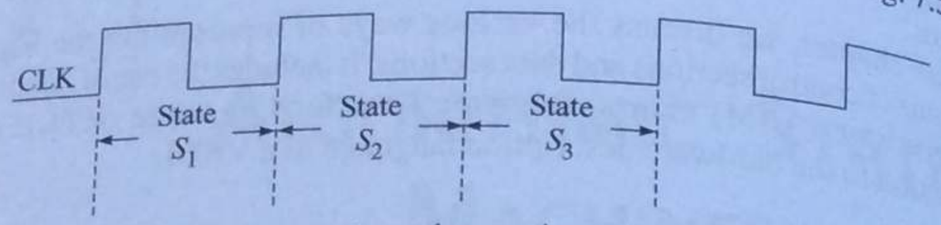


Fig. 7.5 Representation of state changes on the time axis

7.2.2 Decision Box

A decision box is a diamond-shaped box with two or more exit paths, as shown in Fig. 7.6. It expresses the conditions on the inputs which decide the state change. The input condition, which is to be tested, is written inside the box. Generally the condition written inside the box is a Boolean expression that is evaluated to determine which branch to take.

The symbol shown in Fig. 7.6 is incorporated in the ASM chart by appending it to a state box. Then the present state and the condition of the input select the corresponding exit path and thus decide the next state.

The state diagram with a condition is shown in Fig. 7.7 and its ASM chart representation with the use of decision box is shown in Fig. 7.8.

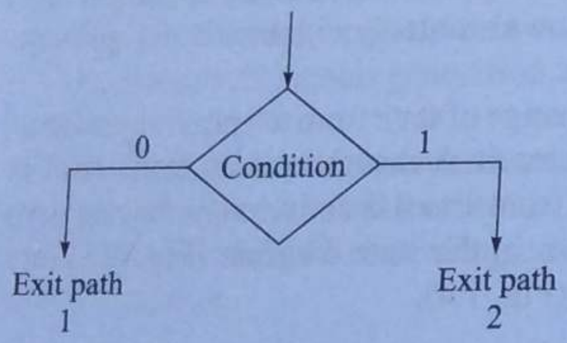


Fig. 7.6 Decision box

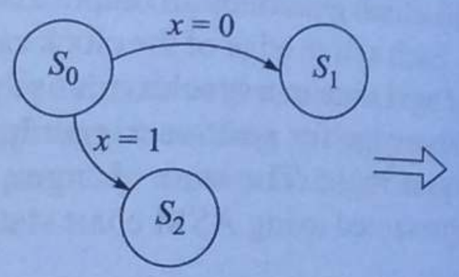


Fig. 7.7 State diagram with condition

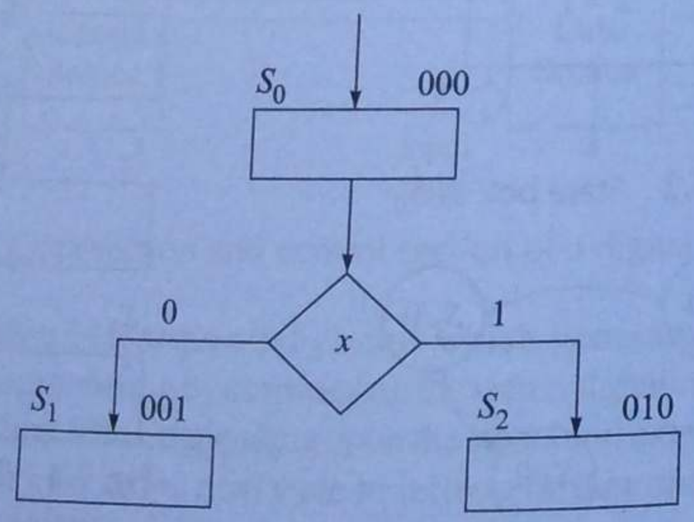


Fig. 7.8 ASM chart representation for State diagram (Fig. 7.7)

The state changes represented on the time axis for input X=0 or X=1 is shown in Fig. 7.9.

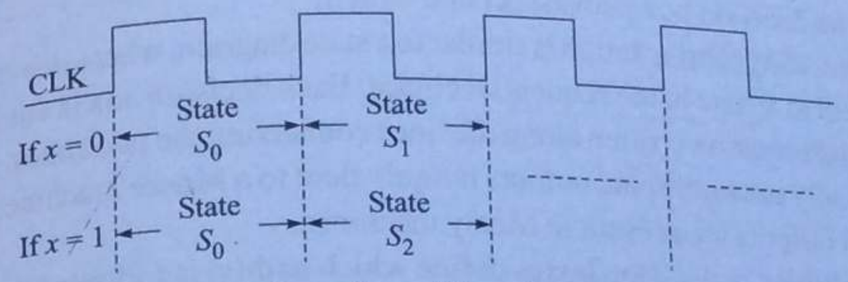


Fig. 7.9 Representation of conditional state changes on time axis

7.2.3 Conditional Box

A conditional box is an oval-shaped block in the ASM notation which is used to represent the output produced during a state only when a certain input condition is satisfied. Hence the input path to the conditional box must always come from one of the exit paths of a decision box. Figure 7.10 represents the conditional output generation using a conditional box.

The state diagram with a conditional output is shown in Fig. 7.11 and its ASM chart representation with the use of decision box and conditional box is shown in Fig. 7.12.

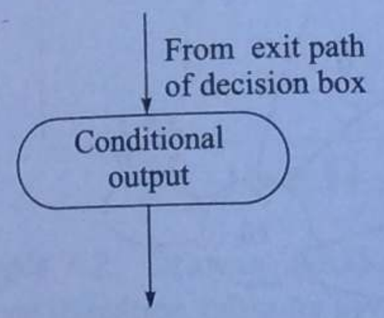


Fig. 7.10 Conditional box

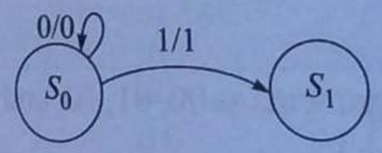


Fig. 7.11 State diagram with conditional output

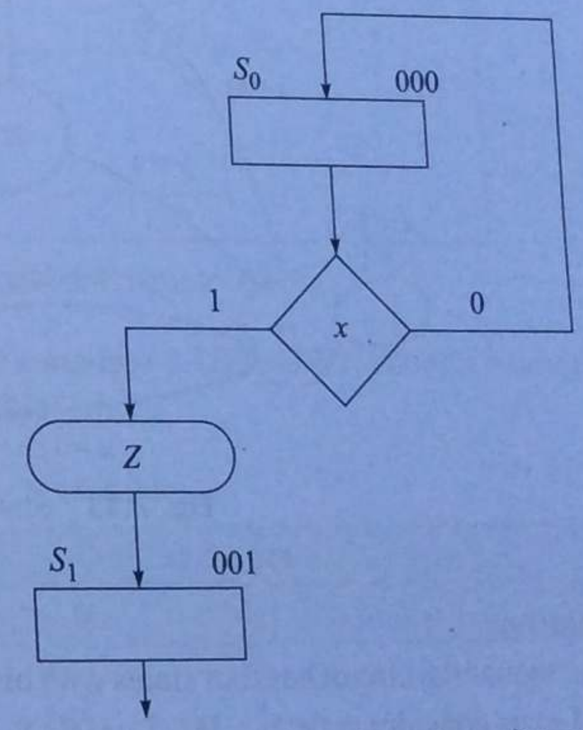


Fig. 7.12 Representation of conditional output on ASM chart

7.2.4 ASM Block

An ASM chart is constructed from an ASM block. It is a structure which represents one state, and contains diamond-shaped decision boxes and oval-shaped conditional boxes associated with it. An ASM block has one entrance path and one or more exit

paths. Unconditional outputs are the functions of parent states only, and hence are written inside the state box; whereas conditional outputs depend upon the present state and the decision box path associated with it.

Thus, the ASM chart notation is similar to a state diagram, where one state block is equivalent to a state in the sequential circuit. Each decision box is equivalent to the binary information written along the lines connecting the two states. An ASM chart with only unconditional outputs is equivalent to a Moore machine, whereas conditional outputs are present in Mealy machines.

The conditions in decision boxes define which path(s) is followed through the ASM block. The path through an ASM block from entrance to exit is referred to as a *link path*.

The following examples demonstrate how an ASM chart is drawn from the initial specifications. We will discuss the realization of ASM in the next section.

Example 7.1 Draw an equivalent ASM chart for the state diagram shown in Fig. 7.13. It has four states and two inputs, x and y.

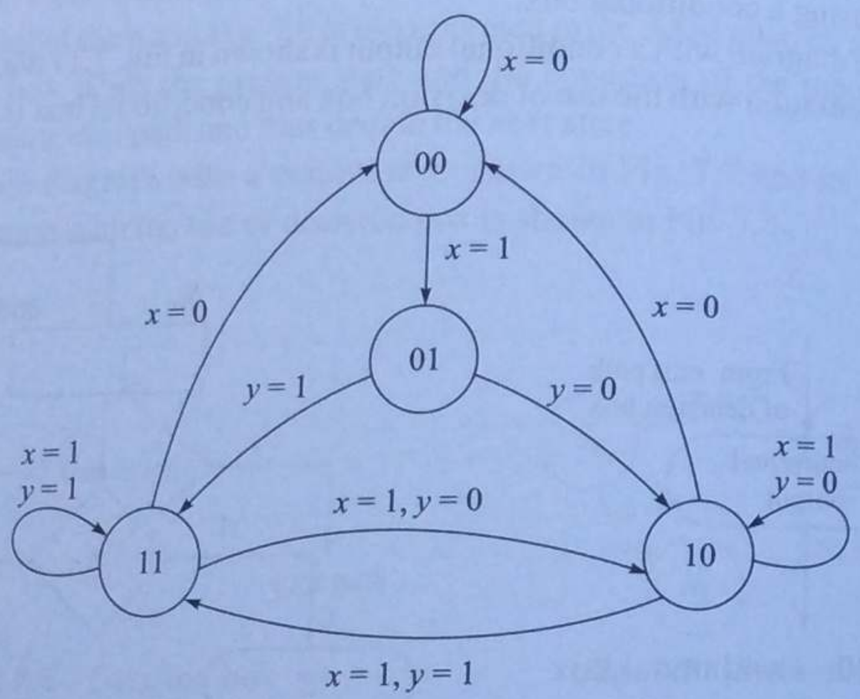


Fig. 7.13 State diagram

Solution

The sequential circuit has four states with binary assignments as 00, 01, 10, and 11.

Let us consider state $S_0 = 00$, $S_1 = 01$, $S_2 = 10$, $S_3 = 11$

Figure 7.13 shows that there are no outputs generated within the states or on the transitions, hence the ASM chart will not contain any conditional output block and all the state boxes are empty. The ASM chart will contain four state boxes and the transition between them will be through the decision boxes corresponding to the input conditions written on the transitions in the state diagram. The ASM chart for Example 7.1 is as shown in Fig. 7.14.

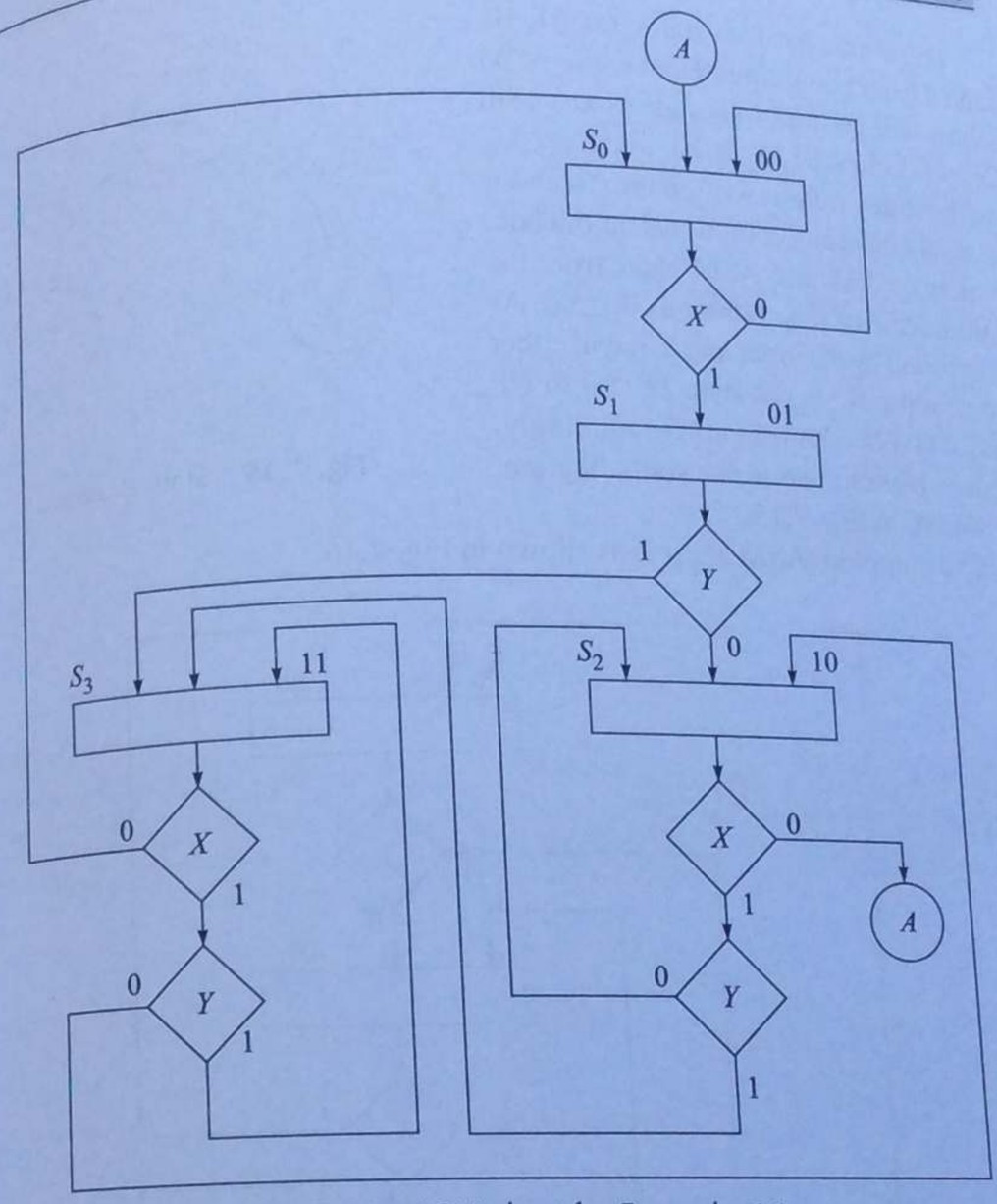


Fig. 7.14 ASM chart for Example 7.1

Example 7.2 Draw an ASM chart for a modulo-4 UP/DOWN counter having the state transition table as given in Table 7.1.

Table 7.1 State table

Present State	*Next State*	
	X = 0	*X = 1*
00	01	11
01	10	00
10	11	01
11	00	10

Solution

According to the state table, the circuit has a control input X for deciding the UP/DOWN counting mode, such that

$X = 0$ UP counting

$X = 1$ DOWN counting

The circuit has four states, 00, 01, 10, and 11, and no output. Hence, the ASM chart will contain four state boxes with $S_0 \equiv 00$, $S_1 \equiv 01$, $S_2 \equiv 10$, and $S_3 \equiv 11$, without any output written into it, and it does not have any conditional output box.

Let us draw the ASM chart from the initial state of the counter as 00 (S_0). As controlled by the input line X, it will either go to state 01 (S_1) or state 11 (S_3) in UP and DOWN counting correspondingly. This is represented in the state diagram, as shown in Fig. 7.15.

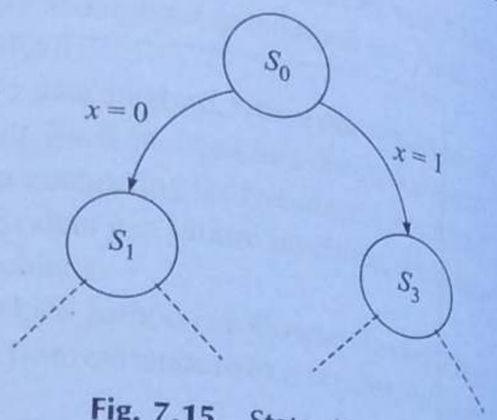


Fig. 7.15 State diagram

The complete ASM chart is as shown in Fig. 7.16.

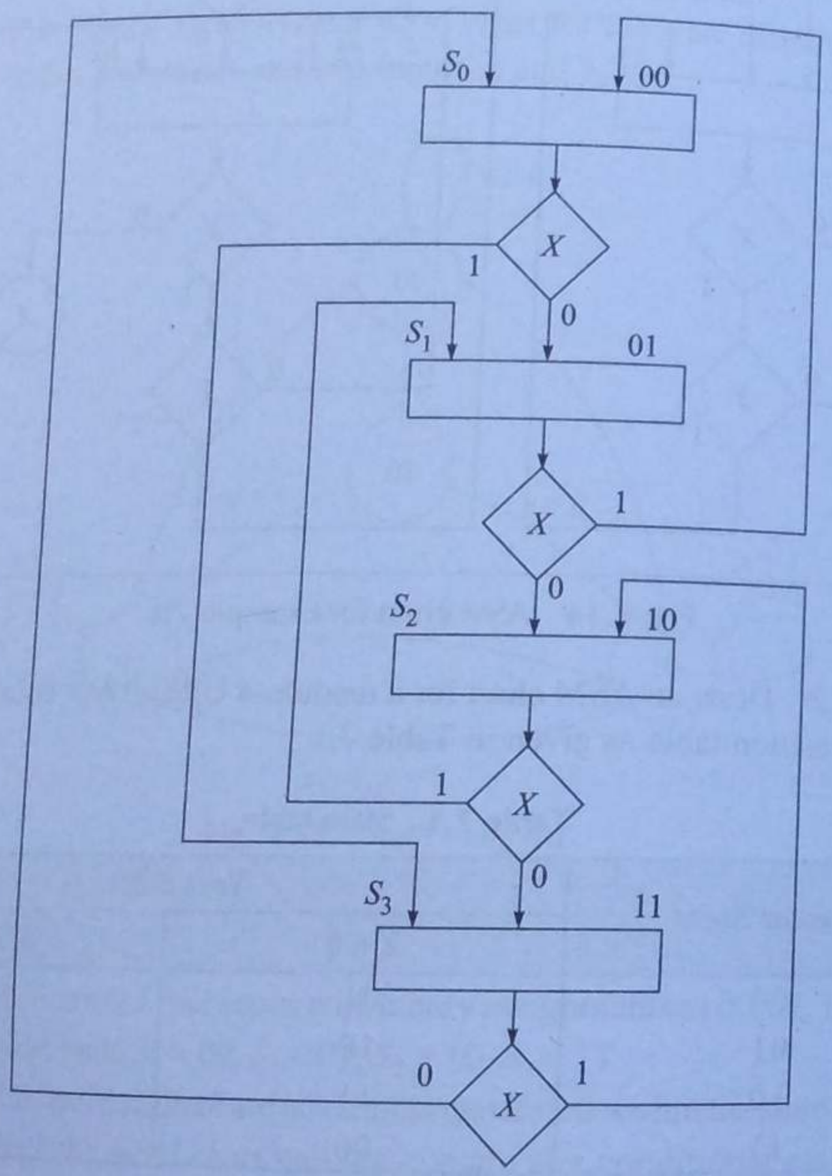


Fig. 7.16 ASM chart

Example 7.3 Develop an ASM chart for a controllable waveform generator that will output any one of the four waveforms, as shown in Fig. 7.17.

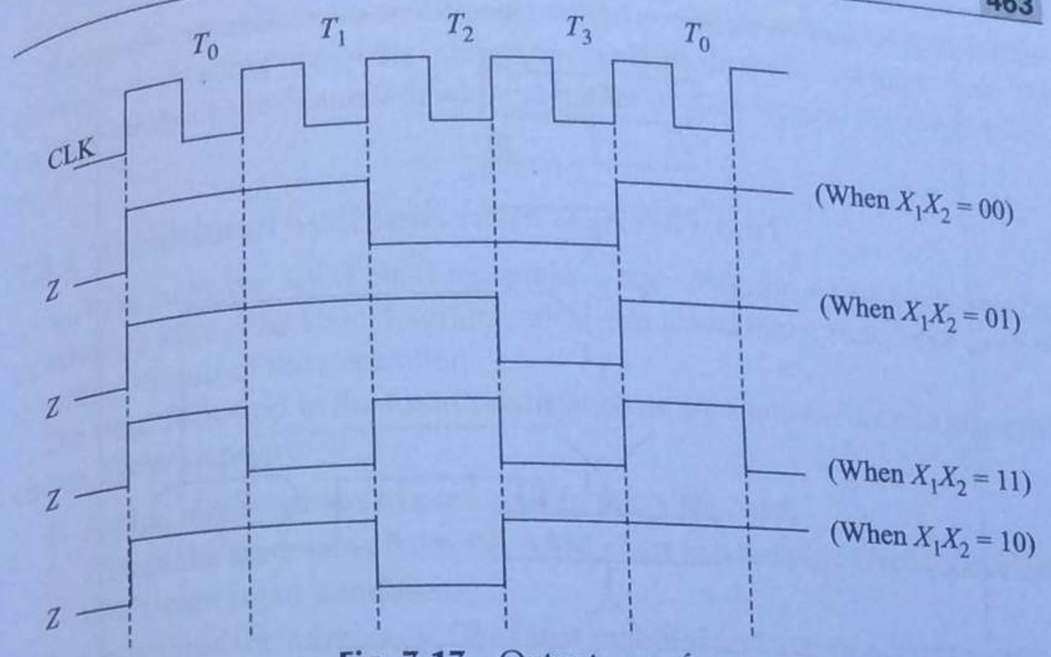


Fig. 7.17 Output waveforms

Solution

The controllable waveform generator generates one of the waveforms, as shown in Fig. 7.17, according to inputs X_1, X_2. The block diagram of a controllable waveform generator is shown in Fig. 7.18. It has two inputs X_1, X_2 and one output Z.

Fig. 7.18 Block diagram of controllable waveform generator

The period of first two waveforms is four clock cycles. The periods of third and fourth waveforms are two and three clock cycles, respectively.

The ASM chart for the waveform generator will have four states, where one state will correspond to one clock cycle of the waveform with the longest period. For each state, the output will be conditional on the values of inputs present at that time.

The ASM chart of the controllable waveform generator is shown in Fig. 7.19.

As noticeable from the waveforms, the output is at logical 1 for the first clock cycle in all the output waveforms. Hence, state T_0 contains $Z = 1$ as the unconditional output. In the second clock cycle, the output is low in waveform 3 only and high for all others. Hence in state T_1, output $Z = 1$ only if $X_1 \neq X_2 \neq 1$. Similarly, in state T_2, output $Z = 1$ only if $X_1X_2 = 01$ or $X_1X_2 = 11$. Hence, the conditional output is generated only if $X_2 = 1$. For $X_1X_2 = 10$, the fourth state is not used and the transition leads to the initial state T_0. Otherwise, the circuit goes to state T_4 for which the output $Z = 0$ (always zero). From T_3, the circuit returns to T_0 so that the waveforms may be repeated.

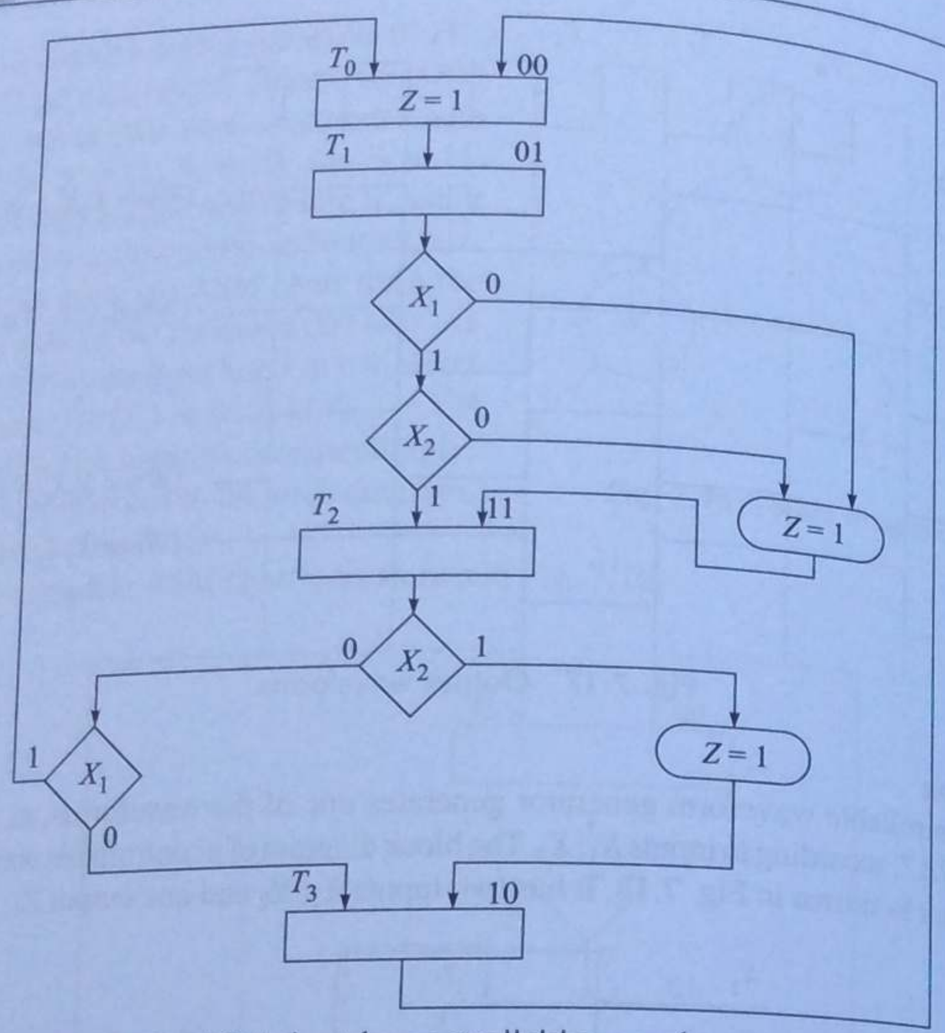


Fig. 7.19 ASM chart for controllable waveform generator

The ASM chart indicates that the combinations of input variables (X_1X_2) used for T_0, T_1, T_2 and T_3 state assignments are taken as 00, 01, 11, and 10 for convenience.

7.3 REALIZATION OF ASM CHARTS

An ASM chart provides all the information necessary to design a digital system. The requirements of the design of the data section are given inside the state and conditional boxes (outputs). The control logic is determined from the decision boxes and the state transitions.

A state generator is constructed from a given ASM chart. This state generator records the present state and generates the next state. Thus, the control section design using this state generator follows the same procedure of designing a sequential circuit, as discussed in earlier chapters. However, in some cases, this method is not suitable due to a large number of states and inputs.

The different techniques used for synthesizing the circuits are described using ASM charts. These techniques are:

1. Traditional synthesis using flip-flops and decoders
2. Multiplexer controller method for synthesis
3. One shot method of synthesis
4. ASM synthesis using ROM/PLA devices

For simple controllers, the traditional method is suitable, whereas larger designs are synthesized using one of the other three methods. In the following section, the first two methods are discussed, while the other two are beyond the scope of this book.

7.3.1 Traditional Synthesis from an ASM Chart

As we have seen, the ASM chart resembles a state diagram where each state box represents a state. The state diagram can be converted into a state table from the sequential circuit of the controller.

The steps followed in the traditional synthesis are same as that of a sequential circuit. These steps are:

1. Assign binary values to each state in the ASM chart.
2. Obtain the state table from the ASM chart in a suitable form (considering don't-care input conditions).
3. Determine the number of flip-flops required and assign a letter symbol to each.
4. Choose the type of flip-flop (Generally D flip-flop is convenient to use in case where number of flip-flops plus inputs is more, because the input functions of D flip-flop can be directly obtained without excitation table).
5. Using map or other simplification method, derive the flip-flop input functions and circuit output functions.
6. Draw the logic diagram (Circuit realization of ASM chart).

Example 7.4 Realize the ASM chart shown in Fig. 7.20 using D flip-flops and combinational logic.

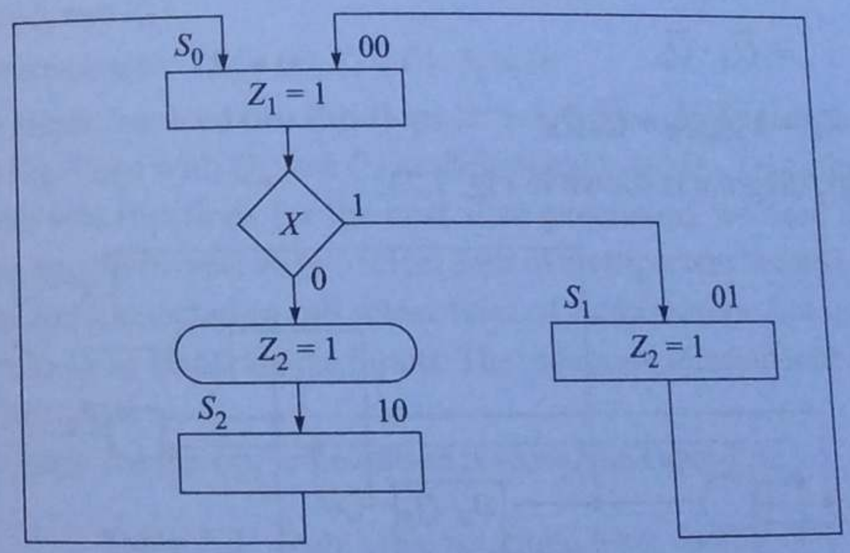


Fig. 7.20 ASM chart

Solution

From the ASM chart, it is observed that there are three states, one input (X) and two outputs (Z_1 and Z_2). These three states are: $S_0 \equiv 00$, $S_1 \equiv 01$, $S_2 \equiv 10$. For three states, we need to use two flip-flops ($2^2 > 3$) for the implementation. Let us have two D flip-flops with Q_A and Q_B as their outputs and D_A, D_B as their inputs.

The state table for the ASM chart of Fig. 7.16 will be as shown in Table 7.2.

Table 7.2 State table

Present state		*Next state*		*Input*	*Outputs*	
Q_A	Q_B	D_A	D_B	x	Z_1	Z_2
0	0	1	0	0	1	1
0	0	0	1	1	1	0
0	1	0	0	X	0	1
1	0	0	0	X	0	0

The state table shown in Table 7.2 has columns for present state, next state, input, and outputs. Since we have selected D flip-flops, the next state will be same as the flip-flop inputs. Hence, it is directly written in terms of D_A and D_B. Since every state transition does not have a decision box, state transitions $01 \rightarrow 00$ and $10 \rightarrow 00$ are shown with the input x (don't-care). Similarly $Z_1 = 1$ during state S_0, it is written as 1 for both the input conditions ($x = 0$ and $x = 1$) in the state table. $Z_2 = 1$ if the present state is S_0 and $x = 0$, or if the present state is S_1.

From the state table, it is observed that the flip-flop input D_A is 1, if $Q_A = 0$, $Q_B = 0$, and $x = 0$. And the flip-flop input D_B is 1, if $Q_A = 0$, $Q_B = 0$, and $x = 1$. Hence, the expressions for D_A and D_B are

$$D_A = \bar{x} \cdot \bar{Q}_A \cdot \bar{Q}_B \tag{7.1}$$

$$D_B = x \cdot \bar{Q}_A \cdot \bar{Q}_B \tag{7.2}$$

$Z_1 = 1$ if $Q_A = 0, Q_B = 0, x = 0$, and $Q_A = 0, Q_B = 0, x = 1$. And $Z_2 = 1$ if $Q_A = 0, Q_B = 0, x = 0$, and $Q_A = 0, Q_B = 1, x = X$. Hence, the expressions for Z_1 and Z_2 are

$$Z_1 = \bar{x}\bar{Q}_A\bar{Q}_B + x\bar{Q}_A\bar{Q}_B = \bar{Q}_A \cdot \bar{Q}_B \tag{7.3}$$

$$Z_2 = \bar{x}\bar{Q}_A\bar{Q}_B + \bar{Q}_A Q_B \tag{7.4}$$

The logic diagram is shown in Fig. 7.21.

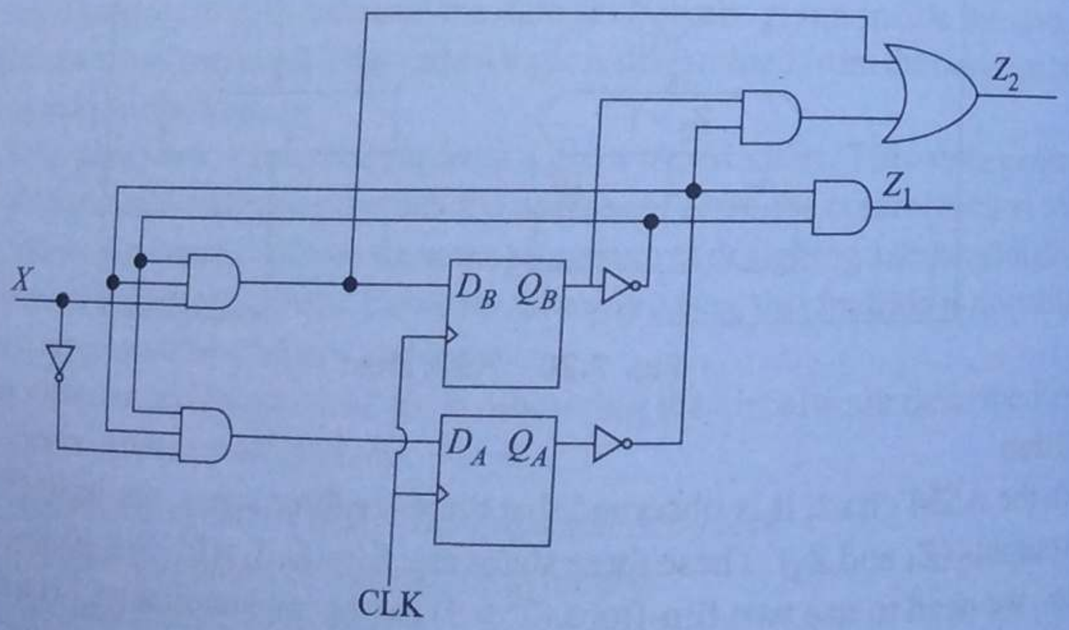


Fig. 7.21 Logic diagram

7.3.2 Multiplexer Controller Method

This is a simple method of synthesizing the combinational logic for any controller having the following desirable features:

1. It produces a design which has a direct correspondence with the algorithm.
2. It is a standard method which can be applied to any ASM chart.
3. It needs the complete ASM chart for its realization.
4. It is very easy to learn and implement.

In this method, the design is pitted in a regular pattern of three levels of components. The first level contains multiplexers which determine the next state (replacing the discrete gates used in the traditional method). The second level consists of registers to hold the present state (similar to flip-flops used in the traditional method) and the third level has decoders to provide outputs for each control state (replacing the combinational circuit using gates in the traditional method).

This method uses a multiplexer for the input to each state flip-flop. Each multiplexer produces a new input to its state flip-flop. The set of multiplexers, each producing an input to its respective flip-flop, gives the output for the next state. The present state code, which is the output of the flip-flops, feeds the select input lines of each multiplexer which in turn selects the appropriate multiplexer input for the next state (flip-flop inputs).

Example 7.5 Implement the ASM chart shown in Fig. 7.20 using multiplexer control method.

Solution

From the ASM chart, it is observed that there are three states, one input (X) and two outputs (Z_1 and Z_2).

The three states are: $S_0 \equiv 00, S_1 \equiv 01, S_2 \equiv 10$

For three states, we need two flip-flops ($2^2 > 3$) for the implementation. Let us have two D flip-flops with Q_A and Q_B as their outputs and D_A, D_B as their inputs. Driving these two flip-flops for the next state generation, we need to use two multiplexers, and there must be two select lines of multiplexers because the output of flip-flops are connected to the select lines of multiplexers. Let us use a 4:1 multiplexer; I_0, I_1, I_2, and I_3 are the inputs. The inputs are selected such that we get the desired next state

The state table for the implementation is shown in Table 7.3.

Table 7.3 State table for multiplexer control

Present state		*Next state*		*Input*	*Outputs*	
Q_A	Q_B	D_A	D_B	x	Z_1	Z_2
0	0	1	0	0	1	1
0	0	0	1	1	1	0
0	1	0	0	X	0	1
1	0	0	0	X	0	0

The output of Mux-1 driving flip-flop A, $I_0 = \bar{x}$, $I_1 = 0$, $I_2 = 0$, and $I_3 = X$.
The output of Mux-2 driving flip-flop B, $I_0 = x$, $I_1 = 0$, $I_2 = 0$, and $I_3 = X$.
The output equations remain same as in Example 7.4, which are

$$Z_1 = \overline{Q}_A \cdot \overline{Q}_B \tag{7.5}$$

$$Z_2 = \bar{x}\,\overline{Q}_A \cdot \overline{Q}_B + \overline{Q}_A \cdot Q_B \tag{7.6}$$

The logic diagram for the circuit is as shown in Fig. 7.22.

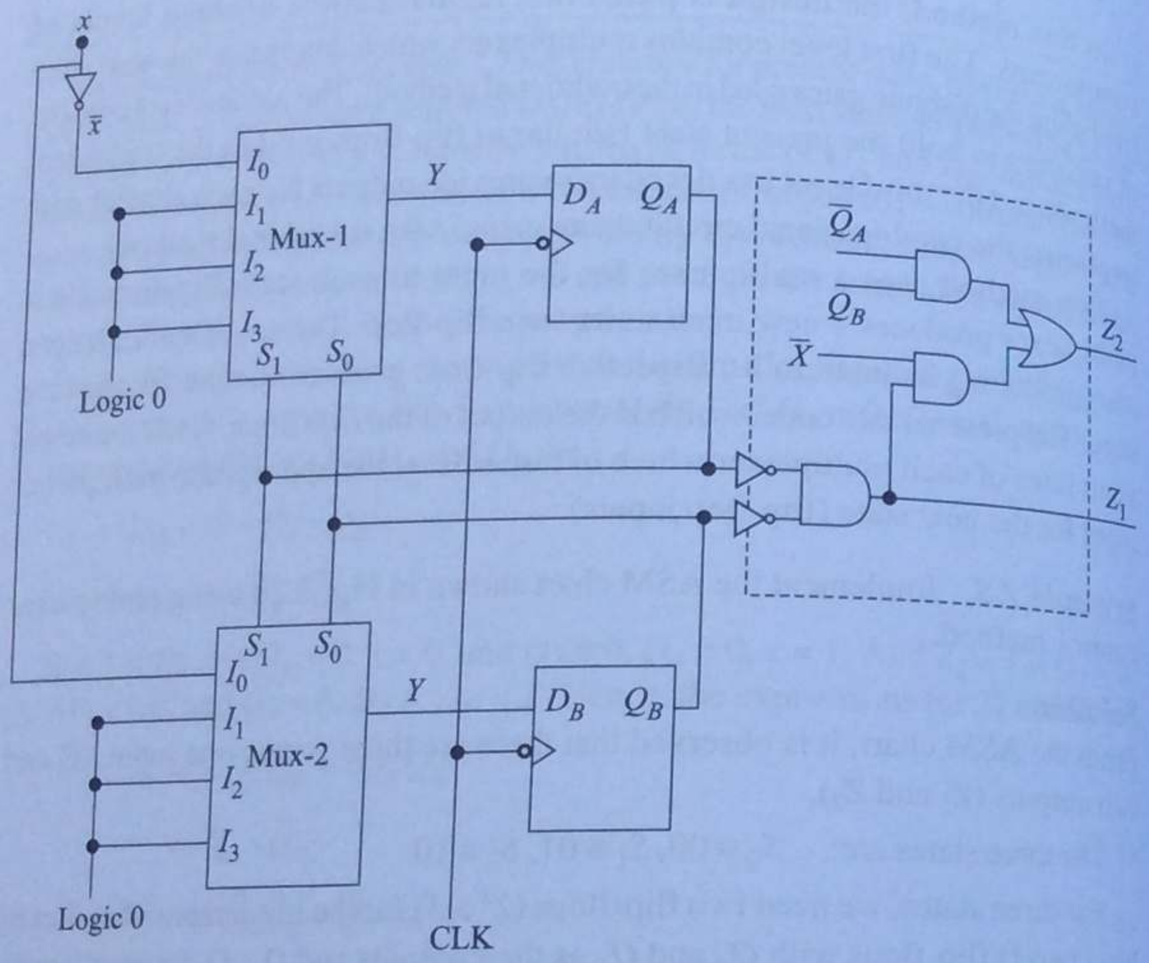


Fig. 7.22 Logic diagram

The outputs Z_1 and Z_2 are generated in the similar way as in Example 7.4. This combinational circuit can be replaced by using a decoder with x, Q_A and Q_B as select lines.

7.4 EXAMPLES ON ASM CHARTS

Example 7.6 Obtain ASM charts for the following state transitions.

(a) If $x = 0$, the control goes from state T_1 to T_2. If $x = 1$, generate a conditional operation and go from T_1 to T_2.

(b) If $x = 1$, the control goes from T_1 to T_2 and then to T_3. If $x = 0$, the control goes from T_1 to T_3.

(c) Start from state T_1. If $xy = 00$, go to T_2; if $xy = 01$, go to T_3. If $xy = 10$, go to T_1; otherwise go to T_3.

Solution

(a) From the description, there are two distinct states T_1 and T_2 with one external input x. When $x = 0$, the control goes from T_1 to T_2, otherwise from T_1 to T_2 with conditional operation. The ASM chart is shown in Fig. 7.23.

(b) From the description, there are three distinct states T_1, T_2 and T_3 with one external input x. When $x = 1$, the control goes from T_1 to T_2 and to T_3, else generate a conditional operation and transit from T_1 to T_3. The ASM chart is shown in Fig. 7.24.

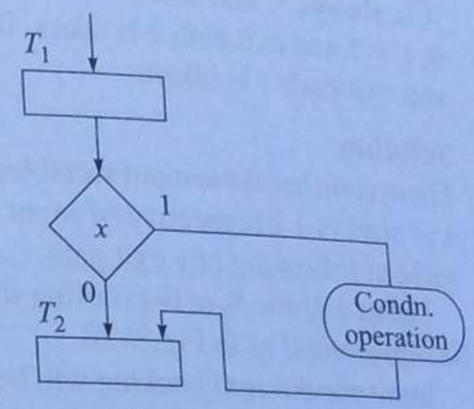


Fig. 7.23 ASM chart

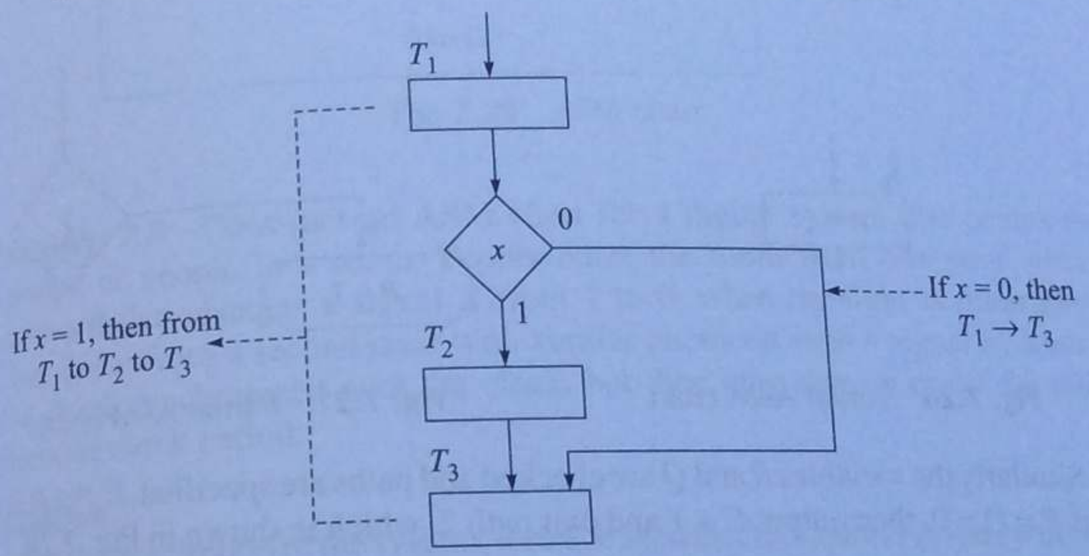


Fig. 7.24 ASM chart

(c) From the description, there are three distinct states T_1, T_2 and T_3 with external inputs x and y. When $xy = 00$, the control goes from T_1 to T_2; when $xy = 01$, control goes from T_1 to T_3; and when $xy = 10$, control goes from T_1 to T_1; else control goes to T_3. The ASM chart is shown in Fig. 7.25.

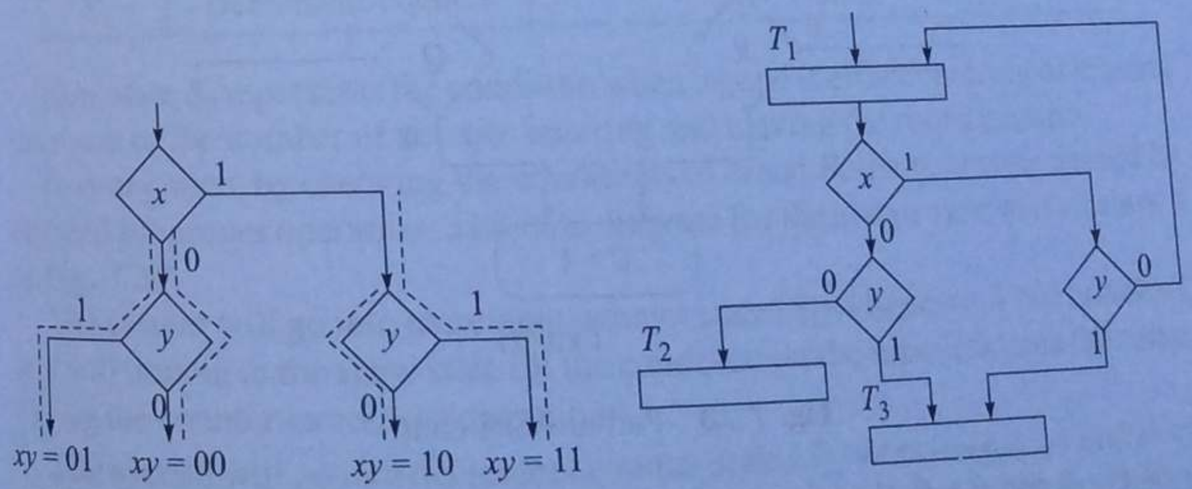


Fig. 7.25 ASM chart for Example 7.6(c)

Example 7.7 Construct an ASM block that has three input variables (P, Q, R), four output variables (A, B, C, D), and two exit paths. For this block, output A is always 1, and B is 1 if and only if $P = 1$. If P and R are 1 or if P and Q are 0, $C = 1$ and exit path 2 is taken. If ($P = 0$ and $Q = 1$) or ($P = 1$ and $R = 0$), $D = 1$ and exit path 1 is taken.

Solution

The system has three input variables and four outputs. From the description, output A is always 1 irrespective of input conditions; output is generated during a state without following any exit path.

Let us assume S_0 as the starting state (initial state) during which $A = 1$ which can be represented as in Fig. 7.26.

Input conditions checking will be followed by state S_0 and two exit paths of the block will be generated, as indicated in Fig. 7.27. Input variable P is checked. If $P = 1$, then output $B = 1$.

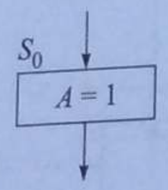


Fig. 7.26 Partial ASM chart

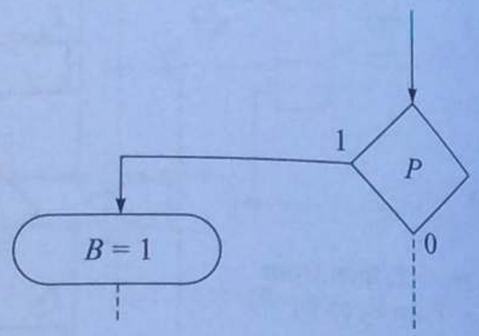


Fig. 7.27 Partial ASM chart

Similarly the variables R and Q are checked and paths are specified. If $P = R = 1$ or if $P = Q = 0$, then output $C = 1$ and exit path 2, which is shown in Fig. 7.28.

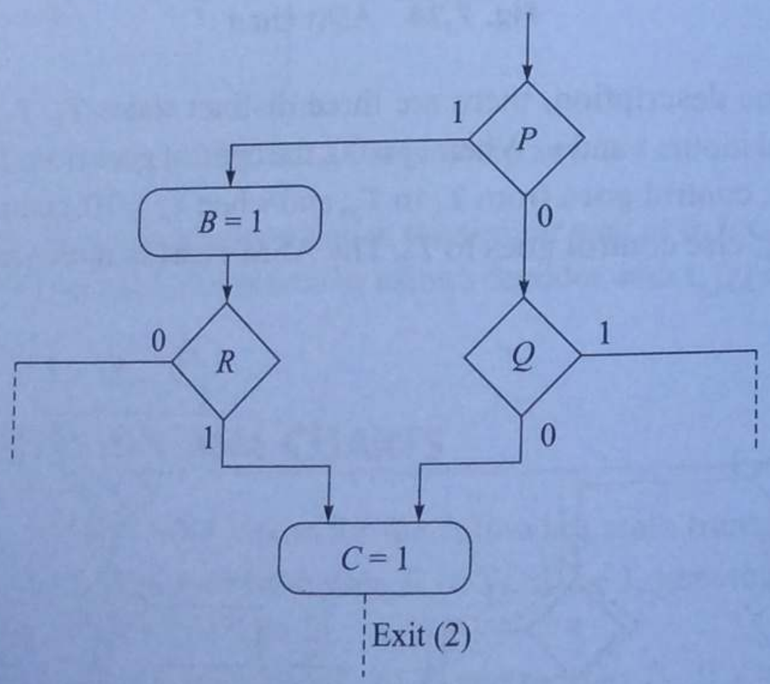


Fig. 7.28 Partial ASM chart

If $P = 1$ and $R = 0$, then output $D = 1$ and exit path 1. If $P = 0$ and $Q = 1$, then output $D = 1$, and exit path 1. The ASM chart is shown in Fig. 7.29.

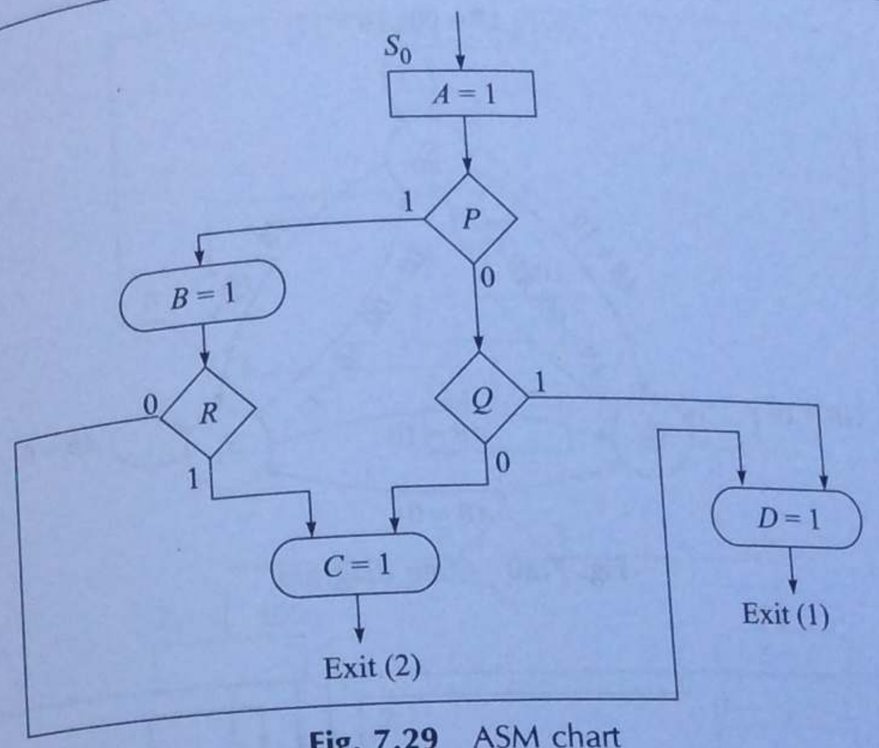


Fig. 7.29 ASM chart

Example 7.8 Construct an ASM chart for a digital system that counts the number of people in a room. People enter the room from one door with a photocell that changes a signal A from 1 to 0, when the light is interrupted. They leave from a second door with similar photocell with a signal B. Both A and B are synchronized with the clock, but they may stay on or off for more than one clock period.

Solution

From the description of the system, it can be seen that the counter system will have three distinct states: increment counter, decrement counter, and inhibit counter. These states with their conditions are given in Table 7.4.

State	Operation	Condition
S_0	Inhibit counter	($A = 1$ and $B = 1$) OR ($A = 0$ and $B = 0$)
S_1	Increment counter	($A = 0$ and $B = 1$)
S_2	Decrement counter	($A = 1$ and $B = 0$)

Here state S_0 represents the condition when no one is either entering or leaving the room or the number of persons entering and leaving the room is same.

In every state, by checking the conditions of A and B, the next state should be decided for proper operation. The state diagram for the above system is as shown in Fig. 7.30.

The system will go into increment counter state (S_1) whenever $A = 0$ and $B = 1$ and will remain in the same state till the condition on the input remains the same, giving the counter increment operation.

The system will go into decrement counter state (S_2) whenever $A = 1$ and $B = 0$ and will remain in the same state till the input remains the same, giving the counter decrement operation. The complete ASM chart will be as shown in Fig. 7.31.

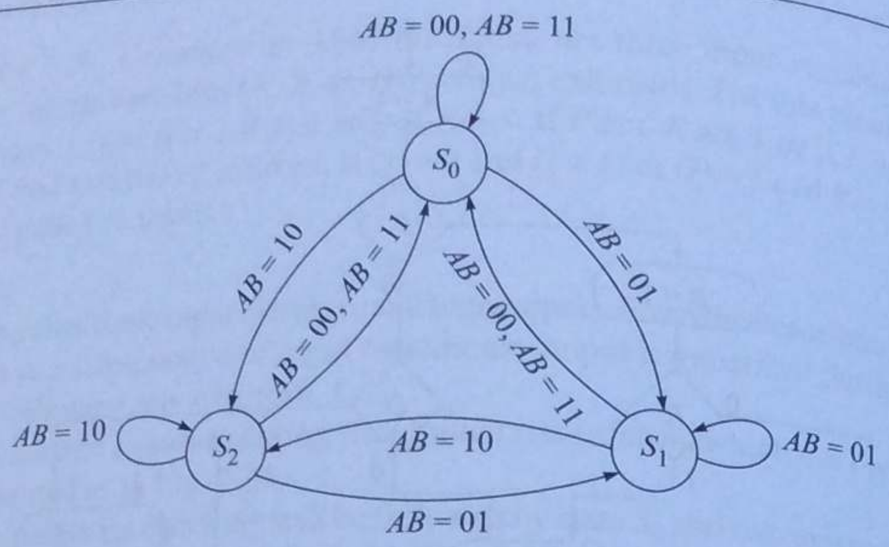


Fig. 7.30 State diagram

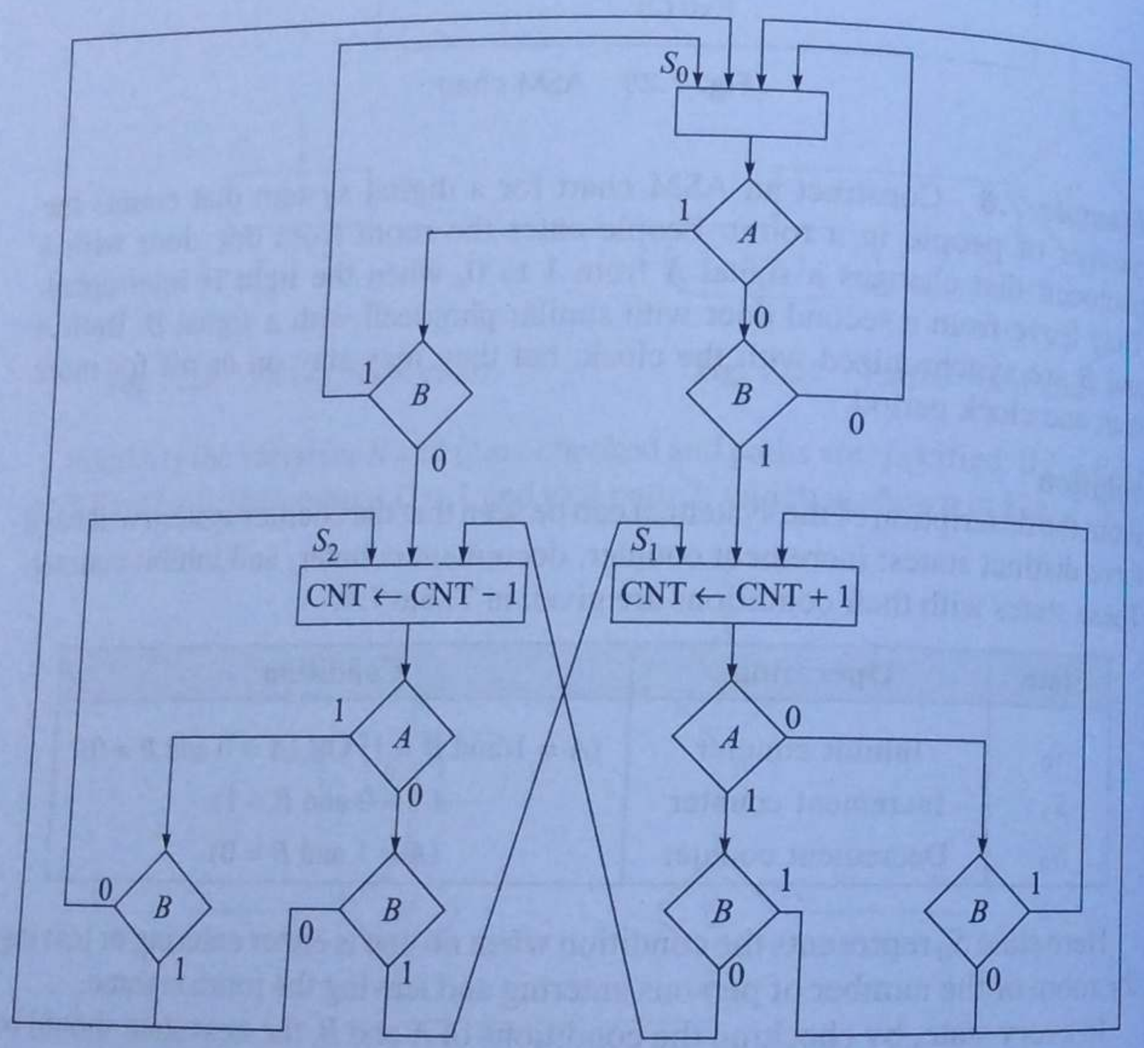


Fig. 7.31 ASM chart

Example 7.9 For the ASM chart shown in Fig. 7.32

(a) Draw the equivalent state diagram.
(b) Derive the state table.
(c) Design the control with D flip-flops and gates.
(d) Design the control with three multiplexers, a register with three flip-flops, and a 3 × 8 decoder.

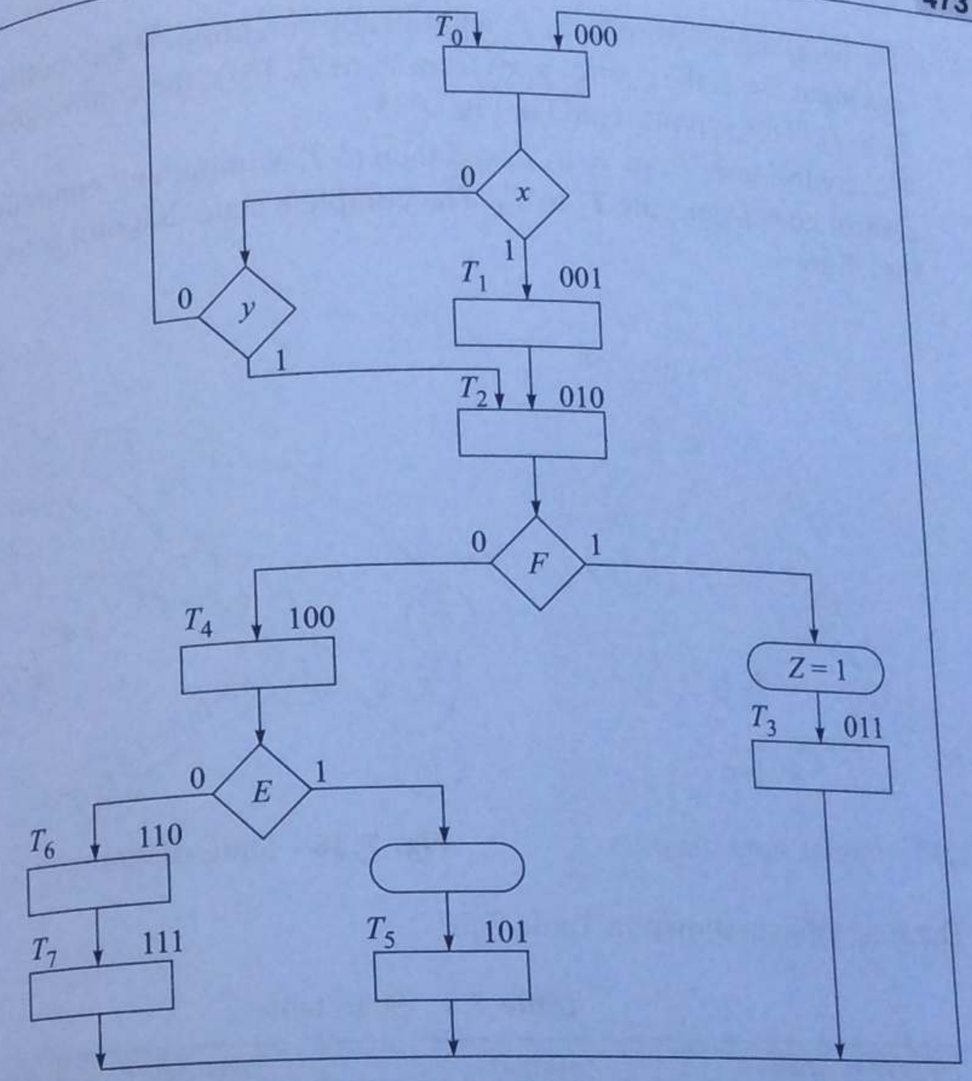


Fig. 7.32 ASM chart

Solution

(a) From the ASM chart, it is found that the control goes from state T_0 to T_1 if $x = 1$, else state T_0 to T_2 if $x = 0$ and $y = 1$. This is represented in Fig. 7.33. The control goes from T_1 to T_2 without any condition. When input $F = 1$ and output $Z = 1$, the control goes from T_2 and T_3. Else, the control goes from T_2 to T_4. This is represented in Fig. 7.34.

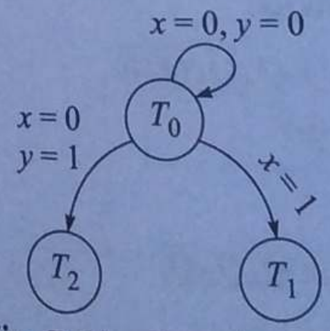


Fig. 7.33 Partial state diagram

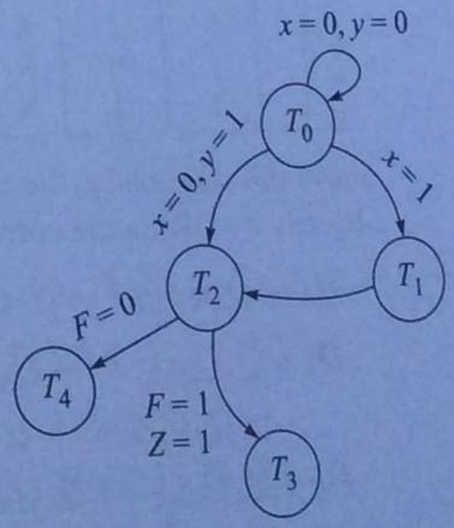


Fig. 7.34 Partial state diagram

The control goes from T_3 to T_0 without any condition. When output $Z = 1$ and input $E = 1$, the control goes from T_4 to T_5. Else, the control goes from T_4 to T_6. This is represented in Fig. 7.35.

The control goes from T_6 to T_7 and then to T_0 without any condition. The control goes from state T_5 to T_0. The complete state diagram is as shown Fig. 7.36.

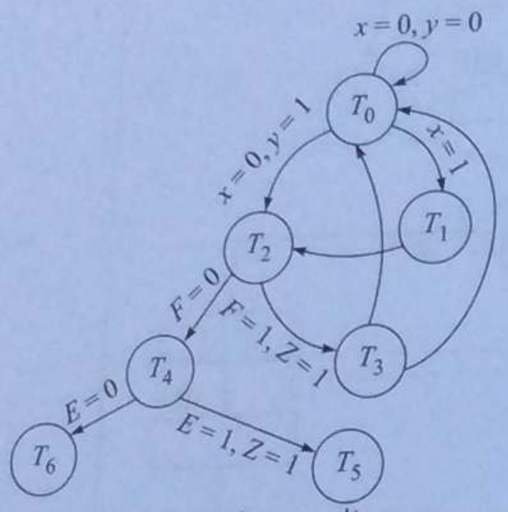


Fig. 7.35 Partial state diagram

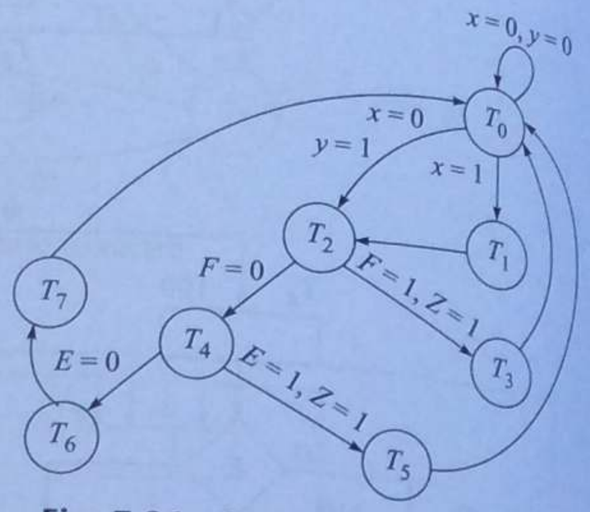


Fig. 7.36 State diagram

(b) The state table is shown in Table 7.5.

Table 7.5 State table

Present State			*Inputs*				*Next State*			*Output*
Q_A	Q_B	Q_C	X	Y	E	F	D_A	D_B	D_C	Z
0	0	0	0	0	X	X	0	0	0	0
0	0	0	1	X	X	X	0	0	1	0
0	0	0	0	1	X	X	0	1	0	0
0	0	1	X	X	X	X	0	1	0	0
0	1	0	X	X	X	1	0	1	1	1
0	1	0	X	X	X	0	1	0	0	0
0	1	1	X	X	X	X	0	0	0	0
1	0	0	X	X	1	X	1	0	1	1
1	0	0	X	X	0	X	1	1	0	0
1	0	1	X	X	X	X	0	0	0	0
1	1	0	X	X	X	X	1	1	1	0
1	1	1	X	X	X	X	0	0	0	0

From the state table, the equations for D_A, D_B, D_C, and Z can be written directly by ORing the corresponding minterms as:

$$D_A = \overline{Q}_A Q_B \overline{Q}_C \overline{F} + Q_A \overline{Q}_B \overline{Q}_C E + Q_A Q_B \overline{Q}_C + Q_A \overline{Q}_B \overline{Q}_C \overline{E} \quad (7.7)$$

$$D_B = \overline{Q}_A \overline{Q}_B \overline{Q}_C \overline{X} Y + \overline{Q}_A \overline{Q}_B Q_C + \overline{Q}_A Q_B \overline{Q}_C F + Q_A \overline{Q}_B \overline{Q}_C \overline{E} + Q_A Q_B \overline{Q}_C \quad (7.8)$$

$$D_C = \overline{Q}_A \overline{Q}_B \overline{Q}_C x + \overline{Q}_A Q_B \overline{Q}_C E + Q_A Q_B \overline{Q}_C E + Q_A Q_B \overline{Q}_C \quad (7.9)$$

$$Z = \overline{Q}_A Q_B \overline{Q}_C F + Q_A \overline{Q}_B \overline{Q}_C E \quad (7.10)$$

The logic diagram for Example 7.9 using D flip-flops and gates is shown in Fig. 7.37.

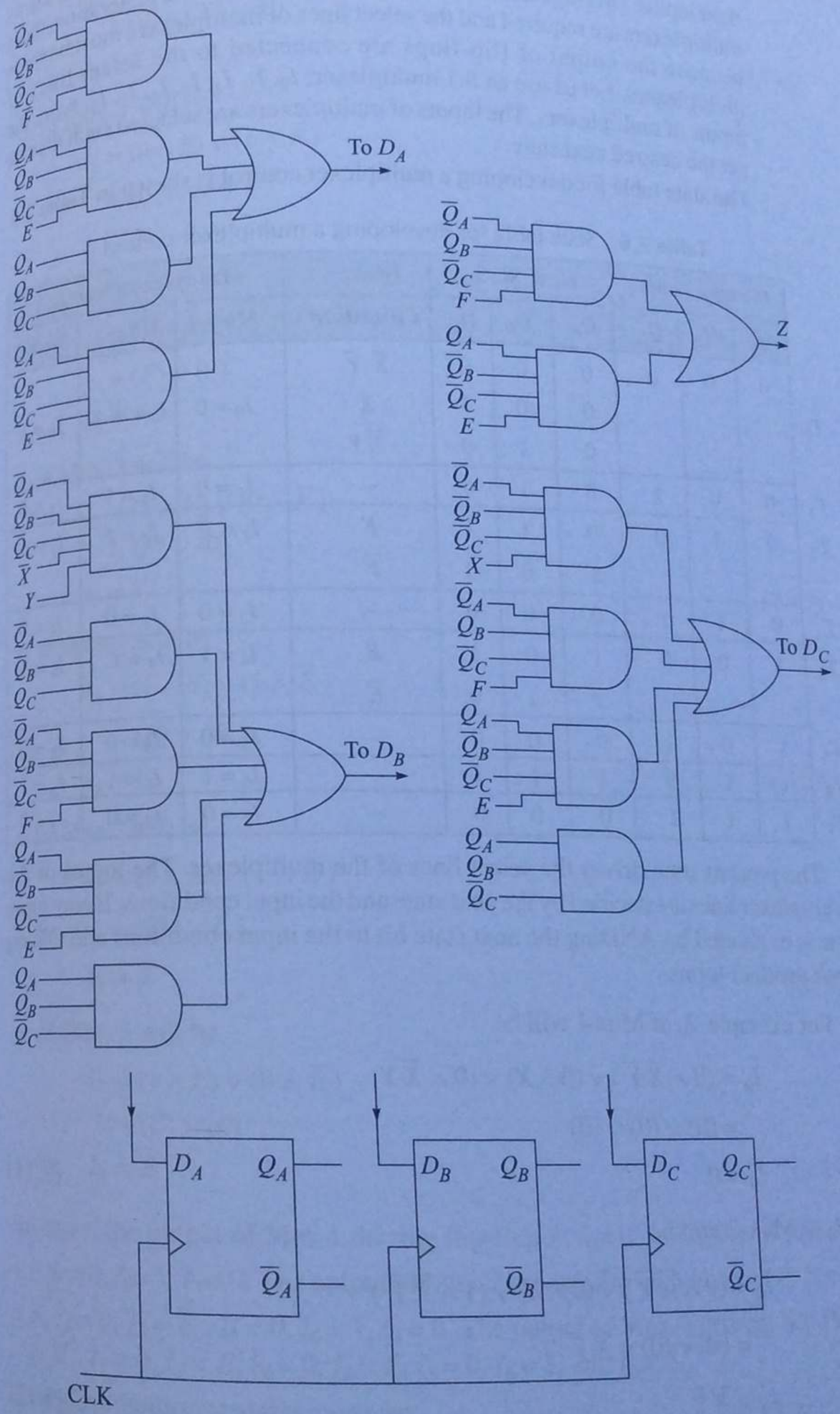


Fig. 7.37 Logic diagram using D flip-flops and gates

(c) For eight states, three flip-flops are needed for the implementation. Let us use three D flip-flops with Q_A, Q_B, Q_C as their outputs and D_A, D_B, D_C as their inputs. Driving these three flip-flops for the next state generation, these multiplexers are required and the select lines of multiplexers must be three because the output of flip-flops are connected to the select lines of multiplexers. Let us use an 8:1 multiplexer; $I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7$ are the inputs of multiplexers. The inputs of multiplexers are selected such that we get the desired next state.

The state table for developing a multiplexer control is shown in Table 7.6.

Table 7.6 State table for developing a multiplexer control

	Present Table			*Next State*			*Input*	*Multiplexer inputs*		
	Q_A	Q_B	Q_C	D_A	D_B	D_C	*Condition*	*Mux-1*	*Mux-2*	*Mux-3*
T_0	0	0	0	0	0	0	$\overline{X}\,\overline{Y}$	$I_0 = 0$	$I_0 = \overline{X}\,Y$	$I_0 = X$
				0	0	1	X			
				0	1	0	$\overline{X}\,Y$			
T_1	0	0	1	0	1	0	–	$I_1 = 0$	$I_1 = 1$	$I_1 = 0$
T_2	0	1	0	0	1	1	F	$I_2 = \overline{F}$	$I_2 = F$	$I_2 = F$
				1	0	0	$\overline{F}$			
T_3	0	1	1	0	0	0	–	$I_3 = 0$	$I_3 = 0$	$I_3 = 0$
T_4	1	0	0	1	0	1	E	$I_4 = 1$	$I_4 = \overline{E}$	$I_4 = E$
				1	1	0	$\overline{E}$			
T_5	1	0	1	0	0	0	–	$I_5 = 0$	$I_5 = 0$	$I_5 = 0$
T_6	1	1	0	1	1	1	–	$I_6 = 1$	$I_6 = 1$	$I_6 = 1$
T_7	1	1	1	0	0	0	–	$I_7 = 0$	$I_7 = 0$	$I_7 = 0$

The present state drives the select lines of the multiplexer. The inputs of the multiplexer lines are decided by the next state and the input conditions. Every input line is evaluated by ANDing the next state bit to the input conditions and ORing such product terms.

For example, I_0 of Mux-1 will be

$$I_0 = (0 \wedge \overline{X}\,\overline{Y}) \vee (0 \wedge X) \vee (0 \wedge \overline{X}\,Y)$$
$$= (0) \vee (0) \vee (0)$$
$$\therefore\ I_0 = 0 \tag{7.11}$$

I_0 of Mux-2 will be

$$I_0 = (0 \wedge \overline{X}\,\overline{Y}) \vee (0 \wedge X) \vee (1 \wedge \overline{X}\,\overline{Y})$$
$$= (0) \vee (0) \vee \overline{X}\,\overline{Y}$$
$$\therefore\ I_0 = \overline{X}\,\overline{Y} \tag{7.12}$$

I_0 of Mux-3 will be

$$I_0 = (0 \wedge \overline{X}\,\overline{Y}) \vee (1 \wedge X) \vee (0 \wedge \overline{X}\,Y)$$
$$= (0) \vee (X) \vee (0)$$
$$\therefore\ I_0 = X \tag{7.13}$$

I_2 of Mux-1 will be

$$I_2 = (0 \wedge F) \vee (1 \wedge \overline{F})$$
$$= (0) \vee (\overline{F})$$
$$\therefore\ I_2 = \overline{F} \tag{7.14}$$

I_2 of Mux-2 will be

$$I_2 = (1 \wedge F) \vee (0 \wedge \overline{F})$$
$$= (F) \vee (0)$$
$$\therefore\ I_2 = F \tag{7.15}$$

I_2 of Mux-3 will be

$$I_2 = (1 \wedge F) \vee (0 \wedge \overline{F})$$
$$= (F) \vee (0)$$
$$\therefore\ I_2 = F \tag{7.16}$$

I_4 of Mux-1 will be

$$I_4 = (1 \wedge E) \vee (1 \wedge \overline{E})$$
$$= (E) \vee (\overline{E})$$
$$\therefore\ I_4 = 1 \tag{7.17}$$

I_4 of Mux-2 will be

$$I_4 = (0 \wedge E) \vee (1 \wedge \overline{E})$$
$$= (0) \vee (\overline{E})$$
$$\therefore\ I_4 = \overline{E} \tag{7.18}$$

I_4 of Mux-3 will be

$$I_4 = (1 \wedge E) \vee (0 \wedge \overline{E})$$
$$= (E) \vee (0)$$
$$\therefore\ I_4 = E \tag{7.19}$$

In short, the output of Mux-1 driving flip-flop A, $I_0 = 0$, $I_1 = 0$, $I_2 = \overline{F}$, $I_3 = 0$, $I_4 = 1$, $I_5 = 0$, $I_6 = 1$, $I_7 = 0$. The output of Mux-2 driving flip-flop B, $I_0 = \overline{X}\,Y$, $I_1 = 1$, $I_2 = F$, $I_3 = 0$, $I_4 = \overline{E}$, $I_5 = 0$, $I_6 = 1$, $I_7 = 0$. The output of Mux-3 driving flip-flop C, $I_0 = X$, $I_1 = 0$, $I_2 = F$, $I_3 = 0$, $I_4 = E$, $I_5 = 0$, $I_6 = 1$, and $I_7 = 0$.

The output equation remains same as:

$$Z = \overline{Q}_A\, Q_B\, \overline{Q}_C\, F + Q_A\, \overline{Q}_B\, \overline{Q}_C\, E \tag{7.20}$$

The logic diagram for the system using multiplexer control is as shown in Fig. 7.38.

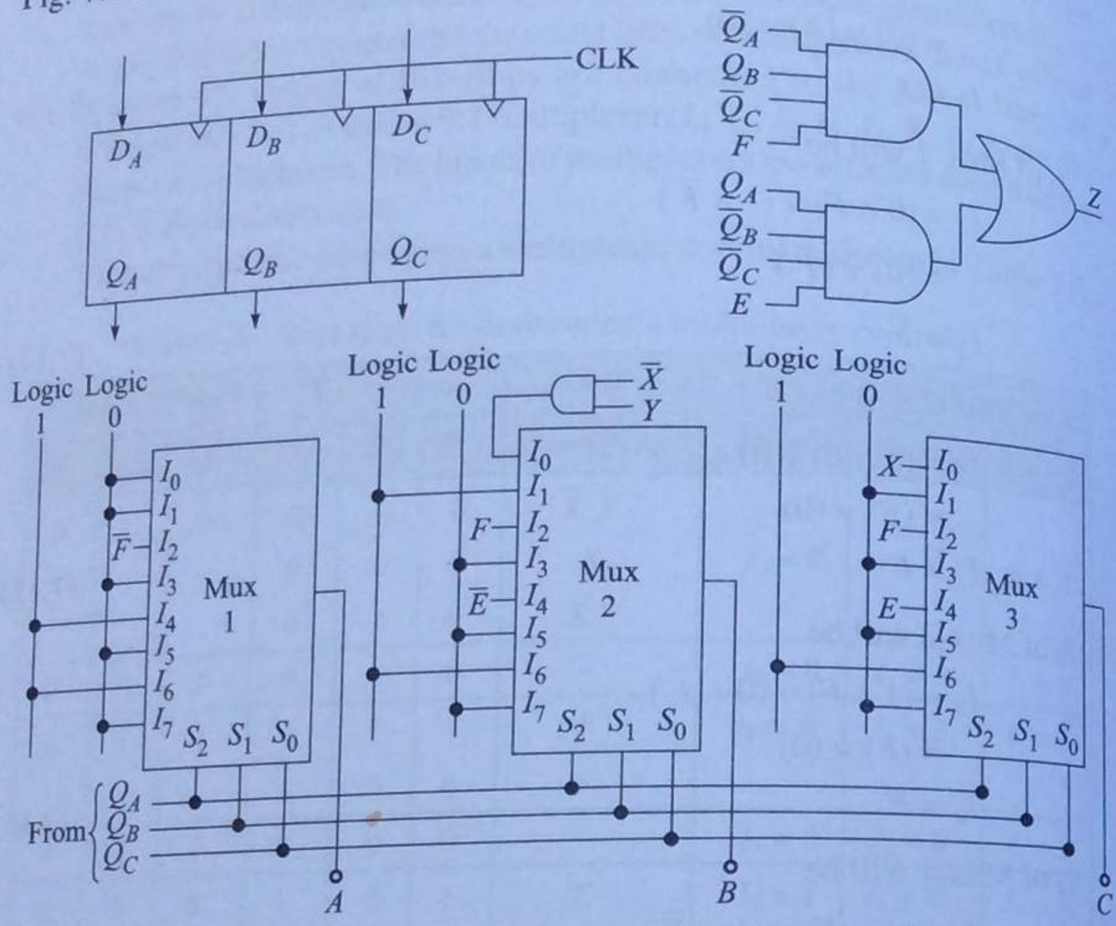


Fig. 7.38 Logic diagram using multiplexer control

7.5 REGISTER TRANSFER LANGUAGE

In the previous section, we discussed ASM charts as a means of describing sequential circuits which are a bit complex in nature. In practice, there are far more complex digital systems, in particular, digital computers. In such cases, ASM charts are not sufficient to describe the complete behaviour of the system, whereas the ASM chart is a convenient tool for description of the control section of the digital systems. Even though the state box of an ASM chart provides informal description of the data sections like generation of output or register operations, it does not give one-to-one correspondence between the ASM chart and the hardware realization.

In order to have a close one-to-one correspondence between the specifications and hardware realizations, it is preferred to use a notation which preserves this relationship for the data section as well as the control section. One such notation which is used in MSI circuits for digital systems is Register Transfer Language (RTL). It is also referred to as Register Transfer Language (RTL).

The primary RTL operation is the transfer of information from one place to another. RTL techniques become very useful when the sequential system consists of operations such as transfer of data from one register to another under the control of some signals. The ASM chart technique for such a system becomes very tedious. In such cases, the digital system [illegible] by splitting the

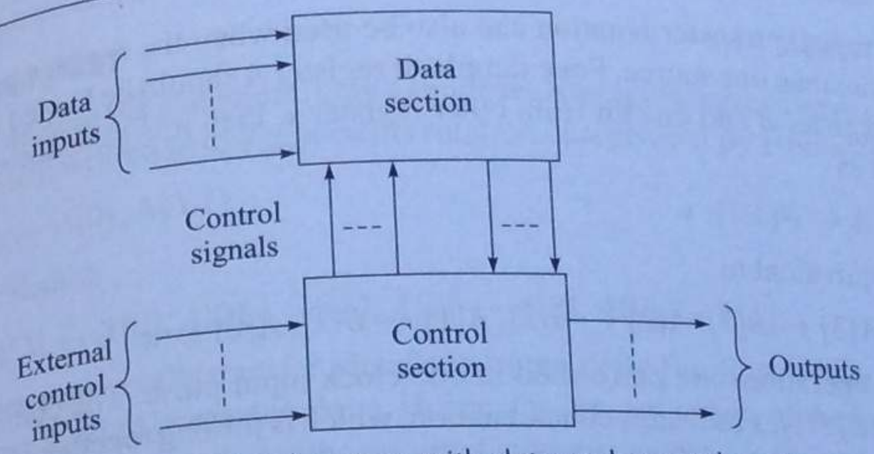


Fig. 7.39 Digital system with data and control sections

7.6 RTL NOTATIONS

The primary operation of RTL is transfer of information from one place to another. Various RTL notations have been developed that describe these primary operations. In this section, these notations have been discussed.

7.6.1 Register Transfer Statement

This statement is used to represent the transfer of contents from register 1 to register 2.

For example, $Z \leftarrow X$ (7.21)

The above statement represents that—when a clock pulse is applied, the current content of register X will be transferred (copied) to register Z.

This is a valid statement for a 1-bit register transfer as well as an n-bit register transfer.

For example, if A and B are 4-bit registers containing

$A = A[3], A[2], A[1], A[0]$ and

$B = B[3], B[2], B[1], B[0]$

then, the statement $A \leftarrow B$ implies the following 1-bit transfers,

$A[3] \leftarrow B[3], A[2] \leftarrow B[2], A[1] \leftarrow B[1]$, and $A[0] \leftarrow B[0]$

The 1-bit register transfer by the statement $Z \leftarrow X$ and the 4-bit register transfer by the statement $A \leftarrow B$ results in the implementations shown in Fig. 7.40 and Fig. 7.41.

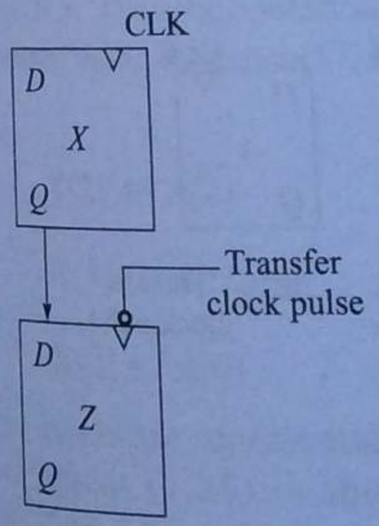


Fig. 7.40 Implementation of

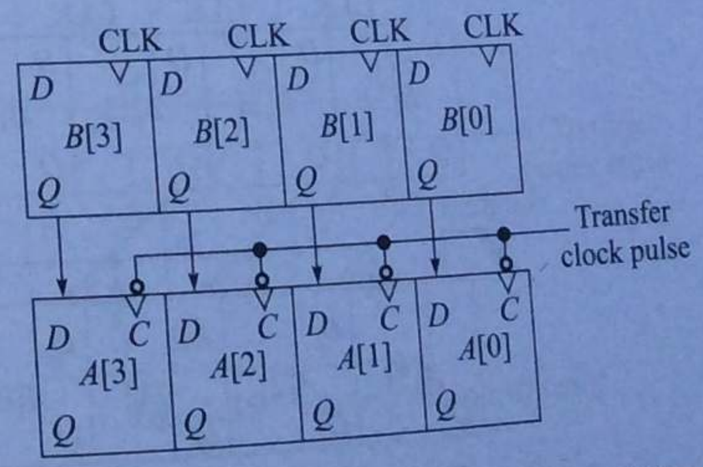


Fig. 7.41 Implementation of 4-bit register

The register transfer notation can also be used when the register gets loaded from more than one source. For example, if register A should be loaded with three bits of register B and one bit from other register x, then it is represented in RTL notation as

$$A \leftarrow B[3:1], x \tag{7.22}$$

It is equivalent to

$$A[3] \leftarrow B[3], A[2] \leftarrow B[2], A[1] \leftarrow B[1], A[0] \leftarrow x$$

These operations are performed in one clock input pulse. The comma in the statement $B[3:1], x$ indicates concatenation, which is joining together of two units of information from different sources into a single unit, or vector of information.

The corresponding hardware for the above statement can be represented as shown in Fig. 7.42.

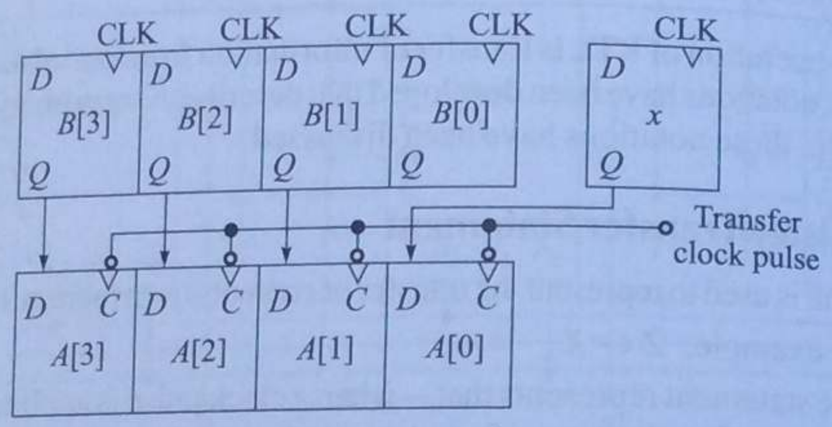


Fig. 7.42 Implementation of a concatenation

Shift operation

With the help of register transfer statement, one can achieve a shift operation as shown below. In the following statement,

$$A \leftarrow 0, B[3:1] \tag{7.23}$$

the operations $A[3] \leftarrow 0$, $A[2] \leftarrow B[3]$, $A[1] \leftarrow B[2]$, and $A[0] \leftarrow B[1]$ will take place.

Thus, the register A will contain the contents of register B, shifted by 1-bit and logic 0 pushed in the MSB ($A[3]$) indicating a logical right shift operation. This implementation is represented in Fig. 7.43.

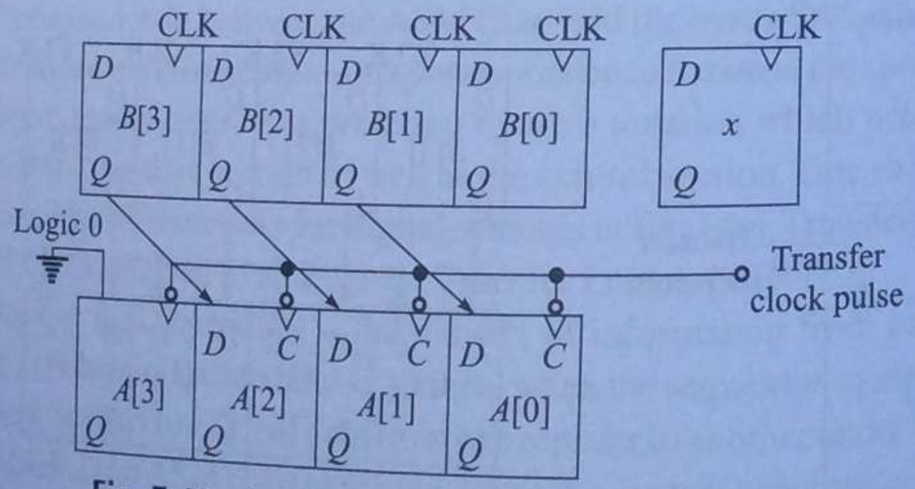


Fig. 7.43 Implementation of a shift operation

Rotate operation

In the register transfer statement, only a single register operation can be performed. The statement written below represents rotation of register A by 1-bit to the right.

$$A \leftarrow A[0], A[3:1] \tag{7.24}$$

It is equivalent to

$$A[3] \leftarrow A[0], A[2] \leftarrow A[3], A[1] \leftarrow A[2], A[0] \leftarrow A[1]$$

This represents a valid transfer, since we assume all the flip-flops within a register to be edge-triggered or master-slave. Hence, the present values of the outputs are stable at the register inputs for the required amount of time.

Figure 7.44 shows the implementation of the above rotation statement.

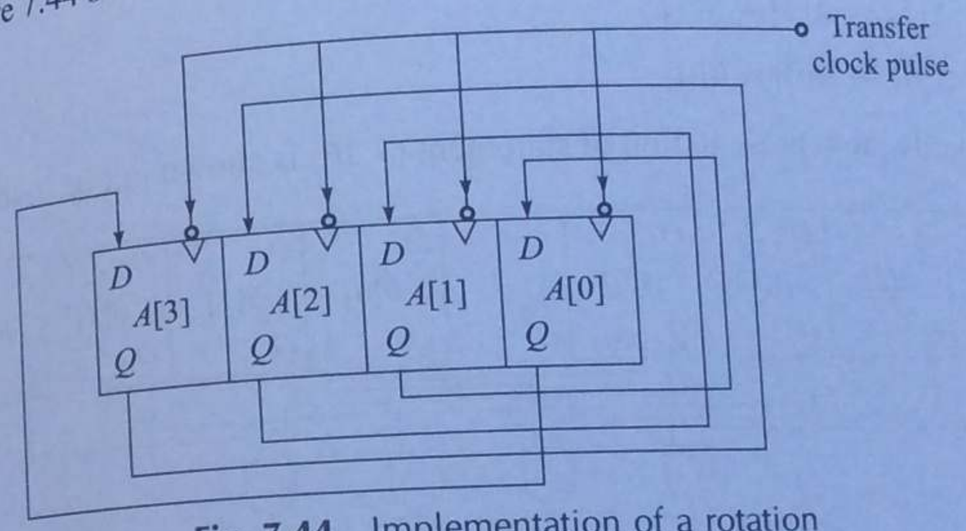


Fig. 7.44 Implementation of a rotation

7.6.2 Logical Operation Statement

Logical operations can also be performed on data. These operations include inversion, ANDing, and ORing of the register contents.

Inversion operation

The invert of register A gets loaded in register X using the statement

$$X \leftarrow \overline{A} \tag{7.25}$$

Let us assume X and A are 3-bit registers. Then the operation is performed using statement (7.25) as

$$X[2] \leftarrow \overline{A[2]}$$

$$X[1] \leftarrow \overline{A[1]} \text{ and}$$

$$X[0] \leftarrow \overline{A[0]}$$

The hardware representation of statement (7.25) is shown in Fig. 7.45.

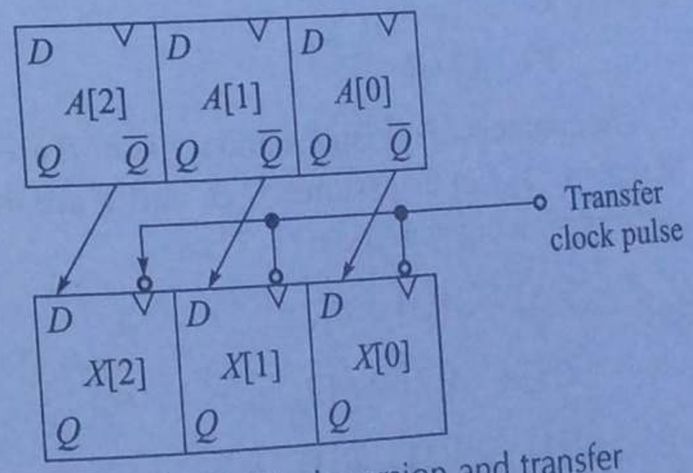


Fig. 7.45 Inversion and transfer

ANDing operation

ANDing operation in RTL is represented by a '$\wedge$' sign. The statement in RTL indicates the ANDing of two registers and transferring the result to the destination register. It is given as

$$X \leftarrow A \wedge B \tag{7.26}$$

The contents of register A and register B are AND bitwise, and the result is stored in register X. Let us assume A, B, and X are 4-bit registers, then the operation is performed using statement (7.26) as

$$X[3] \leftarrow A[3] \wedge B[3]$$
$$X[2] \leftarrow A[2] \wedge B[2]$$
$$X[1] \leftarrow A[1] \wedge B[1]$$
$$X[0] \leftarrow A[0] \wedge B[0]$$

The hardware representation of statement (7.26) is shown in Fig. 7.46.

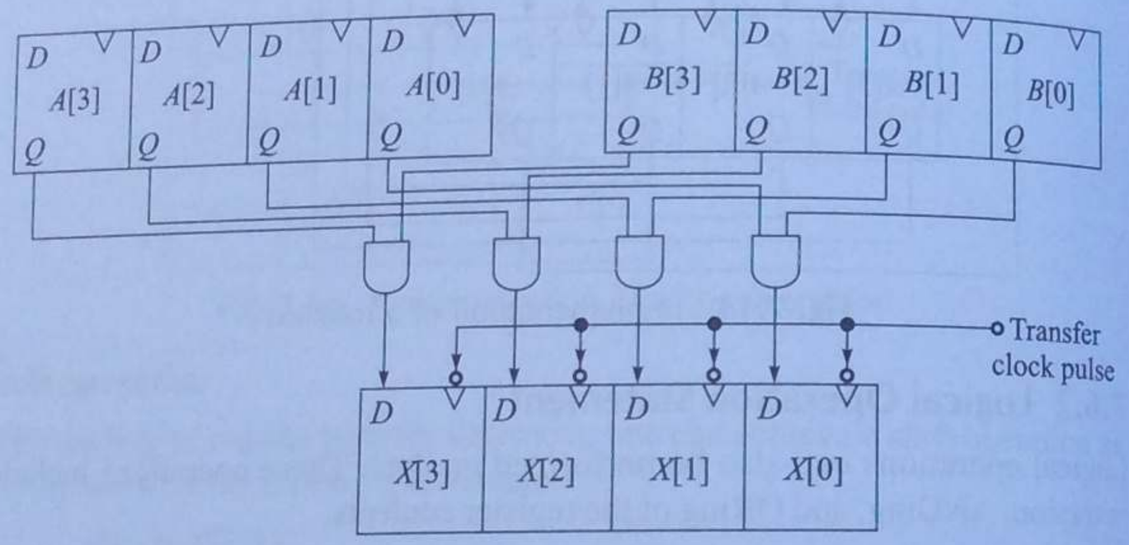


Fig. 7.46 ANDing and transfer operation

ORing operation

ORing operation in RTL is represented by a '$\vee$' sign. The statement in RTL indicates the ORing of two registers and transferring the result to the destination register. It is given as

$$Y \leftarrow A \vee B \tag{7.27}$$

The contents of register A and register B are OR bitwise, and the result is stored in register Y. Let us assume A, B, and Y are 4-bit registers, then the operation is performed using statement (7.27) as

$$Y[3] \leftarrow A[3] \vee B[3]$$
$$Y[2] \leftarrow A[2] \vee B[2]$$
$$Y[1] \leftarrow A[1] \vee B[1]$$
$$Y[0] \leftarrow A[0] \vee B[0]$$

The hardware representation of statement (7.27) is shown in Fig. 7.47.

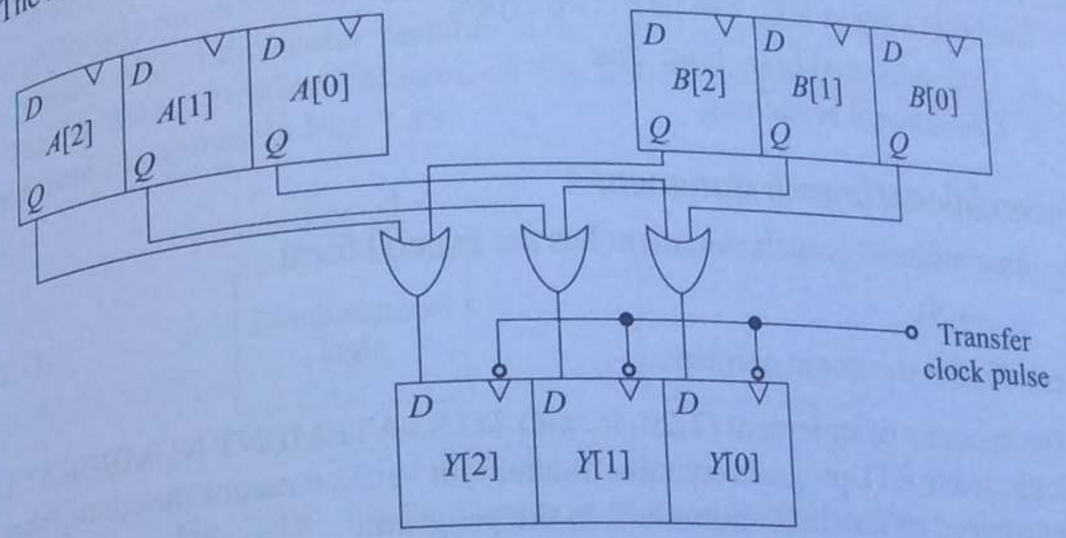


Fig. 7.47 ORing and transfer operation

7.6.3 Connection Operation Statement

In RTL, connecting the contents of a register to an array of output lines needs a different notation than that of a register transfer notation. A register transfer statement is executed by a transfer clock pulse, whereas a connection remains intact for one or more clock pulses.

The symbol '=' is used for the connection operation. The connections of outputs of a register A to an output vector Z is expressed as

$$Z = A \tag{7.28}$$

If register Z and register A are 4-bit vectors, then this statement (7.28) has the following meaning:

$$Z[3] = A[3]$$
$$Z[2] = A[2]$$
$$Z[1] = A[1]$$
$$Z[0] = A[0]$$

The hardware implementation of a connection statement is shown in Fig. 7.48.

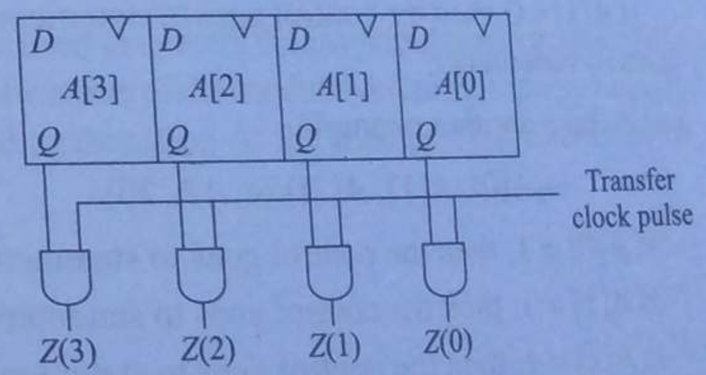


Fig. 7.48 Realization of connection statement

7.6.4 Branch Statements

RTL description of a digital system consists of statements that affect the transfer of information and the sequencing of this transfer operation. Hence, each RTL statement consists the following two parts:

1. Transfer statement, and
2. Branch statement.

We have discussed transfer statements in the previous sections (7.6.1 and 7.6.2). In this section, we will discuss branch statements.

The branching in RTL can be of two types:

1. Unconditional branching, and
2. Conditional branching.

Unconditional branch statement

The unconditional branch statement has the general form

$$\rightarrow(S) \tag{7.29}$$

where S is the statement number.

The meaning of statement (7.29) is "GO TO STATEMENT NUMBER S." For example, if any RTL program contains a statement $\rightarrow(3)$, it means the control should be transferred to statement number 3 in the program.

Conditional branch statement

The conditional branch statement has the general form

$$\rightarrow(f_1, f_2 \ldots f_n)/(S_1, S_2 \ldots S_n) \tag{7.30}$$

where $f_1, f_2 \ldots f_n$ are logical functions, and $S_1, S_2 \ldots S_n$ are statement numbers.

The values of functions $f_1, f_2 \ldots f_n$ are either logic 0 or logic 1. Only one of the functions f_k will be logic 1(true) at a time. If the function $f_k = 1$, then the statement S_k will get executed. If all the functions are logic 0 (i.e. false), then the next sequential statement in the program is executed.

For example,

$$\rightarrow(\overline{A}[1], A[1])/(5, 6)$$

If $A[1] = 0$, then the control goes to statement 5; else if $A[1] = 1$, then the control goes to statement 6.

Let us take another example.

$$\rightarrow(A[0], A[1], A[2])/(8, 12, 20)$$

If $A[0] = 1$, then the control goes to statement 8.

If $A[1] = 1$, then the control goes to statement 12.

If $A[2] = 1$, then the control goes to statement 20.

If $A[0] = A[1] = A[2] = 0$, then the control goes to the next sequential statement, i.e. statement 4.

7.6.5 Conditional Transfer Statement

In RTL notation, a conditional transfer statement is used when the transfer is to be carried out only if a certain condition is satisfied.

For example, if the transfer of contents of register B to A is to take place only if B is greater than A, such a conditional transfer statement is represented in RTL as

$$A * \text{GREATER}\ (B; A) \leftarrow B \tag{7.31}$$

This statement (7.31) contains a transfer which will actually take place only if GREATER $(B; A) = 1$. The '*' sign appearing on the left hand side of the transfer will uniformly mean that the vector on the left hand side of '*' will be the target of the transfer only if the scalar variable at the right hand side of '*' is 1.

The hardware implementation of the above conditional transfer can be represented as shown in Fig. 7.49.

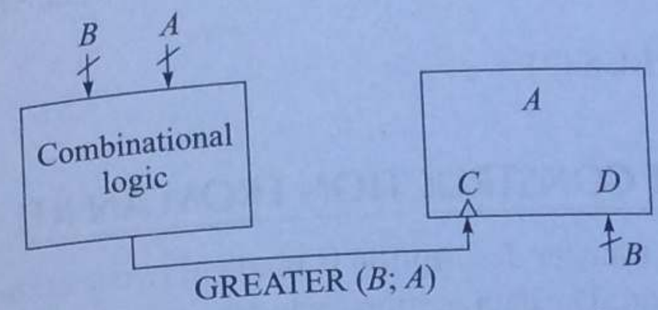


Fig. 7.49 Implementation of conditional transfer

The combination circuit compares the inputs B and A, and generates the clock so that the contents of register B are transferred to register A.

7.6.6 Count Statement

The count statement in RTL is used to express the incrementing or decrementing behaviour of counters. The increment counter statement has the general form

$$\text{CNT} \leftarrow \text{INC (CNT)} \tag{7.32}$$

and the decrement counter statement has the general form

$$\text{CNT} \leftarrow \text{DEC (CNT)} \tag{7.33}$$

7.6.7 Declaration Statement

Declaration statements in RTL are used to specify the principal components of the system. These are used to specify the name of the module, memory, inputs, outputs, termination of the sequence, and termination of the module. The examples of declaration statements are:

```
MODULE  : DATA SHIFTER                    (7.34)
MEMORY  : A[2]; B[2]; C[2]                (7.35)
INPUTS  : X[2]                            (7.36)
OUTPUTS : Y[2]                            (7.37)
    ⋮
END SEQUENCE
END
```

7.6.8 BUS Connection Statement

In an RTL module, if the user wishes to employ a bus with contents that can be transferred into more than one target register, then a BUS connection statement is used which puts the contents of register C over the BUS.

A BUS connection statement has the general form

$$\text{IBUS} \leftarrow \text{C} \tag{7.38}$$

The meaning of statement (7.38) is to put the contents of register C on IBUS. The data connected to the bus appears at the bus output during the entire clock period of the step in which the connection is active.

For a bus connection statement, the bus should be declared with a declaration statement, such as

BUSES : IBUS [3]

7.7 DATA UNIT CONSTRUCTION FROM AN RTL DESCRIPTION

A complete register transfer description is made in two sections, the data section and the control section. The data section includes registers and data paths with the logic required to implement a specific set of transfers. The following example illustrates the construction of a data unit.

Let us consider a digital system having a 2-bit external input A, a 2-bit output Z, and three 2-bit registers W, X, and Y. The sequence of operation on these lines is as follows:

- First, the 2-bit data on input line A is to be loaded in the register.
- The data is shifted in register Y.
- The contents of register Y are complemented and ANDed with the contents of register X.
- Finally, the result is placed on output line Z.

```
MODULE  : DATA MANIPULATOR
MEMORY  : W[2]; X[2]; Y[2]
INPUTS  : A[2]
OUTPUTS : Z[2]
   1. X ← A
   2. Y ← X[0], X[1]
   3. W ← Ȳ
   4. Y ← W ∧ X
   5. Z = Y
END SEQUENCE
```

In the RTL code, first a name is assigned to the system with the declarative statement starting with MODULE. Next, the principal components of the system are specified in the declaration statement which specifies the MEMORY, the INPUTS and the OUTPUTS. For registers with more than one bit, the dimension is specified in the bracket. After the declaration statements, the control sequence of the module appears with numbered steps terminated by the statement END SEQUENCE.

The steps which are followed for deciding the hardware are as follows:

Step 1: In the right hand side of the statements, we have the signals which must be developed for storage in registers or connections on output lines.

Step 2: Connect these signals to the destination on the left hand side.

After step 1, the following hardware gets generated for the data manipulator module, as shown in Fig. 7.50.

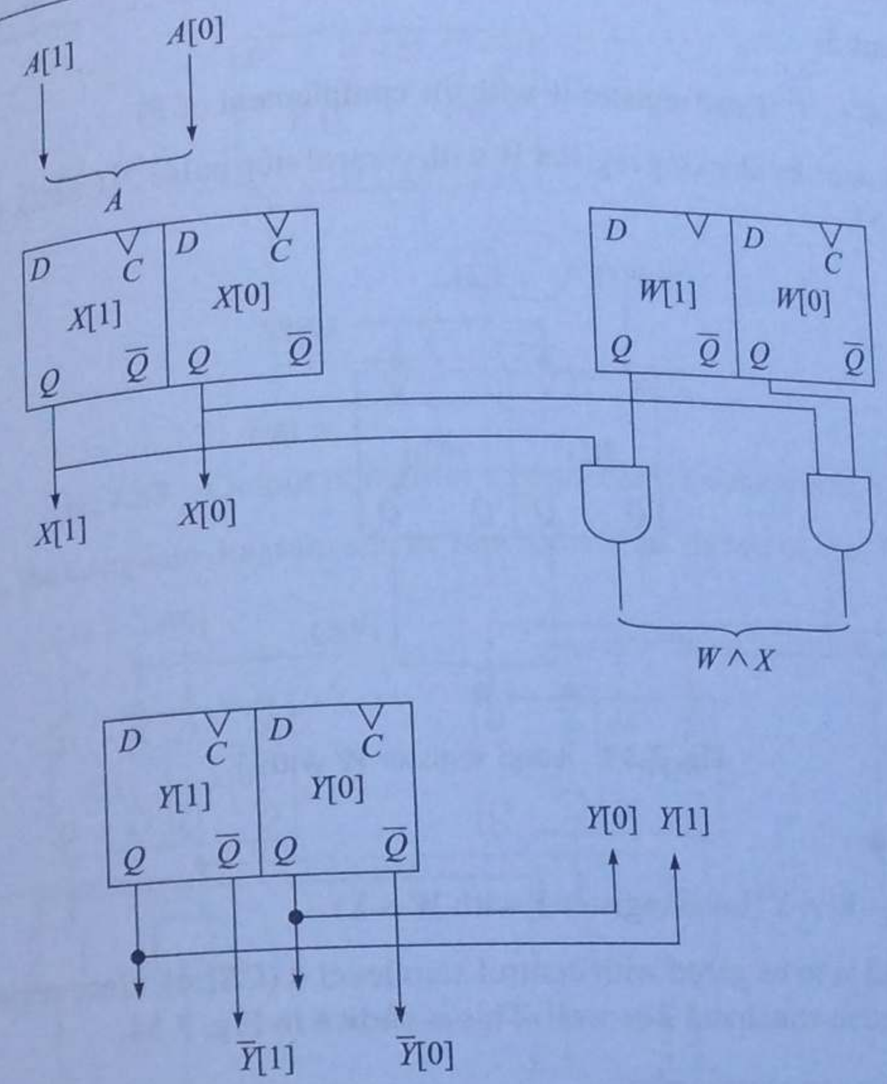


Fig. 7.50 Development of data section

Statement 1:

$X \leftarrow A$ (Load input A lines to register X)

This is to be done by clocking register X with control step pulse 1 (CSP1), as shown in Fig. 7.51.

Statement 2:

$Y \leftarrow X[0], X[1]$ (Shift register X into register Y)

The register Y gets loaded in this statement as well as in statement 4, hence an enable signal is required. Here outputs of register X are ANDed with the control step level 2(CSL 2) and provided as input to register Y, as shown in Fig. 7.52.

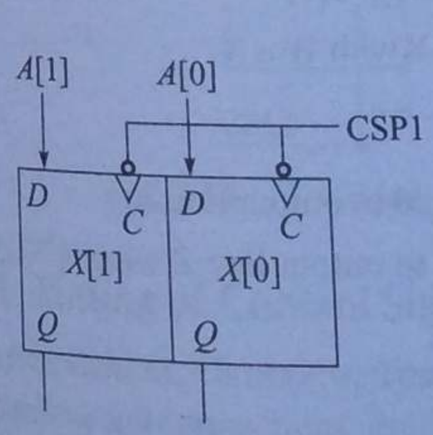


Fig. 7.51 Load input A lines to register X

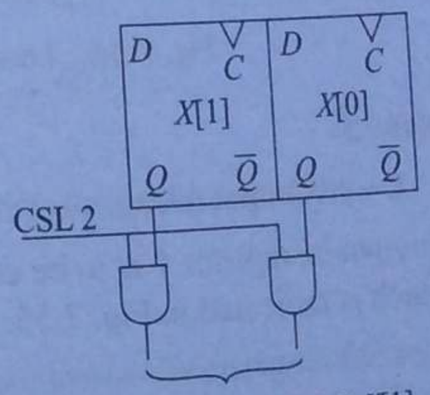


Fig. 7.52 Load X[0], X[1] to register Y

Statement 3:

$W \leftarrow \overline{Y}$ (Load register W with the complement of Y)

This is done by clocking register W with control step pulse 2 (CSP2), as shown in Fig. 7.53.

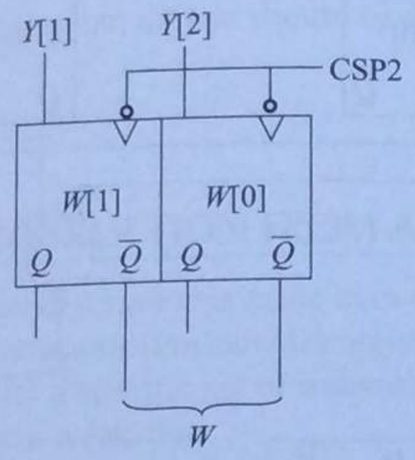


Fig. 7.53 Load register W with $\overline{Y}$

Statement 4:

$Y \leftarrow W \wedge X$ (Load register Y with $W \wedge X$)

This signal is to be gated with control step level 4 (CSL4), since register Y is getting loaded in statement 2 as well. This is shown in Fig. 7.54.

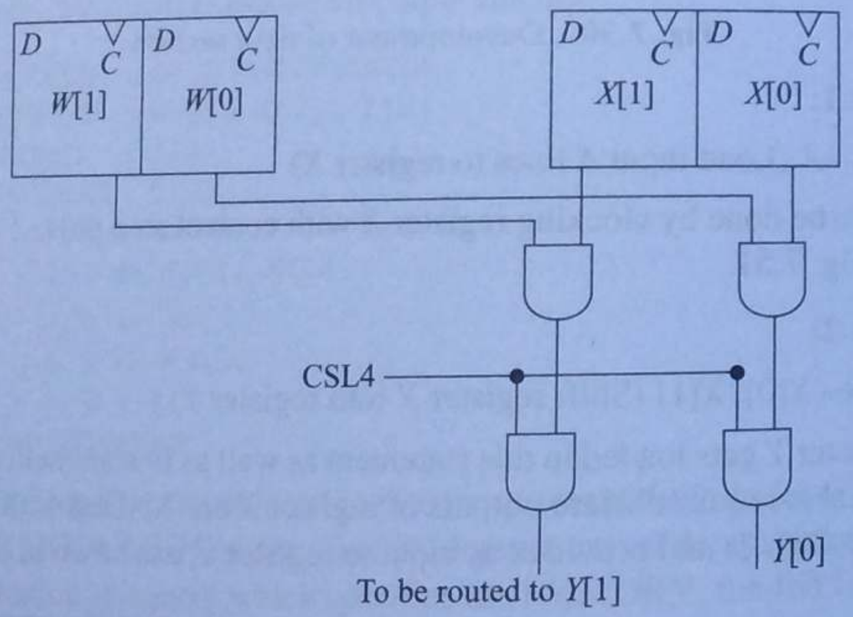


Fig. 7.54 Load register Y with $W \wedge X$

Statement 5:

$Z = Y$ (Output of register Y is connected to output line Z)

The output of register Y is to be connected to output line Z when CSL5 goes high, which is indicated in Fig. 7.55.

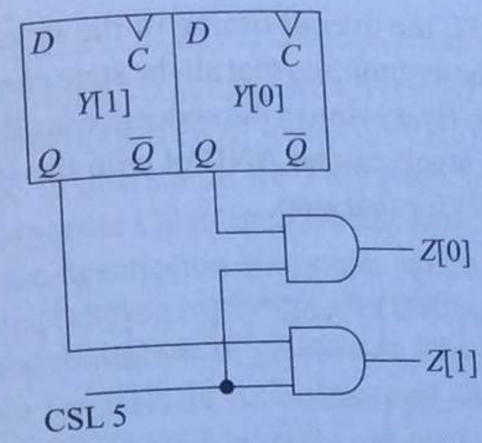


Fig. 7.55 Output of register Y connected to output line Z

Thus, the complete diagram can be represented, as shown in Fig. 7.56.

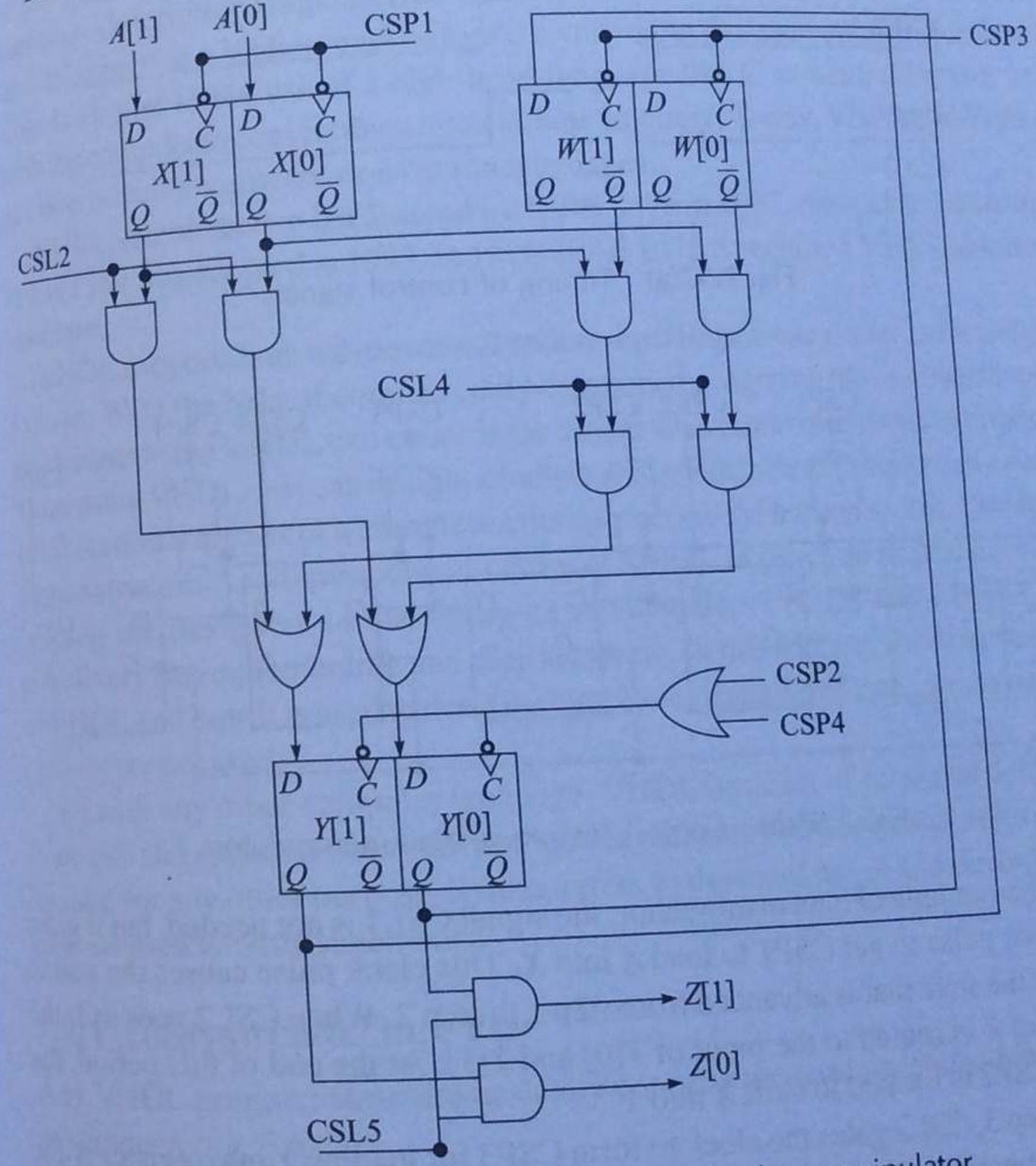


Fig. 7.56 Complete data section for data manipulator

7.7.1 Timing of Control Signals

In the data section, it is observed that certain control levels are required to accomplish the transfers and connections required in various steps. If the steps are to be executed in a proper order, the control levels and pulses must be issued in the corresponding order.

As shown in Fig. 7.57, the overall timing of the system is synchronized by a clock signal and under the assumption that all the state changes are triggered by the falling edge of this clock. Every step in the sequence is allocated one clock period. Each CSL (control step level) can be ANDed with the system clock to generate Single Clock Pulse (CSP) for that step.

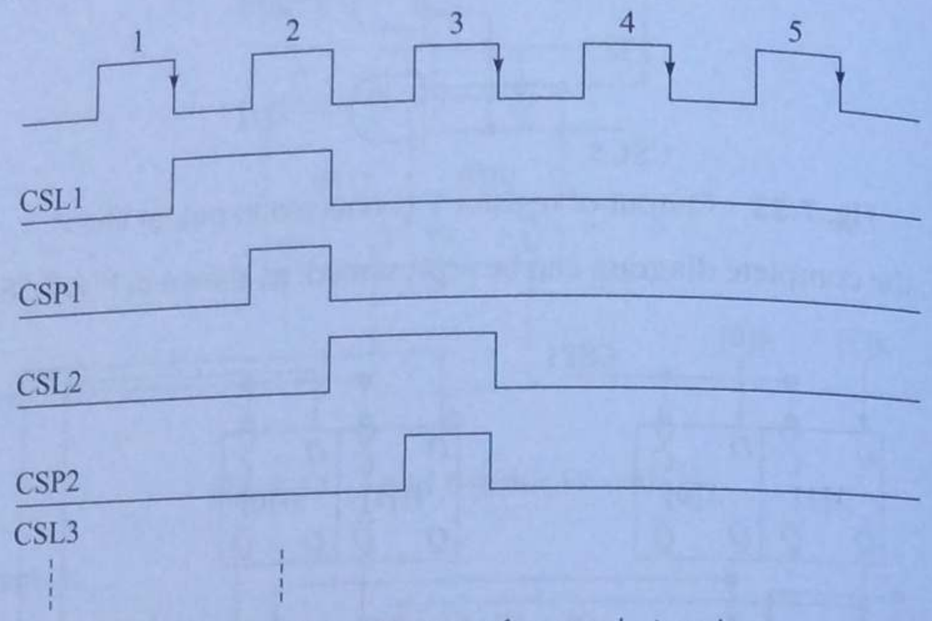


Fig. 7.57(a) Timing of control signals

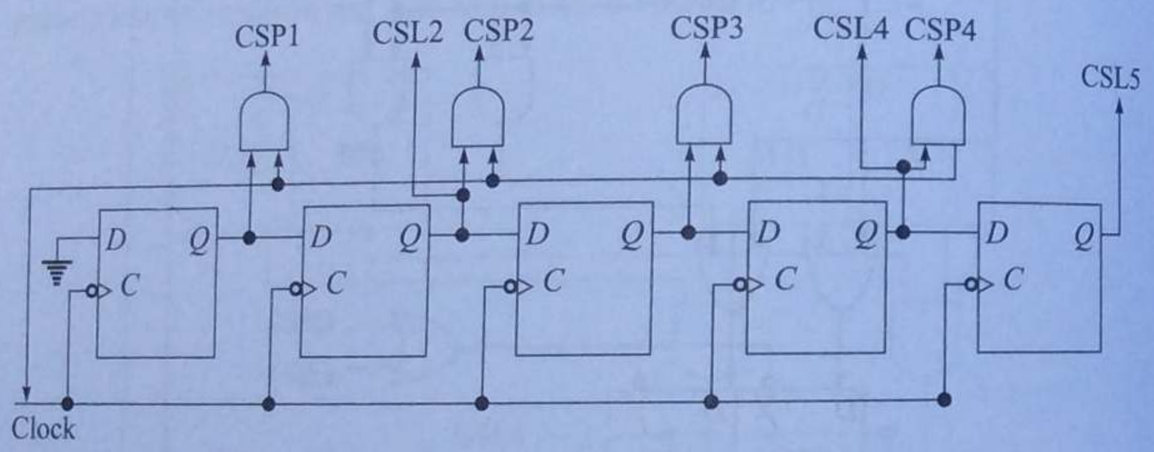


Fig. 7.57(b) Control section of data manipulator

In the example of data manipulator, the signal CSL1 is not needed, but it gates the clock pulse to get CSP1 to load A into X. This clock pulse causes the unit to change the state that is advanced from step 1 to step 2. When CSL2 goes to 1, the output of X is routed to the input of $Y[0]$ and $Y[1]$. At the end of this period, the gated CSP2 is formed to shift X into Y.

At step 3, CSL3 gates the clock to form CSP3 for loading Y into register W for complementing, since the outputs of W are taken from $\overline{Q}$ which is equivalent to $\overline{Y}$. At step 4, CSL4 gates $(W \wedge X)$ to the data inputs of Y which is loaded with CSP4 formed by gating CSL4 with the clock pulse. At step 5, CSL5 gates the outputs of Y to output lines Z. No CSP5 is needed, since no register gets loaded at step 5. Thus, the control unit can be designed, as shown in Fig. 7.57(b).

7.8 VHDL

VHDL is a hardware description language which is used to describe the structure and behaviour of a digital system. VHDL stands for VHSIC Hardware Description Language, where VHSIC in turn stands for Very High Speed Integrated Circuits. Even though the name suggests a high speed integrated circuit design, VHDL is a general purpose hardware description language, which is used to design a wide variety of digital systems, ranging in complexity from a few gates to a very large number gates (containing thousands of gates in an IC).

The improvements in the field of IC technology allow us to place more and more components on an IC. Digital systems are also growing in terms of their complexity. As the system becomes very complex, the detailed design of the system at the gate and flip-flop level becomes very tedious and time-consuming. A hardware description language like VHDL allows the digital system to be designed and debugged at a higher level before conversion to the gate and flip-flop level. This is similar to the use of a high-level language like C to write programs and with the compiler to convert them into machine language. Today, VHDL and Verilog are two popular hardware description languages.

VHDL established to IEEE standard IEEE-1076 in 1987. An updated standard, IEEE-1164 was adopted in 1993. In 1996, IEEE 1076.3 became a VHDL synthesis standard.

VHDL supports both top-down and bottom-up design methodologies for a digital system. With the help of commercially developed programs like synthesis tools, the designs using VHDL can create logic circuit structures from their descriptions. Thus, using VHDL, one can design, simulate, and synthesize anything from a simple combinational circuit to a complete microprocessor system on a chip. Due to the highly structured and hierarchical nature of VHDL, it plays an important role in helping the designers to use complex devices like CPLDs and FPGAs very effectively for implementing any digital system. In this section, the basic features of VHDL and how it is used to describe simple combinational and sequential logic circuits are covered.

As with any other computer language, VHDL has a set of rules that define the *keywords* and *syntax*. A keyword has a special meaning in the language, and it cannot be used for any other purpose. Syntax refers to the word usage and the order that must be used to write statements.

7.8.1 Entity-architecture Pair

Any VHDL program structure basically contains the entity-architecture pair(s), as shown in Fig. 7.58.

The pair basically consists of:

1. An entity for declaring all the input, output signals, and their types in a digital circuit (or module).
2. Architecture for describing the functioning of the digital circuit with one of the modeling styles (discussed in Sections 7.8.4 to 7.8.6).

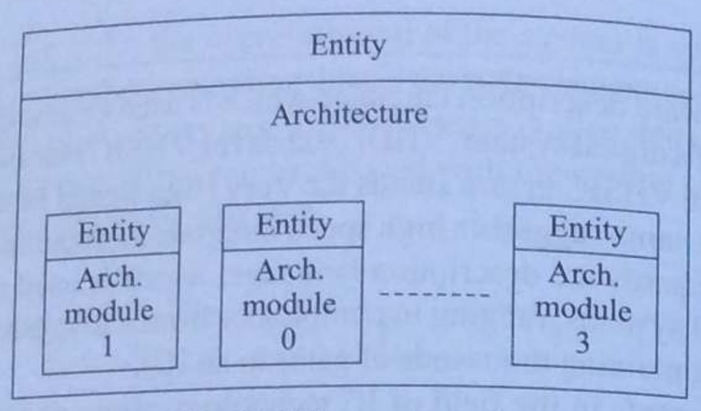


Fig. 7.58 VHDL program structure

7.8.2 Entity Declaration

To write a VHDL program, we need to declare an entity. An entity declaration means declaration of all the input, output signals, and their types in a digital circuit. The basic ports of a digital system and their flow directions are as follows:

(a) IN : Input port (The value of the signal can be read).

(b) OUT : Output port (The value can be assigned to the signal).

(c) INOUT : Bidirectional port (The value can be read as well as assigned).

(d) BUFFER : Output port with read capability (Not a bidirectional port).

The format for entity declaration is as follows:

```
entity <entity_name> is
port (<signal_name1>:<mode><type>;
      <signal_name2>:<mode><type>;
                    .
                    .
                    .
    );
end <entity_name>;
```

Here **entity, is, port,** and **end** are the keywords.

Let us consider the example of a full-adder circuit having inputs and outputs, as shown in Fig. 7.59.

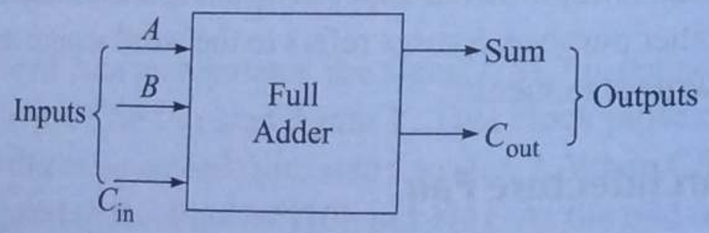


Fig. 7.59 Block diagram of full-adder

It will have an entity declaration as:

```
entity FULL_ADDER is
      port(A, B, Cin : in BIT, – Inputs;
           Sum, Cout : out BIT – Outputs
           );
end FULL_ADDER;
```

Here **entity, is, port, in, out,** and **end** are the keywords.

The entity called FULL_ADDER has three input ports, A, B, and C_{in}; a mode specified as **in,** two output ports Sum and C_{out}; and a mode specified as **out**. BIT is a predefined type of the language that contains the values 0 and 1. Since the ports A, B, C_{in} are specified as the type BIT, it can take the values 0 or 1.

Let us consider another example of an entity declaration for a 4:1 multiplexer circuit shown in Fig. 7.60.

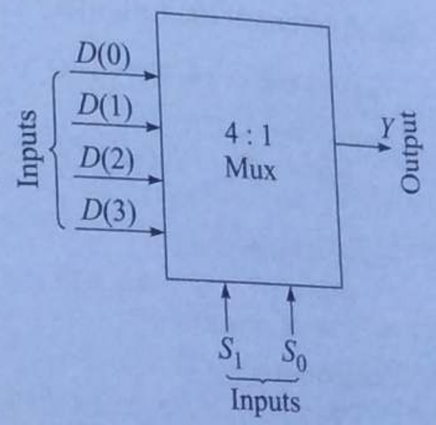


Fig. 7.60 Block diagram of 4 : 1 Mux

```
entity mux_4_1 is
  port (D : in BIT_VECTOR (0 to 3);
        S1, S0: in BIT;
        Y: out BIT
        );
end mux_4_1;
```

The entity mux_4_1 has six input ports and one output port. BIT_VECTOR is a predefined array type in the language. In this example, instead of declaring the 4 input lines as D_0, D_1, D_2, and D_3, they are declared with the help of an array BIT_VECTOR of 4-bits (0 to 3). Individual inputs can be addressed as $D[0]$, $D[1]$, $D[2]$, and $D[3]$. S_1 and S_0 signals are of mode **in** and type **BIT**.

From the above examples, it is seen that the entity declaration does not specify anything about the internals of the entity. It only specifies the name of the entity and the interface ports.

7.8.3 Architecture Body

The internal details of an entity are specified by an architecture body using one of the modeling styles. An architecture specifies the behaviour, function interconnections, and the relationship between the inputs and outputs of an entity. An entity can have more than one architecture, but there cannot be an architecture without an entity.

The syntax for architecture is as follows:

```
architecture <architecture_name> of <entity_name> is
        [<architecture_delcarations>]
begin
        <architecture_statements>
end <architecture_name>;
```

Here **architecture, is, of, begin,** and **end** are the keywords.

The architecture body between **begin** and **end** can have one of the following modeling styles:

1. Structural modeling,
2. Data flow modeling, or
3. Behavioural modeling.

7.8.4 Structural Modeling

An architecture that uses components is often called as *structural design* or *structural modeling*, because it defines the precise interconnection structure of signals and

entities within that entity. In this regard, a pure structural description is equivalent to a schematic diagram or net list of the circuit.

Such a model for the FULL_ADDER entity is described below which defines the structure of the full-adder circuit shown in Fig. 7.61.

```
architecture FA_STRUCT of FULL_ADDER is
component XOR1
  port (X, Y: in BIT; Z : out BIT);
end component;
component AND2
  port (P, Q : in BIT; Y : out BIT);
end component;
component OR3
  Port (L, M, N : in BIT; 0 : out BIT);
end component;
signal S1, S2, S3, S4: BIT;
begin
  G0 : XOR1 port map (A, B, S1);
  G1 : XOR1 port map (S1, Cin, Sum);
  G2 : AND2 port map (A, B, S2);
  G3 : AND2 port map (B, Cin, S3);
  G4 : AND2 port map (A, Cin, S4);
  G5 : OR3 port map (S2, S3, S4, Cout);
end FA_STRUCT;
```

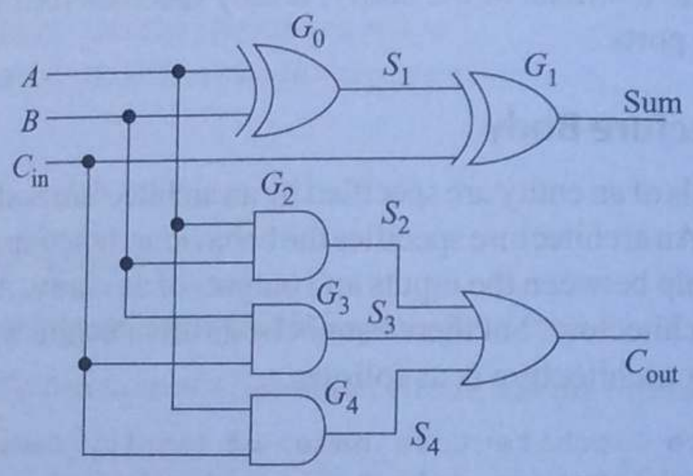


Fig. 7.61 Full-adder circuit

Here, the name given to the architecture body is FA_STRUCT and the entity declaration is FULL_ADDER (previous section) which specifies the interface ports of this architecture body. The architecture body has two parts:

1. Declarative part, and
2. Statement part.

Declarative part comes before **begin** and statement part comes after **begin**.

Declarative part

The component declarations appear in the declarative part. They are XOR_1, AND_2, and OR_3. These declarations specify the interface of these components. These components may either be predefined components in the library or they may be defined separately and added to the library.

The declarative part also contains some signal declarations (S_1, S_2, S_3, S_4) which are used only within the architecture for interconnections. Since they do not appear in the entity, the **'mode'** need not be specified for them.

Statement part

The declared components and signals in the declarative part of the architecture are instantiated in the statement part of the architecture using component instantiation statements. G_0 to G_5 are the component labels for the six components which are instantiated. $Statement_1$ to $Statement_6$ in the body are referred to as component instantiation statements.

A component instantiation statement is a concurrent statement. Hence, the order of this statement is not important. The first statement in which component G_0 is instantiated, the signals A and B are connected to the X, Y input ports of the XOR_1 component, while the output port Z of this component is connected to the internal signal S_1 which is further connected to X input port of another XOR_1 component (G_1).

The syntax for a component instantiation statement is:

Component label 1: Component name **port map** (signal 1, signal 2 ... signal n);

The component name is the name of a previously defined entity that is to be used (declared in the declarative part of the architecture). One instance of the named entity is created for each component statement that invokes its name and each instance must be named by a unique component label. The **port map** keyword introduces a list that associates the ports of the named entity with the signals in the current architecture.

Example 7.10 Write a VHDL code for a 4-bit binary adder using structural modeling having full-adder as a component.

Solution

Figure 7.62 shows a 4-bit binary adder using full-adder as a component.

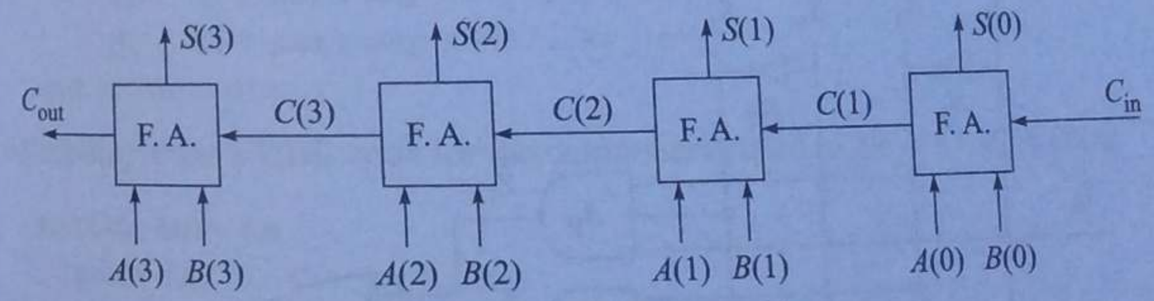


Fig. 7.62 4-bit binary adder

In the figure, $A[3], A[2], A[1], A[0]$; $B[3], B[2], B[1], B[0]$ are the 4-bit inputs; $S[3], S[2], S[1], S[0]$ are the 4-bit outputs; C_i is input carry; and C_{out} is output carry.

The structural model of a 4-bit adder is as follows:

```
library ieee;
use ieee.std_logic_1164.all;
entity Adder_4 bit is
  port (A, B : in bit_vector (3 downto 0);
        Ci : in bit;
        S : out bit_vector (3 downto 0);
```

```
        Cout : out bit
        );
end Adder_4 bit;
architecture struct_adder_4 bit of adder_4 bit is
  component Full adder
  port (A, B, Cin : in bit;
        Sum, Cout : out bit
        );
  end component;
  signal temp : bit_vector (3 downto 1);
begin
     FA0 : full_adder port map (A(0), B(0), Ci, S(0), temp(1));
     FA1 : full_adder port map (A(1), B(1), temp(1), S(1), temp(2));
     FA2 : full_adder port map (A(2), B(2), temp(2), S(2), temp(3));
     FA3 : full_adder port map (A(3), B(3), temp(3), S(3), Cout);
end struct_adder_4 bit;
```

In the architecture body of the code, **full adder** is declared as a component, and a 3-bit internal carry signal is also declared as a **bit vector**.

In the statement part of the architecture body, four instantiation statements are written. Each statement corresponds to a full adder, having names FA_0, FA_1, FA_2, FA_3, and a port map. The signal names following the port map correspond one to one with the signals in the component port.

Example 7.11 Write a VHDL code for a 4:1 multiplexer with an active high output using structural modeling.

Solution

Figure 7.63 shows the internal diagram of a 4:1 multiplexer having two select lines S_1 and S_0, four input lines $I_0 - I_3$, and output Y.

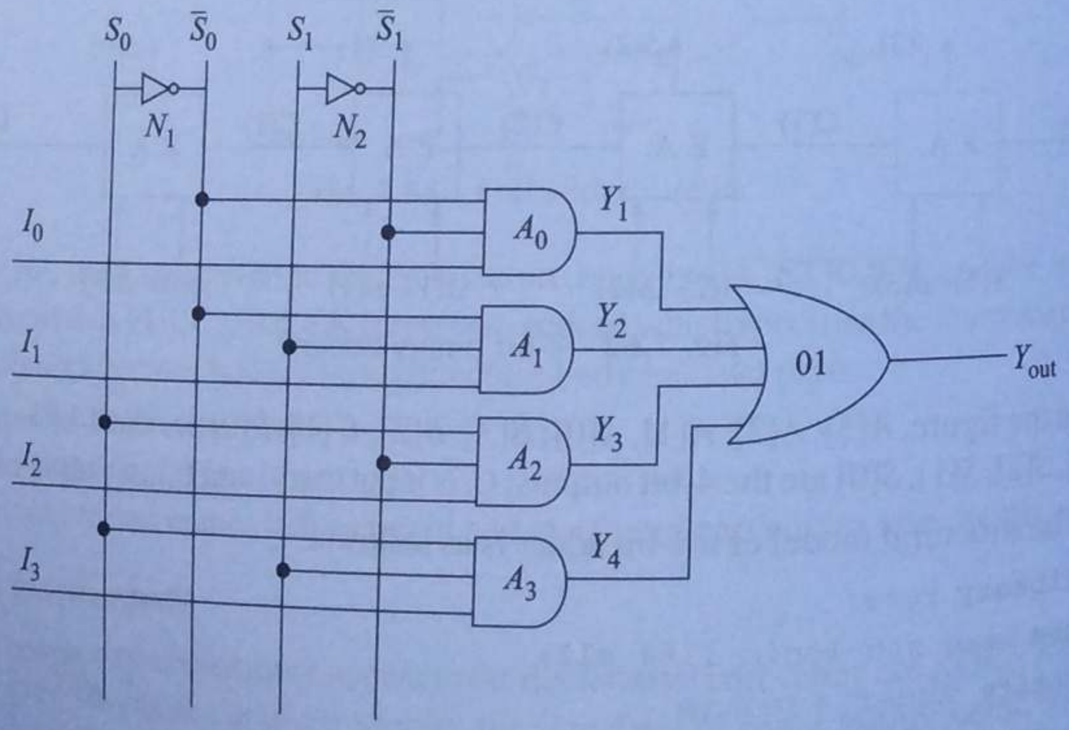


Fig. 7.63 4:1 multiplexer

The structural model of a 4:1 multiplexer is as follows:

```
library ieee;
use ieee.std_logic_1164.all;
entity mux_4_1 is
    port (I : in bit vector (3 downto 0);
          S: in bit vector (1 downto 0);
          Y: out bit);
end mux_4_1;
architecture struct_mux_4:1 of mux 4-1 is
  component AND3
    port (A, B, C: in bit;
          Y : out bit);
    end component;
  component INV
    port (A : in bit; A_inv : out bit);
    end component;
  component OR4
    port (A, B, C, D : in bit;
          Y : out bit);
    end component;
  signal Y : bit_vector (4 downto 1);
  signal S0_inv, S1_inv: bit;

begin
     A0 : AND3 port map (S0_inv, S1_inv, I(0), Y(1));
     A1 : AND3 port map (S0_inv, S(1), I(1), Y(2));
     A2 : AND3 port map (S(0), S1_inv, I(2), Y(3));
     A3 : AND3 port map (S(0), S(1), I(3), Y(4));
     O1 : OR4 port map (Y(1), Y(2), Y(3), Y(4), Y);
     N1 : INV port map (S(0), S0_inv);
     N2 : INV port map (S(1), S1_inv);
end struct_mux_4_1 ;
```

Following is the VHDL code for the components used in the 4:1 multiplexer:

```
entity AND3 is
  port (A, B, C : in bit;
        Y: out bit
        );
end AND3;
architecture A1_AND3 of AND3 is
begin
     Y <=A AND B AND C;
end A1_AND3;
```

```
entity OR4 is
  port (A, B, C, D : in bit;
        Y: out bit
        );
end OR4;
architecture A2_OR4 of OR4 is
begin
  Y ⇐A OR B OR C OR D;
end A2_OR4;
-------------------------------------------------------
-------------------------------------------------------
entity INV is
  port (A : in bit; A_INV : out bit);
end INV;
architecture A3_INV of INV is
begin
  A_INV ⇐ not A;
end A3_INV;
```

Here, the symbol '⇐' is used to assign values.

7.8.5 Data Flow Modeling

In this type of modeling, the data flow is expressed primarily using concurrent signal assignment statements. The structure of the entity is not explicitly specified in this type of modeling, but it is inferred from the equations.

Some of the books on VHDL distinguish between behavioural and data flow architectures and others treat them together as behavioural description. The primary difference is that, one uses processes (discussed in Section 7.8.6) and the other does not.

Data flow descriptions are used in cases where the simple design equations of the circuit are in hand. For example, the full-adder circuit discussed in Section 7.8.4 has the data flow equations as

$$\text{Sum} = A \oplus B \oplus C$$

$$C_{out} = AB + AC + BC$$

These equations can be represented using data flow modeling, having concurrent statements as

Sum ⇐ *A* XOR *B* XOR *C*;

C_{out} ⇐ (*A* AND *B*) OR (*A* AND *C*) OR (*B* AND *C*);

It is also possible to use the following conditional signal assignment statements in data flow modeling, since they are concurrent.

When-else statement

In a **when-else** statement, a Boolean expression combines the individual Boolean terms using VHDL's built-in Boolean operators, such as **and**, **or**, and **not**. Boolean terms are typically Boolean variables or the results of comparisons using relational operators:

= (equal to), / = (not equal to), > (greater than), > = (greater than or equal), < (less than), and < = (less than or equal).

The syntax of a when-else statement is as follows:

(expression 1) **when** (condition) **else** (expression 2);

A few examples of when-else statements are given below.

```
1. X ⇐ '1' when b <= c else '0';
2. Y ⇐ j when X = '0' else
       k when X = '1' else
       m when others;
3. Yout ⇐ I0  when (S0 = '0' and S1 = '0') else
          I1  when (S0 = '1' and S1 = '1') else
          '0' when others;
```

With-select statement

This is another kind of concurrent signal assignment statement. The statement evaluates the given expression and when the value matches one of the choices, it assigns the corresponding signal-value to the signal-name. The choices in the **when** clause can be a single value of the given expression or a list of values. The keyword **others** is used when all the probable values of the choices are not covered.

The syntax of **with-select** statement is shown below.

```
with select_expression select
        target_signal ⇐ expression 1 when choices,
                      ⇐ expression 2 when choices,
                                  ⋮
                      ⇐ expression 3 when others;
```

Following examples illustrate the use of **with-select** statement.

```
1. with select_lines select
                    Y ⇐ I0 when "00",
                        I1 when "01",
                        I2 when "10",
                        I3 when others;

2. with present_state select
          next_state ⇐ I0 when S0,
                       S2 when S1,
                       S3 when S2,
                       S0 when others;
```

Example 7.12 Write a VHDL code using data flow modeling for a 3:8 decoder having active high outputs and an enable line.

Solution

The entity for a 3:8 decoder is shown in Fig. 7.64.

```
entity decoder 3_8 is
  port (En, S2,S1, S0 : in bit;
        Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7: out bit
        );
end decoder 3_8;
```

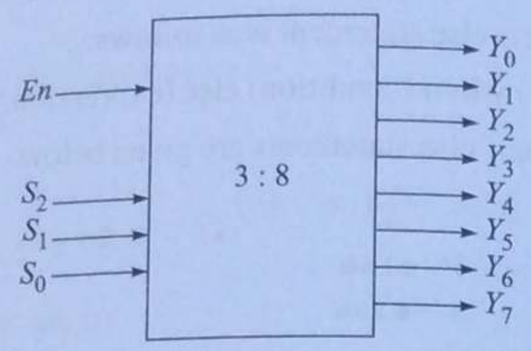


Fig. 7.64 3:8 decoder

Table 7.7 Truth table of 3:8 decoder

E_n	S_2	S_1	S_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

The complete VHDL code for a 3: 8 decoder using **when-else** statement is given below.

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder 3_8 is
      port (En, , S2,S1, S0: in bit;
            Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7: out bit
            );
end decoder 3_8;

architecture dat_flow1 of decoder 3_8 is
    signal temp : std_logic_vector (3 downto 0);
begin
   temp ⇐ En & S2 & S1 & S0;
    Y0 ⇐ '1' when temp = "1000" else
         '0';
    Y1 ⇐ '1' when temp = "1001" else
         '0';
    Y2 ⇐ '1' when temp = "1010" else
         '0';
    Y3 ⇐ '1' when temp = "1011" else
         '0';
    Y4 ⇐ '1' when temp = "1100" else
         '0';
    Y5 ⇐ '1' when temp = "1101" else
         '0';
```



```
    Y6 ⇐ '1' when temp = "1110" else
         '0';
    Y7 ⇐ '1' when temp = "1111" else
         '0';
end dat_flow1;
```

The 3:8 decoder entity has the architecture body dat_flow1. In the declarative part of the architecture, a temporary signal 'temp' is declared for considering the status of enable line and select line together at any instance. This is done with the statement

temp ⇐ En & S_2 & S_1& S_0;

In the architecture statement, '&' is a concatenation operator in VHDL. With the help of this operator, single bits En, S_2, S_1, and S_0 are converted to a 4-bit vector. Hence, when $En = 1$, $S_2 = 0$, $S_1 = 0$ and $S_2 = 0$, temp is assigned the value

temp ⇐ '1' & '0' & '0' & '0' results in

temp = "1000"

By checking the value of temp in the expression part of the **when** statement, output signals are assigned the values 0 or 1. Eight when-else statements realize the truth table of the 3:8 decoder, as shown in Table 7.7.

Example 7.13 Write a 3:8 decoder VHDL code using **with-select** statement.

Solution

An alternative code for a 3:8 decoder is given below which uses **with-select** statement in the architecture. Instead of declaring the outputs and select inputs individually, they are declared as an 8-bit array and a 3-bit array in the entity, as shown below.

```
entity decod 3_8 is
  port (En : in bit;
        S : in bit_vector (2 downto 0);
        Y : out bit_vector (7 downto 0)
       );
```

The complete VHDL code for a 3:8 decoder with data flow modeling using **with-select** statement is given below.

```
library ieee;
use ieee.std_logic_1164.all;

entity decod 3_8 is
  port (En : in bit;
        S: in bit_vector (2 downto 0);
        Y: out bit_vector (7 downto 0)
       );
end decod 3_8;

architecture dat_flow 2 of decod 3_8 is
  signal temp1 : bit_vector (3 downto 0);

 begin
     temp ⇐ En & S;
     with temp select
```

```
    Y ⇐ "00000001" when "1000",
        "00000010" when "1001",
        "00000100" when "1010",
        "00001000" when "1011",
        "00010000" when "1100",
        "00100000" when "1101",
        "01000000" when "1110",
        "10000000" when "1111",
        "00000000" when others;
end dat_flow 2;
```

7.8.6 Behavioural Modeling

The behavioural style of modeling specifies the behaviour of an entity as a set of statements that are executed sequentially in the specified order.

A 'process' statement is a key element in the behavioural modeling. A process statement contains a set of sequential statements which specify the functionality of the model and not the exact structure of it. Process statement is a concurrent statement which executes in parallel with other concurrent statements and other processes.

A process statement has the following syntax:

```
process (signal name, signal name ..... signal name)
    declarations
begin
    sequential statements
end process;
```

The bracket followed by the keyword **process** is referred to as the *sensitivity list* of the process. Whenever one of the signals in the sensitivity list changes, the sequential statements inside the process body are executed in sequence.

The process statement also has a declarative part (before the keyword **begin**) along with the statement part (in between the keywords **begin** and **end** process), similar to an architecture body.

The declarative part may contain one or more of the following declarations:

- type declarations,
- variable declarations,
- constant declarations,
- function definitions, or
- procedure definitions.

7.8.7 Sequential Statements used in Behavioural Modeling

The sequential statements used in behavioural modeling are: **if**, **case**, **loop**, **exit**, **next**, and **wait**. These sequential statements can be used inside a process for signal assignments. The process executes whenever there is an event on signals. When the process executes, all the statements inside the process are executed sequentially, independent of whether there is a change on any signal in its right-hand expression. Contrast this with the execution of concurrent signal assignment statement which gets executed only when an event occurs on a signal in the right-hand side expression.

In the process, all the statements get executed and at the end of the process, their values get updated.

If statement

The **if** statement has the following syntax:

```
if (condition) then
   sequential statement
end if;
```

The **nested if** statements use the clause 'elsif' as shown below.

```
if (condition 1) then
   sequential statement
elsif (condition 2) then
   sequential statement
        :
        :
else
   sequential statement
end if;
```

The conditions are generally **boolean** expressions which are tested for a true or false condition. For example,

```
if ((a = b) and (c = '1')) then
    y ⇐ "0000";
else
    y ⇐ "1111";
end if;
```

The VHDL code for the 3:8 decoder of Example 7.12 using **if** statement is given below.

```
library ieee;
use ieee.std_logic_1164.all;

entity decod 3_8 is
    port (En : in bit;
          S : in bit_vector (2 down to 0);
          Y : out bit_vector (7 down to 0)
         );
end decod 3_8;

architecture behav of decod 3_8 is
  begin
      process (En, S)
    begin
      if En = '1' then
         if (S = "000") then y ⇐ "00000001";
      elsif (S = "001") then y ⇐ "00000010";
      elsif (S = "010") then y ⇐ "00000100";
      elsif (S = "011") then y ⇐ "00001000";
      elsif (S = "100") then y ⇐ "00010000";
      elsif (S = "101") then y ⇐ "00100000";
      elsif (S = "110") then y ⇐ "01000000";
      elsif (S = "111") then y ⇐ "10000000";
```

```
        end if;
      else y ⇐ "00000000";
    end if;
  end process;
end behav;
```

In the above code, process statement is used for which the signals **En** and **S** are in the sensitivity list. Since no declarations other than the parts in the entity are required, the declarative part does not contain any declarations. The statement part contains an **if-then** statement which is a sequential statement used to do conditional signal assignment to the output signal **y**.

The process executes whenever there is an event on signals **En** or **S**. When the process executes, all the statements inside the process are executed sequentially independent of whether there is a change on any signal in its right-hand expression. Contrast this with the execution of concurrent signal assignment statement which gets executed only when an event occurs on a signal in the right-hand side expression. In the process, all the statements get executed and at the end of the process, their values get updated.

CASE statement

A CASE statement is another type of sequential statement with the following format:

```
case (expression) is
  when (choices) ⇒ sequential statement;
    .
    .
  when (choices) ⇒ sequential statements;
end case;
```

The VHDL code for the 3:8 decoder of Example 7.12 using **case** statement is given below.

```
library ieee;
use ieee.std_logic_1164.all;

entity decod 3_8 is
    port (En : in bit;
          S : in bit_vector (2 downto 0);
          Y : out bit_vector (7 downto 0)
         );
end decod 3_8;

architecture behav2 of decod 3_8 is
  begin
    process (En, S)
      begin
        if En = '1' then
          case S is
          when "000" ⇒ y ⇐ "00000001";
          when "001" ⇒ y ⇐ "00000010";
          when "010" ⇒ y ⇐ "00000100";
          when "011" ⇒ y ⇐ "00001000";
          when "100" ⇒ y ⇐ "00010001";
          when "101" ⇒ y ⇐ "00100010";
          when "110" ⇒ y ⇐ "01000100";
          when "111" ⇒ y ⇐ "10001000";
```



```
          end case;
        else
            y ⇐"00000000";
        end if;
    end process;
end behav2;
```

The **case** statement evaluates the given expression, finds a matching value in one of the choices, and executes the corresponding sequential statements. One or more sequential statements can be written for each set of choices.

Loop statement

A **loop** statement is used to iterate through a set of sequential statements. There are three types of iteration schemes.

1. **'for' iteration scheme:**
   The syntax for the 'for' iteration scheme is as follows:

```
for identifier in range loop
        sequential statement
              .
              .
        sequential statement
end loop;
```

   For example,

```
for i in 1 to 7 loop
    y(i) ⇐ '0';
end loop;
```

2. **'while' iteration scheme:**
   The syntax for the 'while' iteration scheme is as follows:

```
while (expression) loop
        sequential statement
              .
              .
        sequential statement
end loop;
```

   For example,

```
while (j < 8 ) loop
      x(j) ⇐ '1';
      j ⇐ j + 1;
end loop;
```

3. **'loop' without any iteration scheme:**
   In this form, no iteration scheme is specified. All the statements in the loop body are repeatedly executed until some other action causes the loop to exit. The action can be caused by exit statement, next statement, or a return statement. The syntax for this statement is as given below:

```
loop
  sequential statement;
          .
          .
  sequential statement;
end loop;
```

Exit statement

This is a sequential statement, used inside a loop. It causes the execution to jump out of the innermost loop or the loop which is specified in the statement.

The syntax for **exit** statement is

```
exit [loop_label] [when condition]
```

If the **loop** level is not specified, then the innermost loop is exited. It is illustrated in the following examples.

```
L1 : loop
   i : = i + 10;
   if i > 100 then
    exit L1;
  endif;
end loop L1;
```

```
L1 : loop
   i : = i + 10;
   exit when i = 100;
end loop L1;
```

Next statement

This is also a sequential statement, used inside a loop with similar syntax like an exit statement

```
next [loop_label] [when condition];
```

Loop level and condition are optional. This statement results in skipping the remaining statements in the current iteration of the specified loop, and the execution resumes with the first statement in the next iteration.

Wait statement

In behavioural modeling, a process statement is used which contains a sensitivity list. The process always suspends after executing the last sequential statement in the process.

An alternative way to suspend the execution of a process is to use a **wait** statement within the process.

There are three basic forms of a **wait** statement:

```
wait on         sensitivity list;
wait until      boolean-expression;
wait for        time-expression;
```

wait on This form of wait statement causes a process to suspend the execution until an event occurs on one or more signals in a group of signals. For example,

```
wait on clk, reset;
```

In this case, an event on any of the signals executes the process with the first statement following the wait statement.

wait until In this case, the process is suspended until the condition specified in the bracket is not satisfied. For example,

```
wait until A = B;
```

In this case, when an event occurs on signal *A* or *B*, the condition is evaluated. If it is true, the process resumes the execution with the first statement following the wait statement, otherwise it suspends again.

wait for In this form of wait statement, the process gets suspended for a time *T*, which is evaluated by a time expression. For example,

```
wait for 20 ns;
```

In this case, the process suspends for 20 ns and resumes its execution from the following statements.

Any process can have a sensitivity list or use **wait** statements, but not both.

7.8.8 Mixed Style of Modeling

When all the three styles of modeling are used within an architecture, it is referred to as the mixed style of modeling. In such an architecture body, the component instantiation statements (as used in structural modeling), concurrent signal assignment statements (as used in data flow modeling), and process statements (as used in behavioural modeling), can be used.

7.8.9 Configurations

There are many different ways in which the operation of a digital circuit is modeled, such as structural, data flow, and behavioural. Hence, there is a possibility of having many architecture bodies for one entity. The VHDL language provides configurations for explicitly associating one of the architecture descriptions with an entity, when it is being used as a component. This process of association is referred to as *binding* an instant of the component. For structural models, configurations can be thought of as the parts list for the model. When the configuration for an entity-architecture combination is compiled into the library, an implantable object is created. There are two types of configurations:

(a) Default configuration, and
(b) Component configuration

Default configurations

The simplest form of explicit configuration is the default configuration. It specifies the configuration name, the entity being configured, and the architecture to be used for the entity.

If the architecture does not contain any blocks or components, then the configuration design unit has the following syntax:

```
Configuration: Configuration name of entity name is
          for architecture name
          end for;
end configuration name;
```

The following example shows the use of default configuration for a 2-bit comparator circuit producing an equality output.

```
library ieee;
use ieee.std_logic_1164.all;
entity comp_2 bit is
    port (A, B : in std_logic_vector (1 downto 0);
               Y : out std_logic
         );
```

```
end comp_2 bit;

architecture A1 of comp_2 bit is
    begin
      Y <= '1' when (A = B) else
           '0';
    end A1;
architecture A2 of comp_2 bit is
      begin
        process (A, B)
          begin
            if (A = B) then
               Y<= '1';
            else
               Y <= '0';
            end if;
          end process;
        end A2;

configuration dat_comp of comp_2 bit is
  for A1
  end for;
end dat_comp;
configuration behav_comp of comp_2 bit is
  for A2
  end for;
end behav_comp;
```

In the above example, the entity contains two 2-bit input vectors A and B, and one output Y. The comparator produces $Y = 1$, if $A = B$. The architecture $A1$ uses data flow style of modeling, while the architecture $A2$ uses behavioural style of modeling.

The first configuration is called dat_comp which binds the architecture $A1$ (data flow) with the entity comp_2 bit to form a simulatable object. The second configuration binds architecture $A2$ (behavioural) with the entity comp_2 bit and forms a simulatable object called behav_comp.

Since the architectures do not contain any components, the configuration area from the **for** clause to the **end for** clause is empty. This configuration is called the default configuration.

Component configuration

These statements are used where the architectures contain instantiated components (structural model).

Two different styles can be used for writing a component configuration for an entity.

1. Lower-level configurations – specifies lower-level configurations for each component.
2. Entity architecture configurations – specifies entity architecture pairs for each component.

Any of the above two styles can be used for writing a component configuration.

Syntax of lower-level configuration

```
configuration configuration_name of entity_name is
  for architecture_name
    for component_instance 1 : component_name use configuration work.
    configuration_name;
    end for;
    for component_instance 2 : component_name use configuration work.
    configuration_name
    end for;
        .
        .
    for component_instance n : component_name use configuration work.
    configuration-name
    end for;
  end for;
end configuration_name;
```

Syntax of entity_architecture configuration

```
configuration configuration_name of entity_name is
  for architecture_name
    for component_instance 1 : component_name use entity work.
    entity_architecture pair
    end for;
    for component_instance 2 : component_name use entity work.
    entity_architecture pair
    end for;
        .
        .
  end for;
  end configuration_name;
```

The following example of a full-adder using half-adder shows the use of component configurations.

```
  library ieee;
  use ieee.std_logic_1164.all;
entity full_adder is
port (ln1, ln2, Cit in : in std_logic;
    Sum, Cout : out std_logic);
end entity full_adder;

architecture structural of full_adder is
    component half_adder is
      port (X, Y: in std_logic;
            Sum, Carry : out std_logic);
    end component half_adder;

    component or _2 is
      port (X, Y : in std_logic;
            Z : out std_logic);
  end component or_2;
    signal S1, S2, S3 : std_logic;
```

– configuration specification

```
for H1 : half_adder use entity WORK.half_adder (behavioural);
for H2 : half_adder use entity WORK.half_adder (structural);
for O1 : or_2 use entity WORK.or_2 (behavioural);

port map (11 ⇒X, 12 ⇒ Y, Z ⇒ z);

  begin  –component instantiation statements
H1 : half_adder port map (X ⇒ ln1, Y ⇒ ln2,
                          Sum ⇒ S1, Carry ⇒ S2);
H2 : half_adder port map (X ⇒ S1,Y ⇒ Cin,
                          Sum ⇒ Sum, Carry ⇒ S2);
O1 : or_2 port map (X ⇒ S2,Y ⇒ S3,Z ⇒ Cout);
end architecture structural;
```

7.8.10 Identifiers

Several identifiers were used in the previous discussion, such as comp_2 bit, full_adder, half_adder, decod 3_8, and mux_4_1. Generally, identifiers are chosen because they are easy to remember.

VHDL specifies a few rules that must be followed while creating identifies. The basic rules that apply for creating identifies are:

- It may contain upper-case and lower-case letters, the digits 0 to 9, and underscore symbol.
- The underscore symbol cannot be used as the first and last character, letter and number must be used in both sides of the underscore.
- Every identifier is a single string of characters, and it may be of any desired length.
- Identifiers are case-insensitive.

Examples of legal identifiers:

DELAY, gate, d4, ab14, Ab5, mux_4, my_circuit_dig

Examples of illegal identifiers:

_ DELAY, ga_ _te, Ab5_, mux_4_

7.8.11 Libraries

Libraries are collections of predefined quantities and procedures that are used by the compiler to interpret the code. The concept of library is very powerful in VHDL. A library is introduced into the VHDL using a statement of the form

```
library library_name;
```

For example, 'ieee' is a standard library in VHDL. It is introduced by the command

```
library ieee;
```

Once a library is introduced, any definition from the library may be used in writing a VHDL code. Library contains the fundamental definitions of type and basic physical units.

The information contained in subgroups is called the *standard*. A particularly useful package is called IEEE 1164, which is based on VHDL language definition. The ieee library and IEEE 1164 package will be initiated in VHDL as

```
library ieee;
use ieee.std_logic_1164.all;
```

7.8.12 Important Data Objects in VHDL

A data object in VHDL holds a value of specific type. It is created by means of an object declaration.

Every data object belongs to one of the following classes.

Signal

Signal objects are used to connect entities together to form models. Signals are a means to communicate dynamic data between entities. Signals can be declared in entity declaration sections, architecture declarations, and package declarations.

The syntax for signal declaration is,

```
signal signal_name : signal_type [: = initial value];
```

The keyword **signal** is followed by one or more signal names of the same signal type specified after a colon. Finally, a signal can contain an initial value which is optional.

Variable

Variables are used for local storage in process statements and subprograms. All assignments to the variables occur immediately.

The syntax for variable declaration is,

```
variable: variable_name : variable_type [: = initial value];
```

The keyword **variable** is followed by one or more variable names of the same variable type specified after a colon, and an optional initial value can be specified.

Constant

Constant objects are the names assigned to specific values of a type. If a model requires a fixed value in a number of instances, a constant value should be used.

The syntax for constant declaration is,

```
Constant: constant_name : type_name [: = value];
```

A constant declared in an architecture can be used by any statement inside the architecture, including a process statement. A constant declared in a process can be used only in a process.

The following section of a code shows different object declaration examples.

```
architecture : decod_arch of decoder is
  SIGNAL A, B, C : std_logic;
  SIGNAL X, Y    : std_logic_vector (2 downto 0);
  SIGNAL N       : integer range 0 to 3;
begin

  process (A, B)
    variable temp1, temp2 : std_logic (: = '0');
```

```
    constant pi : Real (: = 3.1414);
  begin :
  end process
end decod.arch;
```

7.8.13 Important Data Types

Variables, signals, and constants can have any of the predefined VHDL types or a user-defined type. Some of the predefined types are:

bit : 0 or 1
boolean : FALSE or TRUE
integer : an integer in the range $-(2^{31}-1)$ to $+(2^{31}-1)$
real : floating point number in the range –1.0 E38 to +1.0 E38
character : any legal VHDL character including upper and lower case characters and special characters
time : an integer with units fs, ps, ns, us, ms, sec
std_logic : '0', '1', 'U', 'X', 'Z', 'W', 'L', 'H', '–'

A common user-defined type is the enumeration type in which all the values are enumerated. For example,

```
type state is (S0, S1, S2, S3, S4)
signal pr_state, ns_state : state (: = S0);
```

Arrays: The above-mentioned data types in VHDL are scalar types. In order to have arrays in VHDL, we first have to declare an array type and then, declare an array object.

The array type and object declarations have the following general form.

```
type array_type_name is array index_range of element_type;
signal array_name : array_type_name (: = initial values variable)
```

In the preceding declaration, **signal** may be replaced with **or** constant.

7.8.14 VHDL Operators

The predefined VHDL operators can be grouped into seven classes.

1. Binary logical operators—a few examples and their meaning are given in Table 7.8.

Table 7.8 Binary logical operators

Keyword	*Example*	*Meaning*
and	a **and** b	$a \cdot b$
or	a **or** b	$a + b$
nand	a **nand** b	$\overline{a \cdot b}$
nor	a **nor** b	$\overline{a + b}$
xor	a **xor** b	$a \oplus b$
xnor	a **xnor** b	$\overline{a \oplus b}$
not	**not** a	$\overline{a}$

2. Relational operators
   = : Equal to
   /= : Not equal to
   < : Less than
   <= : Less than or equal to
   > : Greater than
   >= : Greater than or equal to
3. Shift operators
   sll : shift left logical filled with 0
   slr : shift right logical filled with 0
   sla : shift left arithmetic filled with right bit
   sra : shift right arithmetic filled with left bit
   rol : rotate left
   ror : rotate right
4. Adding operators
   + : Addition
   – : Subtraction
   & : Concatenation
5. Unary sign operators
   + : Positive
   – : Negative
6. Multiplying operators
   * : Mortification
   / : Division
   mod : Mode
7. Miscellaneous operators
   abs : Absolute
   * * : Exponentiation

To get more information on data types which support these VHDL operators, a standard book on VHDL should be referred.

7.8.15 VHDL Examples

Example 7.14 Write the concurrent VHDL statements for the following Boolean expressions:

1. $Y = \overline{AB} + \overline{AC} + \overline{BC}$
2. $Y = A\overline{B} + \overline{A}C + \overline{BC}$
3. $C_n = \overline{A}_n B_n + \overline{A}_n C_{n-1} + B_n C_n$

Solution

1. VHD statement for the Boolean expression $Y = \overline{AB} + \overline{AC} + \overline{BC}$ is

   $Y \Leftarrow$ **not**(A **and** B) **or not**(A **and** C) **or not**(B **and** C)

2. VHD statement for the Boolean expression $Y = A\overline{B} + \overline{A}C + \overline{BC}$ is

   $Y \Leftarrow$ (A **and** (**not** B)) **or** ((**not** A) **and** C) **or not** (B **and** C)

3. VHD statement for the Boolean expression $C_n = \overline{A}_n B_n + \overline{A}_n C_{n-1} + B_n C_n$ is

   $C_n \Leftarrow$ ((**not** $\overline{A}_n$) **and** B_n) **or** ((**not** $\overline{A}_n$) **and** C_{n-1}) **or** (B_n **and** C_n)

Example 7.15 Write the VHDL code for the Boolean expression of a 2-input NAND gate. Assume the delay of gate is 6 ns.

Solution

```
library ieee;
use ieee.std_logic_1164.all;
entity gate is
```

```
        port (A, B, : in bit;
              Y : out bit
              );
        end gate;
architecture NAN_gate of gate is
  begin
        Y ⇐ A nand B after 6ns;
  end NAN_gate;
```

Example 7.16 Write the VHDL code of a full-subtractor using:

1. Data flow modeling
2. Behavioural modeling

Solution

The truth table of a full-subtractor is as shown below in Table 7.8.

Table 7.8 Full-subtractor truth table

A	*B*	C_{in}	*Difference*	*Borrow*
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

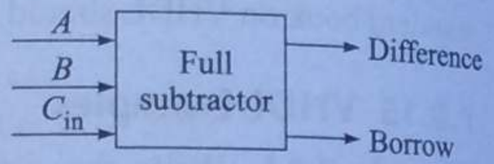


Fig. 7.65 Block diagram of full-subtractor

The equations for difference and borrow are:

$$\text{Difference} = A \oplus B \oplus C_{in}$$

$$\text{Borrow} = \overline{A}B + \overline{A}C_{in} + BC_{in}$$

The entity for the full-subtractor will contain A, B, and C_{in} as the input ports, and output difference and borrow as the output parts. The corresponding block diagram is shown in Fig. 7.65. The VHDL code for full-subtractor is given below.

1. Data flow modeling

```
library ieee;
use ieee.std_logic_1164.all;
  entity full_sub is
       port (A, B, Cin : in bit;
             diff, borrow : out bit
             );
  end full_sub;
architecture dat_full_sub of full_sub is
     begin
       diff   ⇐ (A xor B) xor Cin;
       borrow ⇐ (not A and B) or (not A and Cin) or (B and Cin);
     end dat_full_sub;
```


2. Behavioural modeling

```
library ieee;
use ieee.std_logic_1164.all;
entity full_sub is
  port   (A, B, Cin : in bit;
          diff, borrow : out bit
          );
end full_sub;
architecture behav_full_sub of full_sub is
  begin
    process (A, B, Cin)
  begin
    case (A & B & Cin) is
          when "000" ⇒ diff ⇐ 0; borrow ⇐ 0;
          when "001" ⇒ diff ⇐ 1; borrow ⇐ 1;
          when "010" ⇒ diff ⇐ 1; borrow ⇐ 1;
          when "011" ⇒ diff ⇐ 0; borrow ⇐ 1;
          when "100" ⇒ diff ⇐ 1; borrow ⇐ 0;
          when "101" ⇒ diff ⇐ 0; borrow ⇐ 0;
          when "110" ⇒ diff ⇐ 0; borrow ⇐ 0;
          when "111" ⇒ diff ⇐ 1; borrow ⇐ 1;
    end case;
  end process;
end behav_full_sub;
```

Example 7.17 Write the VHDL code (structural model) for a 4-bit subtractor using the full-subtractor designed in Example 7.14 as a component.

Solution

A 4-bit subtractor is implemented using four full-subtractors, as shown in Fig. 7.66.

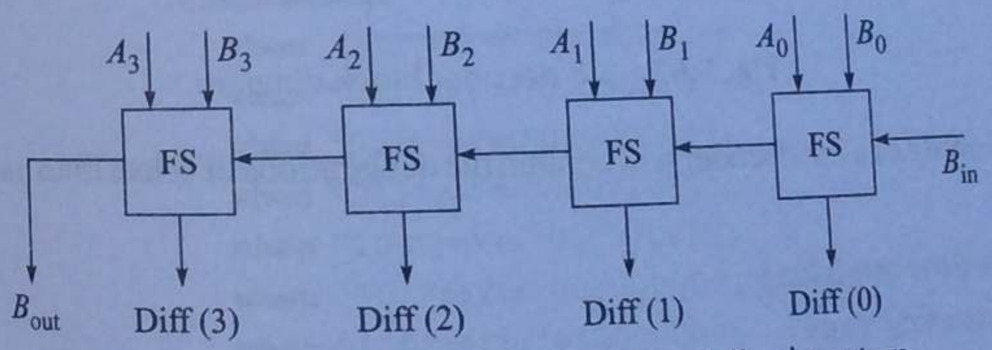


Fig. 7.66 4-bit subtractor using four full-subtractors

The entity for a 4-bit subtractor is as follows:

```
entity sub_4 bit is
  port (A, B : in bit_vector (3 downto 0);
        Bin : in bit;
        Diff : out bit_vector (3 downto 0);
        Bout : out bit
        );
end sub_4 bit;
```

The architecture of a 4-bit subtractor will contain the declaration of full-subtractor as a component in the declarative section. The architecture using structural modeling is given below.

```
architecture struct_sub_4 bit of sub_4 bit is
  component full_sub
    port (A, B, Cin : in bit;
          diff, borrow : out bit
          );
  end component;
  signal temp : bit_vector (3 downto 1);
begin
  FS0 : full_sub port map (A(0), B(0), Bin , diff (0), temp(1));
  FS1 : full_sub port map (A(1), B(1), temp(1), diff(1), temp(2));
  FS2 : full_sub port map (A(2), B(2), temp(2), diff(2), temp(3));
  FS3 : full_sub port map (A(3), B(3), temp(3), diff(3), Bout));
end struct_sub_4 bit;
```

Example 7.18 Write the VHDL code for a 3:8 decoder using

1. Data flow modeling, and
2. Behavioural modeling

Solution

The block diagram of a 3:8 decoder is shown in Fig. 7.67.

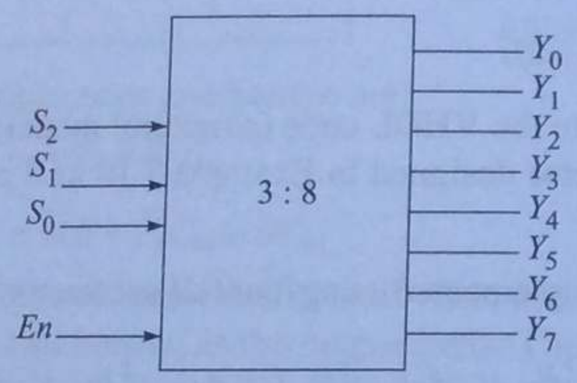


Fig. 7.67 3:8 decoder block diagram

The entity of a 3:8 decoder will contain the declarations of select lines and output lines.

1. Data flow modeling

```
library ieee;
use ieee.std_logic_1164.all;
  entity dec_3_8 is
    port (sel : in BIT_vector(2 downto 0);
          en : in BIT;
           y : out BIT_VECTOR (7 downto 0)
         );
  end dec_3_8;

  architecture datflow_dec of dec_3_8 is
```



```
signal temp : bit_vector (3 downto 0);

begin
temp <= en & sel;
with temp select
      y <= "00000001" when "1000",
           "00000010" when "1001",
           "00000100" when "1010",
           "00001000" when "1011",
           "00010000" when "1100",
           "00100000" when "1101",
           "01000000" when "1110",
           "10000000" when "1111",
           "00000000" when others;
end datflow_dec;
```

2. Behavioural modeling

```
library ieee;
use ieee.std_logic_1164.all;
  entity dec_3_8 is
    port ( sel : in BIT_vector(2 downto 0);
           en : in BIT;
            y : out BIT_VECTOR (7 downto 0)
         );
end dec_3_8;
architecture behav_dec of dec_3_8 is
  begin
    P1 : process (sel, en)
begin
   if (en='1') then
     case sel is
        when "000"⇒Y⇐"00000001";
        when "001"⇒Y⇐"00000010";
        when "010"⇒Y⇐"00000100";
        when "011"⇒Y⇐"00001000";
        when "100"⇒Y⇐"00010000";
        when "101"⇒Y⇐"00100000";
        when "110"⇒Y⇐"01000000";
        when "111"⇒Y⇐"10000000";
   end case;
else
   y⇐"00000000";
end if;
end process p1;
  << enter your statements here>>
end behav_dec;
```

Example 7.19 Write the VHDL code for a 4-bit comparator using data flow modeling.

Solution
A 4-bit comparator will have 3 outputs which are $(A = B)$, $(A < B)$, and $(A > B)$. The block diagram for the comparator is as shown in Fig. 7.68.

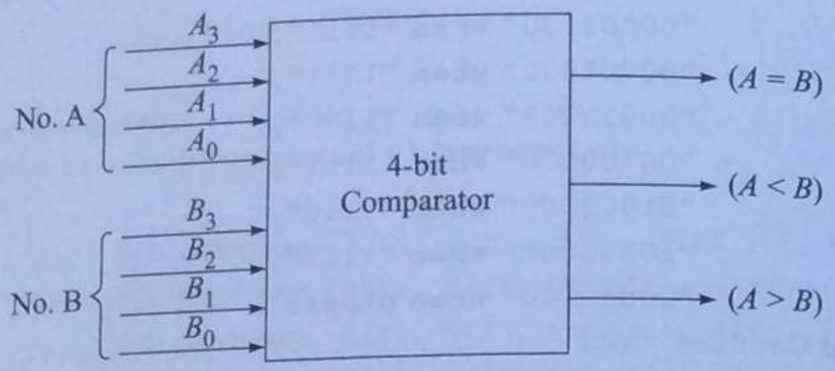


Fig. 7.68 4-bit comparator block diagram

The following VHDL code represents a 4-bit comparator.

```
library ieee;
use ieee.std_logic_1164.all;
  entity cmp_4 bit is
    port ( a : in BIT_VECTOR (3 downto 0);
           b : in BIT_VECTOR (3 downto 0);
           eq : out BIT;
           less : out BIT;
           greater : out BIT
         );
  end cmp_4 bit;

architecture cmp_4 bit_arch of cmp_4 bit is
  begin
  eq<='1' when (a=b) else '0';
  less<='1' when (a<b) else '0';
  greater<='1' when (a>b) else '0';
end cmp_4 bit_arch;
```

Example 7.20 Write the VHDL code for a positive edge-triggered D flip-flop.
(a) without reset and preset inputs,
(b) with reset and preset inputs.
Assume the delay of flip-flop is 5 ns.

Solution
(a) The basic D flip-flop is shown in Fig. 7.69. The truth table is shown in Table 7.9.

Table 7.9 Truth table of D-FF

CLK	*D*	Q_{n+1}
1	1	1
1	0	0
0	X	Q_n

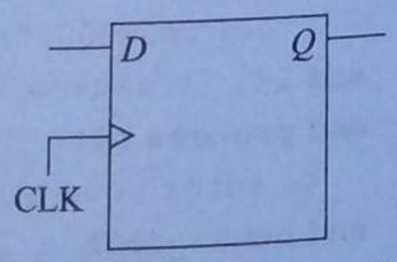


Fig. 7.69 Positive edge-triggered D-FF

The following code is for a positive edge-triggered flip-flop. The process statement contains a checking of condition (CLK'event and CLK = '1') which is equivalent to having a +ve going transition on the clock. CLK' event is referred to as an attribute of signal CLK. Hence whenever there is an event on CLK, CLK'event = True.

```
library ieee;
use ieee.std_logic_1164.all;

entity D_ff is
      port (Din, CLK : in std_logic;
            Q, Q̄ : out std_logic);

end D-ff;
architecture d_ff_arch of d_ff is
  begin
    process (CLK)
    begin
      if (CLK'event and CLK = '1') then
            Q ⇐ Din after 5ns;
            Q̄ ⇐ Din after 5ns;
      else
            Q ⇐ Q;
            Q̄ ⇐ Q̄
      end if;
    end process;
  end D_ff_arch;
```

(b) The following code shows a D-flip-flop with asynchronous reset and preset inputs, as shown in Fig. 7.70.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity D_ff is
        port (Din, reset, preset ,CLK : in STD_LOGIC;
              Q, Q̄ : out STD_LOGIC
             );

end D_ff;
architecture D_arch of D_ff is
  begin
      process(reset, CLK, preset)
      begin
        if(reset = '1') then
            Q ⇐ '0';
            Q̄ ⇐ '1';
```

```
        elsif (preset= '1') then
                Q ⇐ '1';
                Q̄ ⇐ '0';
        elsif(CLK'event and CLK= '1') then
            Q ⇐ Din after 5ns;
            Q̄ ⇐ not Din after 5ns;
        else
          Q ⇐ Q;
          Q̄ ⇐ Q̄;
        end if;
        end if;
    end if;
    end process;
end architecture;
```

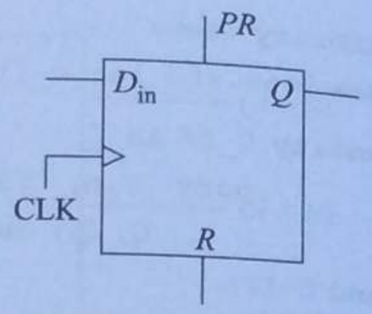


Fig. 7.70 D flip-flop with reset and preset inputs

Example 7.21 Write the VHDL code for a level edge-triggered J-K flip-flop

(a) without reset and preset inputs,

(b) with reset and preset inputs.

Assume the delay of flip-flop is 5 ns.

Solution

```
library ieee;
use ieee.std_logic_1164.all;
entity JK_FF is
    port   (J, K, CLK : in bit;
            Q, Q̄ : out bit
            );
  end JK_FF;
architecture behav_ JK_FF of JK_FF is
    begin
      process (J, K, CLK)
    begin
if(CLK= '1')
      case (J & K) is
            when "00" ⇒ Q ⇐ Q after 5ns; Q̄ ⇐ Q̄ after 5ns;
            when "01" ⇒ Q ⇐ 0 after 5ns; Q̄ ⇐ 1 after 5ns;
            when "10" ⇒ Q ⇐ 1 after 5ns; Q̄ ⇐ 0 after 5ns;
            when "11" ⇒ Q ⇐ Q̄ after 5ns; Q̄ ⇐ Q after 5ns;
      end case;
else
    Q ⇐ Q;
    Q̄ ⇐ Q̄;
endif;
end process;
```



```
library ieee;
use ieee.std_logic_1164.all;
entity JK_FF is
      port (J,K, reset , preset, CLK : in std_logic;
            Q, Q̄ : out std_logic);
end JK-FF;
architecture JK_FF_arch of JK_FF_is
begin
      process (J,K, reset, preset, CLK)
      begin
        if(reset = '1') then
            Q ⇐ '0';
            Q̄ ⇐ '1';
        elsif (preset= '1') then
            Q ⇐ '1';
            Q̄ ⇐ '0';
      elsif(CLK= '1') then
        end if;
      begin
        if (CLK = '1') then
            case (J & K) is
                when "00" ⇒ Q ⇐ Q after 5ns; Q̄ ⇐ Q̄ after 5ns;
                when "01" ⇒ Q ⇐ 0 after 5ns; Q̄ ⇐ 1 after 5ns;
                when "10" ⇒ Q ⇐ 1 after 5ns; Q̄ ⇐ 0 after 5ns;
                when "11" ⇒ Q ⇐ Q̄ after 5ns; Q̄ ⇐ Q after 5ns;
            end case;
        else
            Q ⇐ Q;
            Q̄ ⇐ Q̄;
        end if;
      end process;
end JK_FF_arch;
```

Example 7.22 Write the VHDL code for a presettable 4-bit up-down counter having an asynchronous reset input.

Solution

The counter block diagram of a presettable 4-bit up-down counter is represented in Fig. 7.71, where the counter has 4-input lines for presetting the count, a control line preset for loading this count synchronously, a reset line, a mode control input (U/D), and four-output lines.

The following code implements this 4-bit U/D counter:

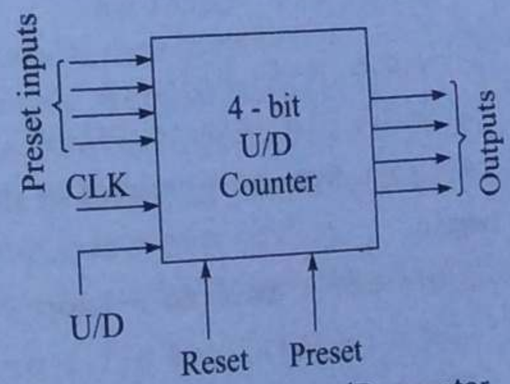


Fig. 7.71 4-bit U/D counter

```
library IEEE;
use IEEE.std_logic_1164.all;
entity spudc is
      port (CLK, reset, preset, ud:in STD_LOGIC;
           Cout:out INTEGER range 0 to 15
           );
end spudc;
architecture spudc_arch of spudc is
  begin
    process (CLK, preset, reset, cnt)
    begin
    if (reset= '1') then
      Cout⇐0;
    elsif (CLK='1' and CLK'event) then
      if( preset='1') then
        Cout⇐cnt;
      else
      if (ud='1') then
        Cout⇐Cout + 1;
      else
          Cout⇐Cout - 1;
        end if;
      end if;
    end if;
  end process;
  end architecture;
```

Example 7.23 Write a VHDL code for the function that adds two 4-bit vectors and a carry and returns a 5-bit sum.

Solution

```
function add4 (A,B: bit_vector(3 downto 0); carry: bit)
  return bit_vector is
variable Cout: bit;
variable Cin: bit := carry;
variable Sum: bit_vector(4 downto 0):="00000";
begin
loop1: for i in 0 to 3 loop
  Cout := (A(i) and B(i)) or (A(i) and Cin) or (B(i) and Cin);
  Sum(i) := A(i) xor B(i) xor Cin;
  Cin := Cout;
```



```
end loop loop1;
Sum(4) := Cout;
return Sum;
end add4;
```

Example 7.24 Write the VHDL code for a procedure that adds two n-bit vectors and a carry and returns an n-bit sum and a carry. Add1 and Add2 are assumed to be of the same length and dimensioned $(n - 1)$ down to 0.

Solution

```
procedure Addvec
  Add1,Add2: in bit_vector;
  Cin: in bit;
  signal Sum: out bit_vector;
  signal Cout: out bit;
  n:in positive) is
  variable C: bit;
begin
  C := Cin;
  for i in 0 to n-1 loop
    Sum(i) ⇐ Add1(i) xor Add2(i) xor C;
    C := (Add1(i) and Add2(i)) or (Add1(i) and C) or (Add2(i) and C);
  end loop;
  Cout ⇐ C;
end Addvec;
```

SUMMARY

In this chapter, we discussed advance methods for describing and designing the sequential circuits. These are: (a) algorithmic state machines, (b) register transfer language, and (c) VHDL.

Algorithmic state machines

ASM charts are suitable where the system contains more number of input and output variables. ASM notation uses three basic symbols. We have discussed these symbols, their meaning, and use in drawing ASM charts. We have also discussed two important methods of designing the sequential circuits described by ASM charts: traditional method using flip-flops and gates and multiplexer control method.

The control using programmable logic devices can be studied after grasping PLD architecture and design methods.

Register transfer language

RTL notation is preferred where the control and data sections of a system are to be designed systematically. RTL notation allows us to describe various register transfer

operations, logical operations, and branching (conditional transfer operations). We have also studied the implementation methodology of data and control sections described by RTL code.

VHDL

In this section, we discussed the basic modeling styles of designing a digital system using VHDL. We have also discussed the important constructs of the language and the important concurrent and sequential statements used in VHDL. Further, the section describes the configurations used in VHDL. Finally, the important data objects and their subtypes along with the important VHDL operators are mentioned. It helps the students to write VHDL codes for simple combinational and sequential circuits.

KEY TERMS AND DEFINITIONS

Algorithmic state machines (ASM) It is a sequential circuit, used to control a digital system that carries out a step-by-step procedure or algorithm.

ASM chart It is a superior notation, especially while designing synchronous circuits. As the name suggests, all the controllers are the state machines into which the algorithms are translated. An ASM chart is basically composed of three basic elements: state box, decision box, and conditional box.

State box Any state in the control sequence is indicated by a state box. It is rectangular in shape within which the output signals, which are generated or the register operation which should be carried out in that state, are written. The name of the state and the binary assignment of the steps are written at the upper left and right corner of the state box.

Decision box It is a diamond-shaped box with two or more exit paths. It expresses the input conditions which decide the state change. The input condition, which is to be tested, is written inside the box. Generally, the condition written inside the box is a Boolean expression that is evaluated to determine which branch to take.

Conditional box It is an oval-shaped block in the ASM notation which is used to represent the output produced during a state, only when a certain input condition is satisfied. Hence, the input path to the conditional box must always come from one of the exit paths of a decision box.

ASM block The ASM chart is constructed from an ASM block. An ASM block is a structure which represents one state. An ASM block has one entrance path and one or more exit paths.

RTL RTL notation is preferred where the control and data sections of a system are to be designed systematically. It allows us to describe various register transfer operations, logical operations, and branching (conditional transfer operations).

VHDL It is a hardware description language which is used to describe the structure and behaviour of a digital system. VHDL stands for VHSIC Hardware Description Language, where VHSIC in turn stands for Very High Speed Integrated Circuits.

EXERCISE

Algorithmic state machines

1. Explain the various symbols used in drawing an ASM chart of a digital system.
2. Explain the relationship of clock timing and the ASM chart with suitable examples.
3. With the help of a neat diagram, explain the difference between the output generated in a state box and a conditional box.
4. What is an ASM action block? Explain with a suitable example.
5. Show the eight exit paths of an ASM chart that check the conditions of three input variables, A, B, and C.
6. With a suitable example, explain the multiplexer control method of ASM implementation.
7. A circuit implementation of an ASM chart contains three flip-flops. What should be the configuration of the multiplexer used? How many multiplexers will be required for the circuit?

Problems

1. Draw an ASM chart for a modulo-6 counter with a reset input.
2. Draw an ASM chart for the following description. The circuit has three input variables (A, B, C), four outputs (W, X, Y, Z), and two exit paths.

   For this circuit, output Z is always 1, and W is 1 if and only if A and B both are 1. If $C = 1$ and $A = 0$, then $Y = 1$, and exit path 1 is taken. If $C = 0$ or $A = 1$, then $X = 1$ and exit path 2 is taken.
3. Draw an ASM chart of a sequence detector which detects the sequence 110. The output Z becomes 1 along with the third correct bit of the sequence.
4. Draw an ASM chart for the circuit shown in Fig. 7.72.

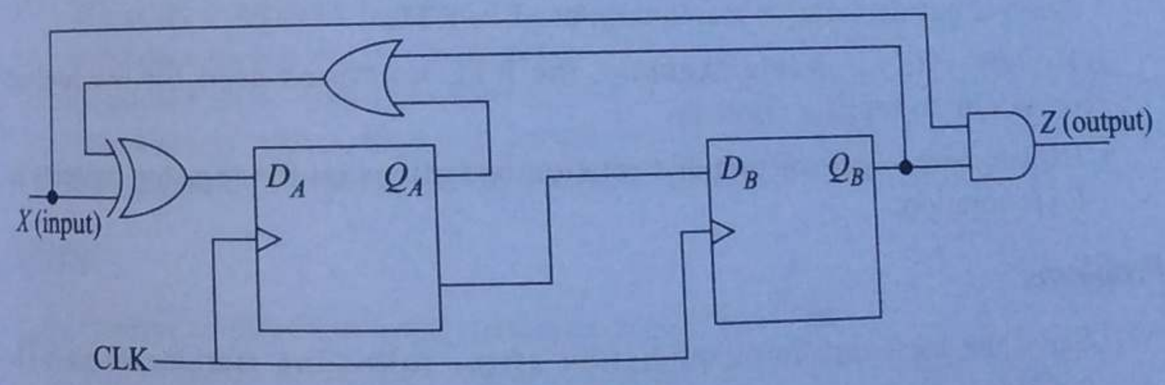


Fig. 7.72

5. Draw an ASM chart and design a circuit using D flip-flops and gates.

   The circuit has a control input X, clock, and outputs A and B. If $X = 1$, on every rising edge of the clock, the code of BA changes from $00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00$ and repeats. If $X = 0$, the circuit holds the present state.
6. Design a synchronous controller having the following ASM chart (shown in Fig. 7.73) using D flip-flops and suitable gates.

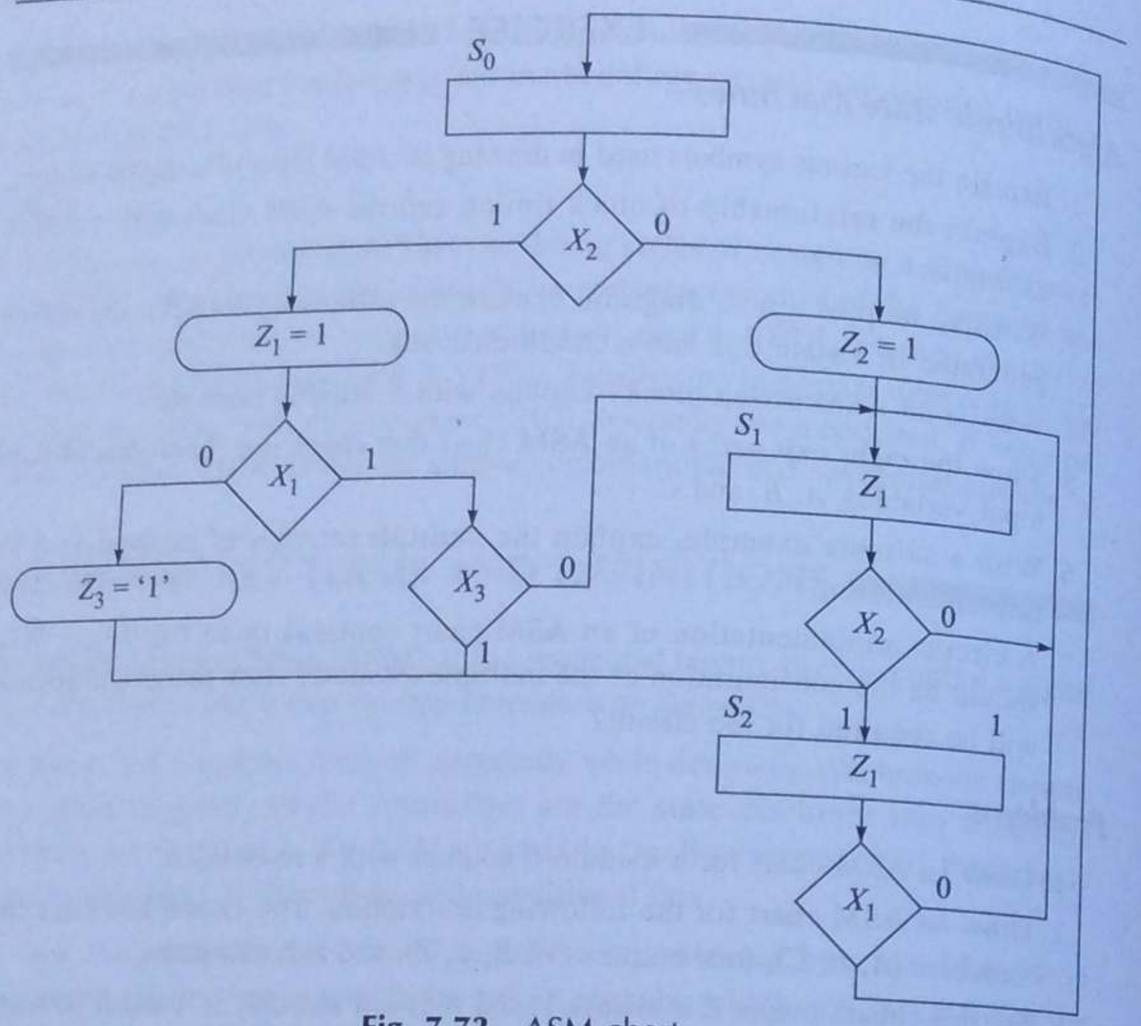


Fig. 7.73 ASM chart

Register transfer language

1. Explain the RTL notation used for register transfer operation, with a suitable example.
2. Explain the unconditional and conditional branch statements in RTL notation.
3. List various declarative statements used in RTL.
4. Explain with a suitable example, the RTL notations used for the logical operation on register contents.
5. Explain how to get shifting and rotation operations on the register contents in RTL notation.

Problems

1. Show the hardware implementation of the following statements in RTL notations:

   (a) $Y \leftarrow X \vee A$

   (b) $B \leftarrow C[0], C[1], C[2], C[3]$

   (c) $Z = Y$

   (d) $D \leftarrow \overline{A}$

   Assume all the registers to be 4-bit.

2. Write the RTL program for a system with 2-bit registers, A, B, and C. Consider X as a 2-bit input line, Y as a 1-bit input line, and Z as a 2-bit output line.

   For this system, the following sequence should take place. The data on X-lines have to be loaded into A-register. If the data on Y-line is 0, then the data in A-register should be complemented and stored in C-register. If the data on Y-line is 1, the data in A-register is shifted to B-register, but one-bit and the data in A and B is ORed and stored in C-register. Finally, the data in C-register is placed on Z-lines.

3. Give the logic diagram and hardware realization of data section and control section for the following RTL description.

```
module  : program_1
memory  : a[2], b[2], c[2]
inputs  : x[2]
outputs : z[2]
  1. B ← X
  2. C ← Ā
  3. A ← B ∧ C
  4. Z = A
END SEQUENCE
```

4. Construct a logic diagram of control and data units specified by the following RTL description, where a and b are FFs, x is an input, and Z is a single output line.

   1. $a \leftarrow x \vee b$
   2. $b \leftarrow x$
      $\rightarrow (a, \overline{a})/(1, 3)$
   3. $Z = 1, b \leftarrow x \oplus b$
      $\rightarrow (1)$

5. Write an RTL program for the following description.

   When the controller receives the "data_out" signal from the processor, it places 4-bit data on the I/O bus data lines. These four bits form a BCD digit which is displayed on a 7-segment display unit. Similarly, when the controller receives the "data-in" signal from the processor, the status of the four switches within the controller is sent to the processor via the same four lines.

VHDL

1. Explain in brief the three modeling styles in VHDL.
2. With suitable examples, explain the conditional signal assignments in VHDL.
3. Explain in brief the following sequential statements in VHDL

   (a) if-then-else (b) case-when (c) for-loop

   (d) Exit (e) Next (f) Wait

4. What are the various classes and types of data objects in VHDL?
5. List the various VHDL operators.
6. What is meant by 'Configuration' in VHDL?
7. Explain the syntax of process statement in VHDL and its importance.
8. What is 'component instantiation' in VHDL? Explain with a suitable example.

Problems

1. Write VHDL codes for the following combinational circuits:
   (a) BCD-7 segment decoder
   (b) 3-bit binary-to-gray code converter using 3:8 decoder as a component
   (c) 4-bit BCD adder using 4-bit binary adder as a component
   (d) 8:1 line multiplexer
   (e) 16:1 multiplexer using 8:1 multiplexer as a component
2. Write VHDL codes for the following sequential circuits:
   (a) Master-slave JK flip-flop using NAND gates as components
   (b) Master-slave JK flip-flop in behavioural modeling
   (c) 3-bit up-down counter (structural and behavioural)
   (d) 3-bit shift register (structural)

Chapter 8

PROGRAMMABLE LOGIC DEVICES

Chapter Outline

- Basic concepts of PLA, PAL, and GAL
- Implementation of various combinational and sequential circuits using programmable logic devices
- Internal architecture of advanced devices like FPGA, CPLD, and ASIC

8.1 INTRODUCTION

Programmable logic devices are a group of integrated circuits that can be configured to perform logic functions. The term *programmable* means that the function may be specified after the chip has been manufactured. As we know, it is always possible to express any combinational logic function in sum-of-products form. Programmable logic devices provide us the facility to create these sum and product terms with the help of AND and OR matrices.

PLD ICs come under the LSI (Large Scale Integration) category. PLD designs typically have lesser power consumption, fewer IC count, and are more reliable than discrete logic designs. There are several types of PLD ICs from different vendors. Generally, PLDs may be classified depending upon the programmability of the AND and OR arrays. PLDs with programmable AND and fixed OR arrays are called Programmable Arrays Logic (PAL). PLDs with programmable AND and OR arrays are called Programmable Logic Array (PLA). This chapter covers PLA, PAL, Generic Array Logic (GAL), Complex Programmable Logic Device (CPLD), and Field Programmable Gate Arrays (FPGAs).

8.2 PROGRAMMABLE LOGIC ARRAY

Historically, the first PLDs were PLAs. A programmable logic array device contains an array of AND and OR functions whose configuration is done by the user. It typically consists of a set of fusible links which establish or remove the contact of a literal in the AND function or contact of a product term in the OR function. Hence, PLAs are those PLDs in which both AND array and OR array are programmable.

The PLA is used to implement a complex combinational circuit. In VLSI design, PLAs are used because it reduces the space requirement. Several PLAs include storage elements like flip-flops on a silicon chip, hence it can also be used for sequential applications.

8.2.1 Block diagram of PLA

PLA is a type of fixed architecture logic device with programmable AND and OR gates. The block diagram of PLA is as shown in Fig. 8.1.

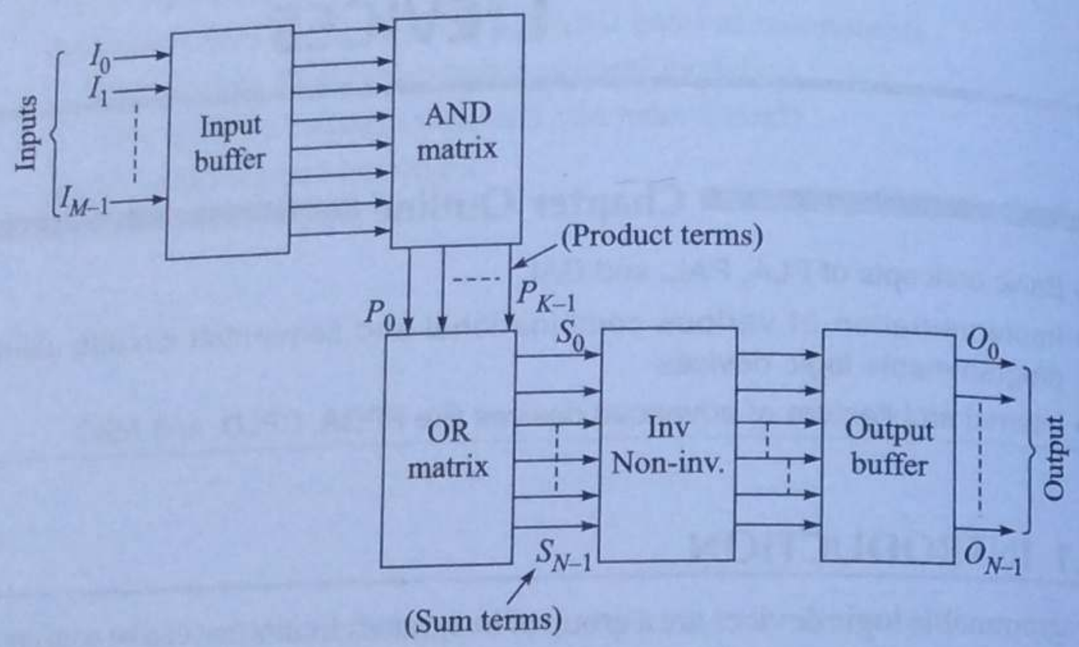


Fig. 8.1 Block diagram of PLA

The block diagram of PLA, shown in Fig. 8.1, has input buffers, AND gates array (matrix), OR gates array (matrix), invert/non-invert matrix, and output buffers. It has M inputs (I_0 through I_{M-1}) and N outputs (O_0 through O_{N-1}).

An AND gate matrix produces K product terms (P_0 through P_{K-1}) and an OR gate matrix produces N min terms (S_0 through S_{N-1}). Such a PLA can be described as an ($M \times N$) PLA with K product terms. In general, K is far less than the number of minterms (2^M), unlike in case of PROM.

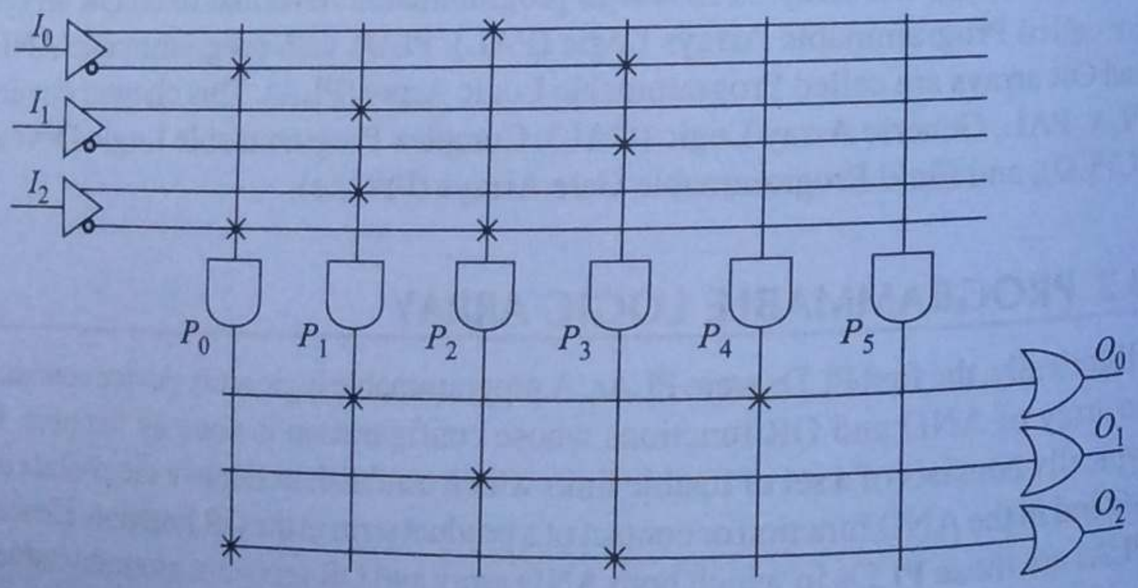


Fig. 8.2 PLA with 3 inputs, 3 outputs, and 6 product terms

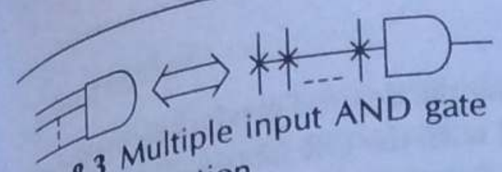


Fig. 8.3 Multiple input AND gate representation

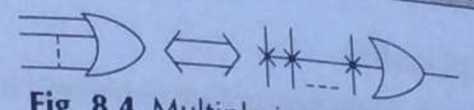


Fig. 8.4 Multiple input OR gate representation

Figure 8.2 shows a small PLA with three inputs, six AND gates, and three outputs. Each input of AND gets connected to both a true and a complemented version of the signal which is produced by a buffer through fusible links. The output of AND gates are connected to each input of OR gates through fusible links. The connections in the array are indicated by X's. The device is programmed by retaining the connections which are needed and removing the rest, since they are made up of fusible links. This construction is similar in both AND and OR array.

The overall arrangement in a PLA can be implemented with the help of input buffer, AND array, OR array, inverting/non-inverting matrix, and output buffer.

Input buffer

Input buffer circuits are provided at the input side to avoid loading the source which drive the inputs. It generates both inverted as well as non-inverted input at the output, as shown in Fig. 8.5.

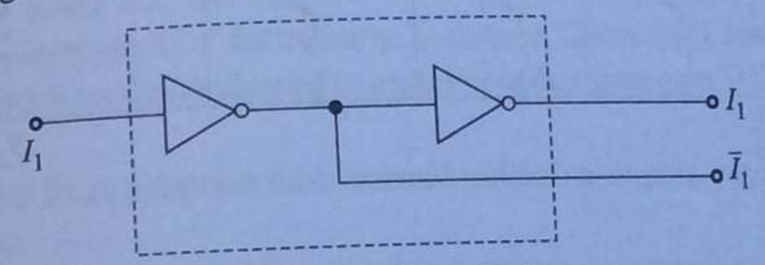


Fig. 8.5 Input buffer

AND matrix

Product terms in the PLA are generated using AND matrix. There are fusible links connected in series with all the diodes, as shown in Fig. 8.6. While programming the PLA, the required product terms are generated by fusing the unwanted links.

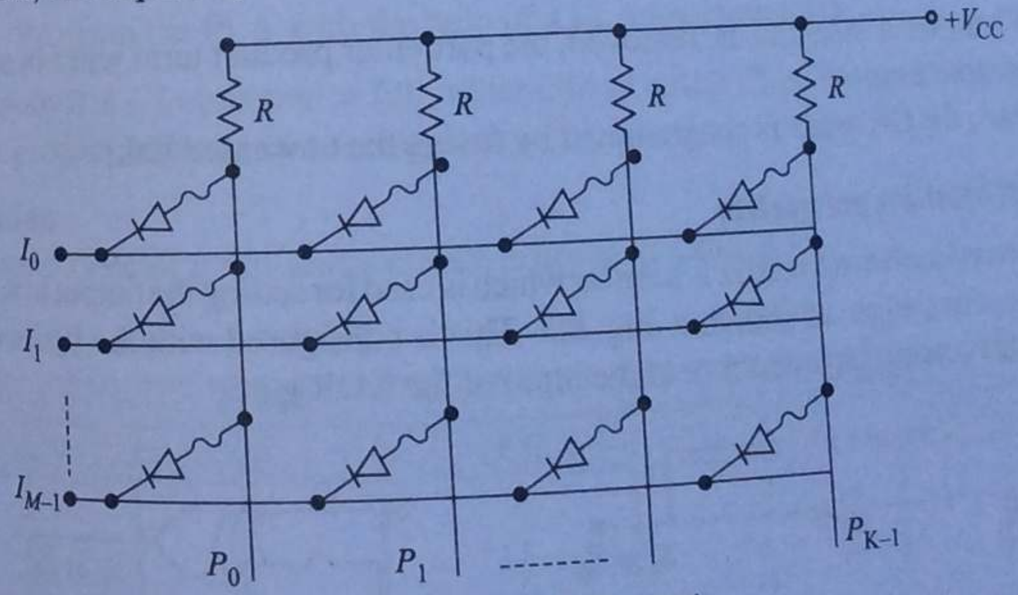


Fig. 8.6 AND matrix

Given below is the equation for each product term, by considering all the fusible links to be intact. (8.1)

$$P_{K-1} = I_0.I_1.I_2...I_{M-1}$$

OR matrix

In a PLA, the outputs are generated by ORing the desired product terms forming the sum terms. These sum terms are formed with the OR matrix. The OR matrix using Resistor Transistor Logic is shown in Fig. 8.7.

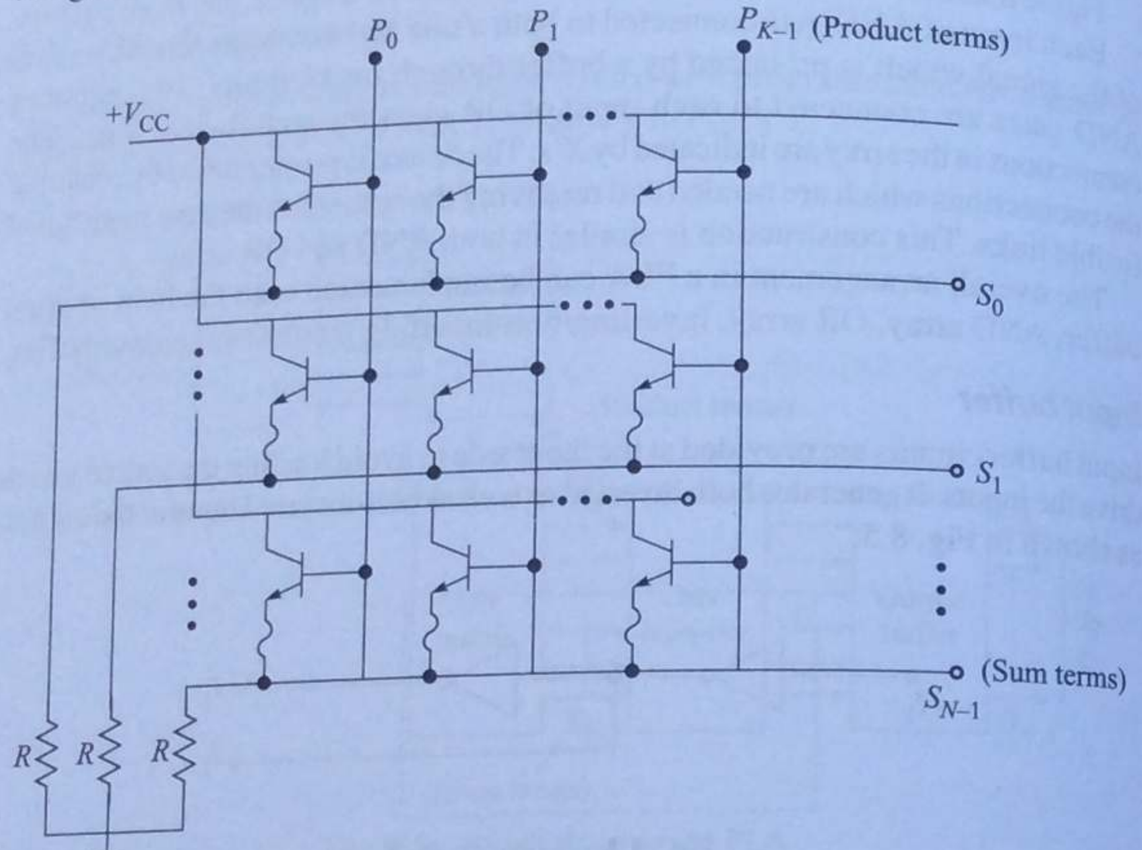


Fig. 8.7 OR matrix

From Fig. 8.7, it can be seen that if none of the fuse links in the emitter of transistor is blown off, then the sum term S_0 is given by the equation

$$S_0 = P_0 + P_1 + P_2 \ldots + P_{K-1} \tag{8.2}$$

Whenever a fuse link is removed, the particular product term will not appear in the sum term.

Thus, the OR array is programmed by fusing the unwanted links.

Invert/non-invert matrix

The invert/non-invert matrix is a buffer which is used for setting the output to active-low or active-high, as shown in Fig. 8.8. This is configured with the fusible link provided across the inverter or at the input of the XOR gate.

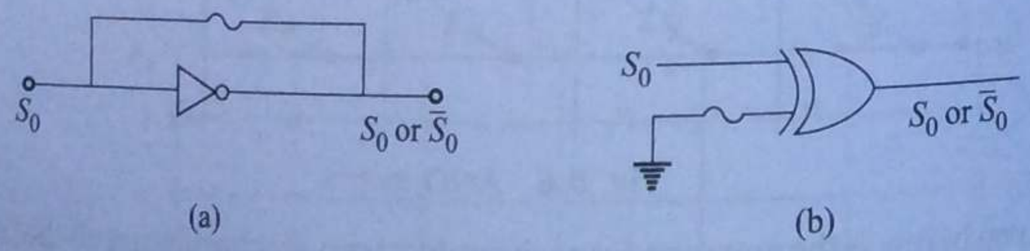


Fig. 8.8 Invert/non-invert matrix

Output buffer

The output buffer circuit is provided at the output side to increase the driving capability of PLA. The output can be totem-pole, open collector, or tri-state. Figure 8.9 shows the output buffer arrangement.

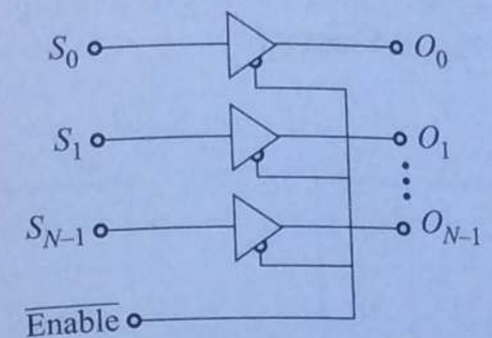


Fig. 8.9 Output buffer

8.2.2 Combinational Logic Design using PLA

PLAs are widely used for implementing combinational logic circuits in order to replace the discrete ICs used, like AND, OR, and Inverter. Since PLA has a programmable AND and OR array, it is beneficial to minimize the given logic function to a minimum SOP form using K-map or Quine-McCluskey method.

The steps which are then followed to implement this minimum SOP form function are:

1. Prepare a PLA program table format indicating inputs, product terms, and outputs.
2. Mark the input connections of AND matrix to generate the required product terms.
3. Mark the input connections of OR matrix to generate the required output (sum) terms.
4. Mark the connections required for invert/non-invert matrix for setting an active high or active low output.
5. Program the PLA with the help of a PLA program table.

Example 8.1 Implement a full-adder circuit using PLA having three inputs, eight product terms, and two outputs.

Solution

The truth table of a full-adder circuit is given in Table 8.1.

Table 8.1 Truth table of a full-adder

Inputs			*Outputs*	
A	*B*	C_{in}	*S*	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

From the truth table, the outputs S (sum) and C_{out} (carry) can be represented as:

$$S = \Sigma m\,(1, 2, 4, 7) \tag{8.3}$$

$$C_{out} = \Sigma m\,(3, 5, 6, 7) \tag{8.4}$$

The K-maps for the outputs S and C_{out} are given in Fig. 8.10 and the minimized expressions are given in Eqs. 8.5 and 8.6.

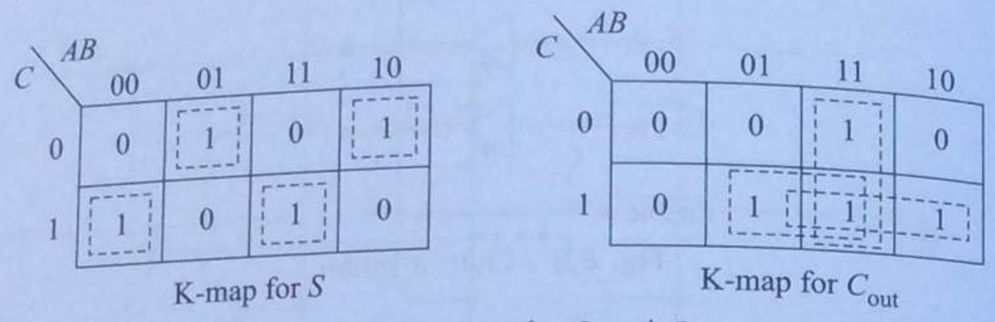


Fig. 8.10 K-maps for S and C_{out}

$$S = \overline{A}B\overline{C}_{in} + \overline{A}\,\overline{B}C_{in} + A\overline{B}\,\overline{C}_{in} + ABC_{in} \tag{8.5}$$

$$C_{out} = AB + BC_{in} + AC_{in} \tag{8.6}$$

From Eqs. 8.5 and 8.6, it can be seen that there are seven distinct product terms and two sum terms. The PLA program table shown in Table 8.2 contains three columns, specifying product terms, inputs, and outputs. In the first column, all the product terms are listed serially. The second column specifies the input conditions (logic 1 or logic 0 or –), required for the AND matrix for the corresponding product term. The third column specifies the connection of the product term in the OR matrix, for a corresponding output. Logic 1 stands for the connection and '–' stands for the absence of the product term in the corresponding output. Below each output column, we write a T (for true) for an active high output or a C (for complement) for an active low output. This is required to invert/non-invert the matrix connections.

Table 8.2 PLA program table for Example 8.1

	Product terms	*Inputs*			*Outputs*	
		A	*B*	C_{in}	*S*	C_{out}
1	$\overline{A}B\overline{C}_{in}$	0	1	0	1	–
2	$\overline{A}\,\overline{B}C_{in}$	0	0	1	1	–
3	$A\overline{B}\,\overline{C}_{in}$	1	0	0	1	–
4	ABC_{in}	1	1	1	1	–
5	AB	1	1	–	–	1
6	BC_{in}	–	1	1	–	1
7	AC_{in}	1	–	1	–	1
					T	*T*

Figure 8.11 shows the diagram of a (3 × 2) PLA with 8 product terms programmed according to Table 8.2.

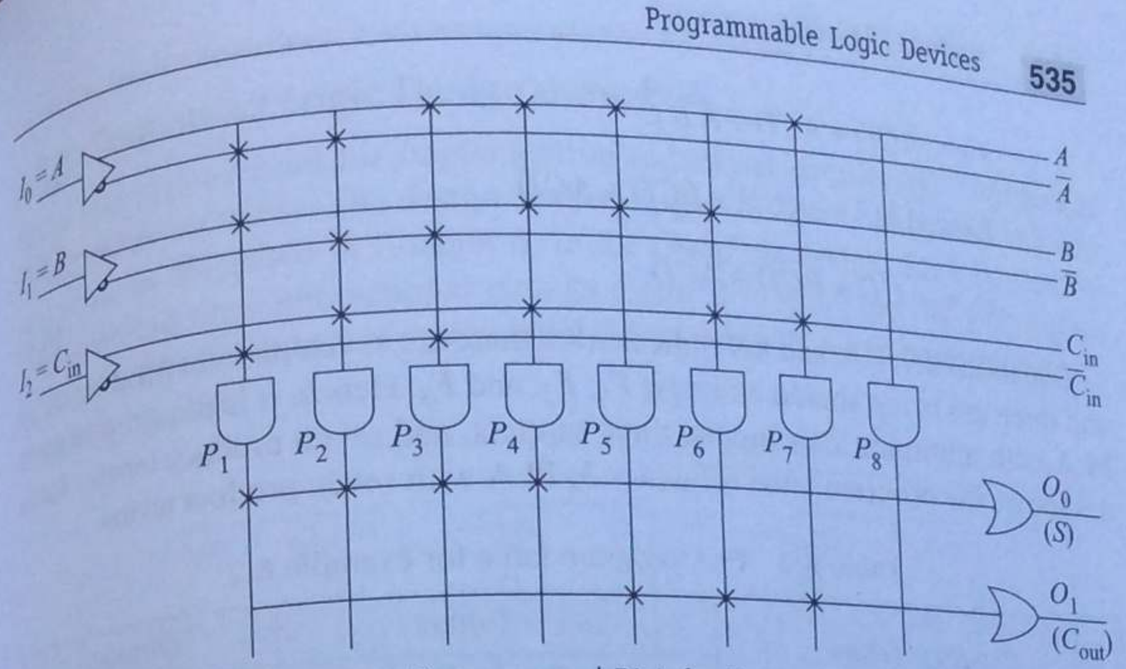


Fig. 8.11 Programmed PLA for Example 8.1

Example 8.2 Implement the following output functions using a suitable PLA.

$$F_1(A, B, C, D) = \Sigma m(3, 7, 8, 9, 11, 15)$$

$$F_2(A, B, C, D) = \Sigma m(3, 4, 5, 7, 10, 14, 15)$$

$$F_3(A, B, C, D) = \Sigma m(1, 5, 7, 11, 15)$$

Solution

The block diagram for the above multiple output system is shown in Fig. 8.12.

While implementing a combinational circuit using PLA, first there is a need to find the function in sum of products form. This decides the required size of PLA and further, it is used to get the PLA program table. Even though K-maps provide an optimum minimization of the function, while implementing the functions using PLA, we must try to find the common product terms among the K-maps of the outputs. This allows us to share the product terms and thus reduces the number of AND gates required. Hence, while looping the minterms for PLA implementation, the approach is slightly different than implementing using AND-OR gates.

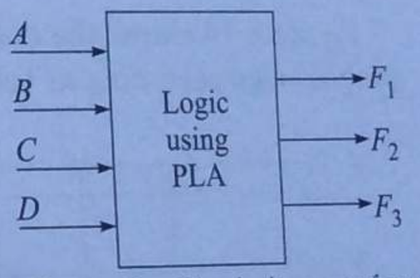


Fig. 8.12 Block diagram for multiple output system

Figure 8.13 shows the K-maps for F_1, F_2, F_3 and gives their SOP forms having a minimum number of distinct product terms.

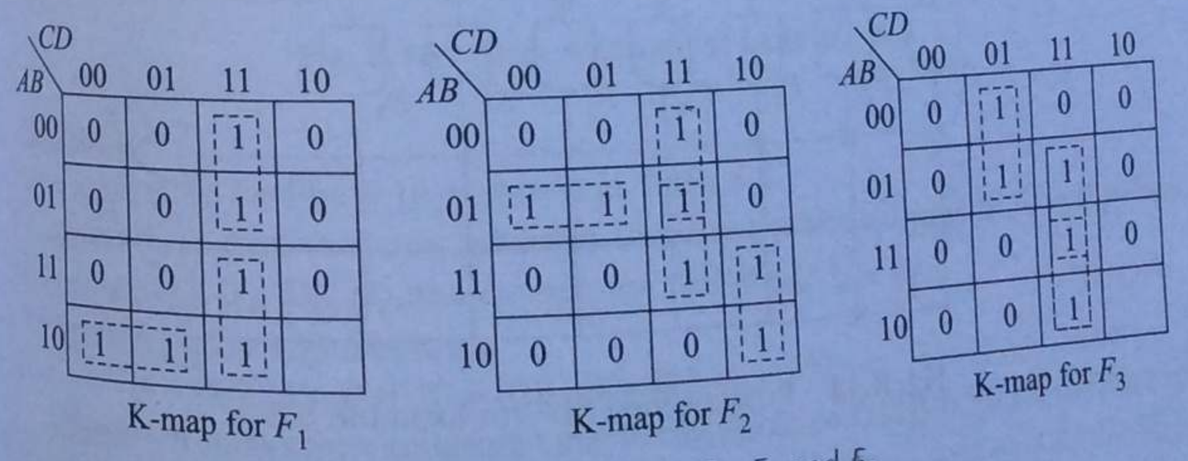


Fig. 8.13 K-maps for F_1, F_2 and F_3

$$F_1 = \overline{A}CD + AC\overline{D} + A\overline{B}\,\overline{C} \tag{8.7}$$

$$F_2 = \overline{A}B\overline{C} + \overline{A}CD + BC\overline{D} + AC\overline{D} \tag{8.8}$$

$$F_3 = \overline{A}\,\overline{C}D + BCD + AC\overline{D} \tag{8.9}$$

Equations 8.7, 8.8, and 8.9 indicate that there are seven distinct product terms and three are being shared amongst F_1, F_2, and F_3. Hence, it is required to use a PLA with minimum four inputs, three outputs, and seven product terms. Table 8.3 shows the program table for a (4×3) PLA with seven product terms.

Table 8.3 PLA program table for Example 8.2

Product terms		Inputs				Outputs		
		A	B	C	D	F_1	F_2	F_3
$\overline{A}CD$	1	0	–	1	1	1	1	–
$AC\overline{D}$	2	1	–	1	1	1	–	1
$A\overline{B}\,\overline{C}$	3	1	0	0	–	1	–	–
$\overline{A}B\overline{C}$	4	0	1	0	–	–	1	–
$AC\overline{D}$	5	1	–	1	0	–	1	–
BCD	6	–	1	1	1	–	1	1
$\overline{A}\,\overline{C}D$	7	0	–	0	1	–	–	1
						T	T	T

Figure 8.14 shows the diagram of a (3×2) PLA with seven product terms programmed according to Table 8.3.

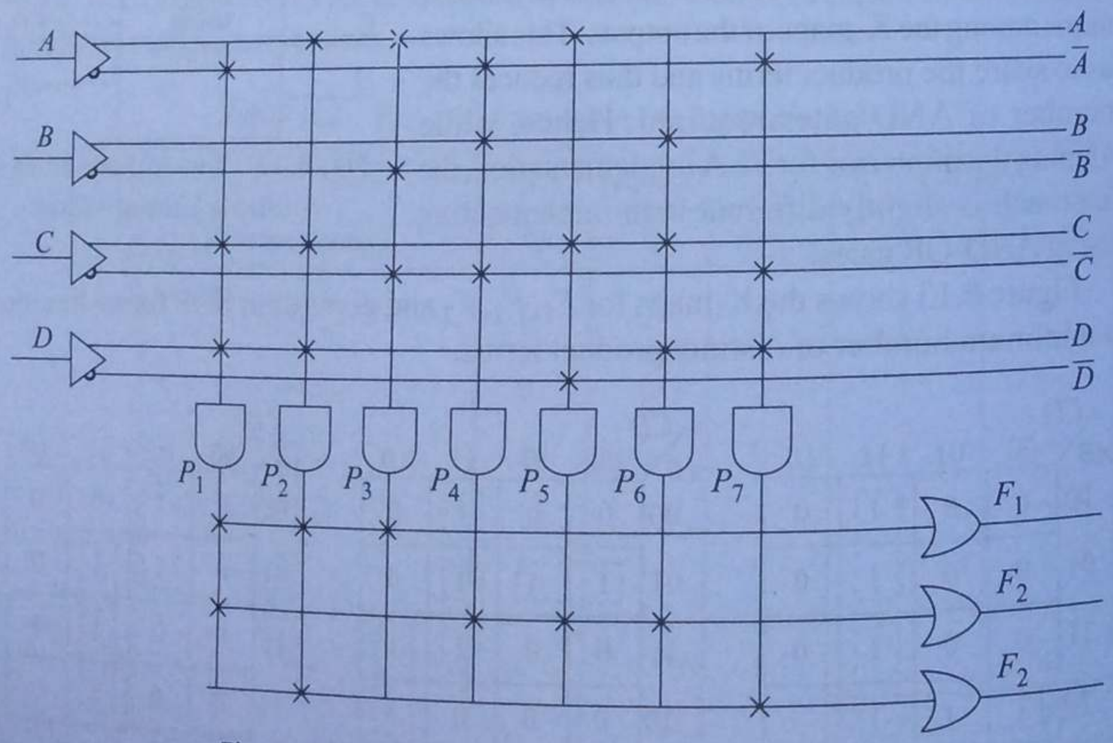


Fig. 8.14 Programmed PLA for Example 8.2

8.2.3 Sequential Logic Design using PLA

PLAs can also be used for implementing sequential circuits like counters or synchronous machines. The design methodology for sequential circuits remains the same, as discussed in Chapter 5. In the complete design, PLA is used for implementing the combinational circuits required to drive the flip-flop inputs. Hence, the block diagram of a sequential circuit using PLA can be represented as shown in Fig. 8.15.

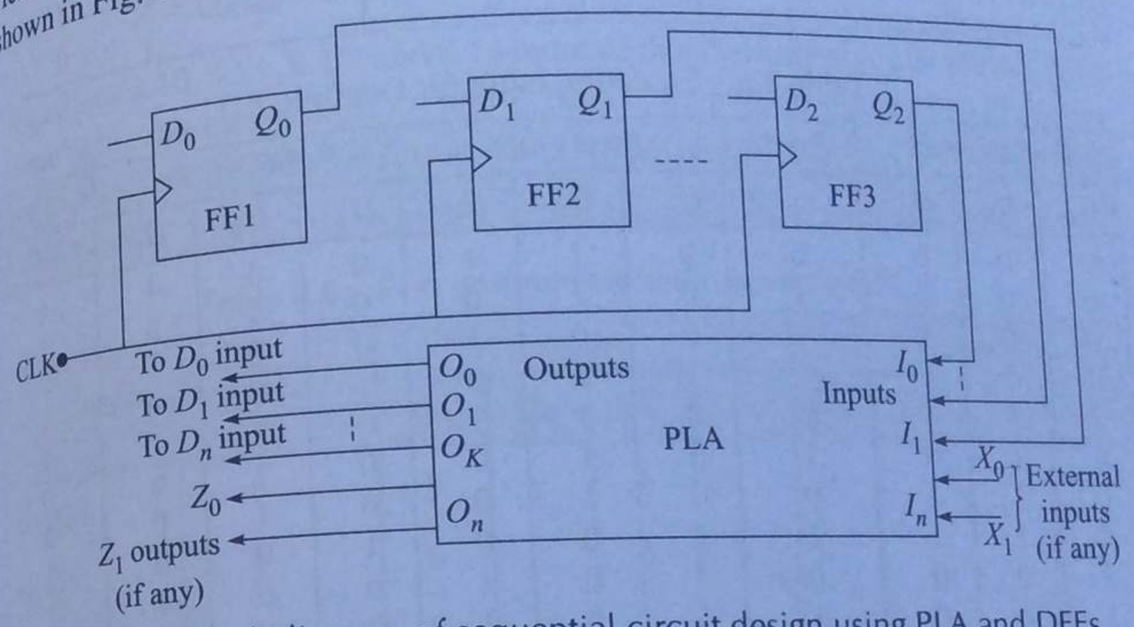


Fig. 8.15 Block diagram of sequential circuit design using PLA and DFFs

Example 8.3 Design a synchronous sequential circuit for the state diagram shown in Fig. 8.16 and implement it using D flip-flops and a suitable PLA.

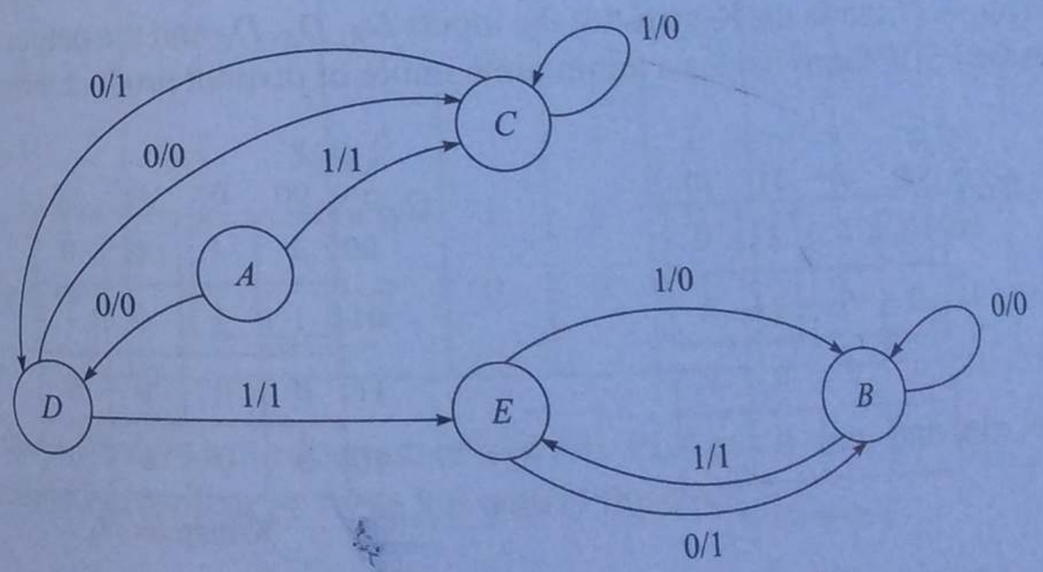


Fig. 8.16 State diagram for Example 8.3

Solution

The state table for Fig. 8.16 is shown in Table 8.4.

Since there are five distinct states, we need to use three flip-flops. The flip-flop inputs will be D_A, D_B, D_C and the outputs will be Q_A, Q_B, Q_C.

Let the state assignments be

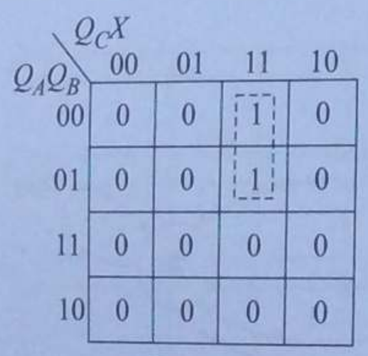
Table 8.4 State table for Example 8.3

Present state	Next state		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
A	D	C	0	1
B	B	E	0	1
C	D	C	1	0
D	C	E	0	1
E	B	B	1	0

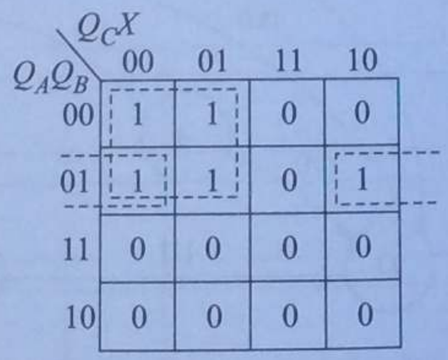
Table 8.5 Excitation table for Example 8.3

Present state			Input	Next state			FF inputs			o/p
Q_A	Q_B	Q_C	x	Q_A	Q_B	Q_C	D_A	D_B	D_C	Z
0	0	0	0	0	1	1	0	1	1	0
0	0	0	1	0	1	0	0	1	0	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	1	0	0	1	0	0	1
0	1	0	0	0	1	1	0	1	1	1
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	1	0	0
0	1	1	1	1	0	0	1	0	0	1
1	0	0	0	0	0	1	0	0	1	1
1	0	0	1	0	0	1	0	0	1	0
1	0	1	X	0	0	0	0	0	0	0
1	1	0	X	0	0	0	0	0	0	0
1	0	1	X	0	0	0	0	0	0	0

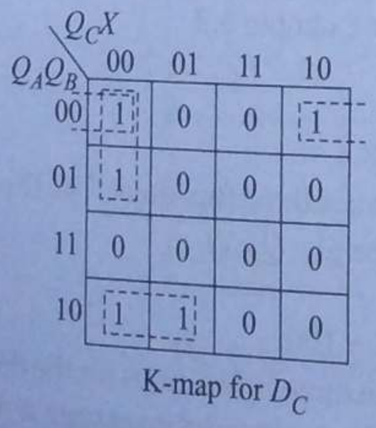
Figure 8.17 shows the K-maps for the inputs D_A, D_B, D_C and the output Z, and gives their SOP forms having a minimum number of distinct product terms.



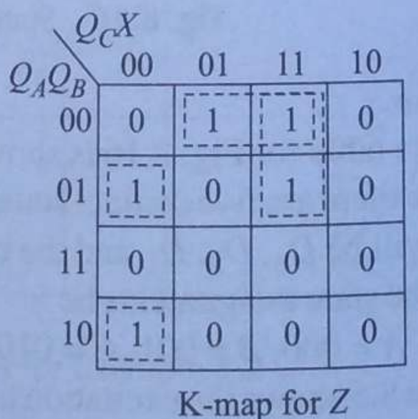
K-map for D_A



K-map for D_B



K-map for D_C



K-map for Z

Fig. 8.17

The design equations for the inputs D_A, D_B, D_C, and the output Z are:

$$D_A = \overline{Q}_A Q_C X \tag{8.10}$$

$$D_B = \overline{Q}_A \overline{Q}_C + \overline{Q}_A Q_B \overline{X} \tag{8.11}$$

$$D_C = \overline{Q}_A \overline{Q}_C \overline{X} + Q_A \overline{Q}_B \overline{Q}_C + \overline{Q}_A \overline{Q}_B \overline{X} \tag{8.12}$$

$$Z = \overline{Q}_A \overline{Q}_B X + \overline{Q}_A Q_C X + \overline{Q}_A Q_B \overline{Q}_C \overline{X} + Q_A \overline{Q}_B \overline{Q}_C \overline{X} \tag{8.13}$$

Equations 8.10, 8.11, 8.12, and 8.13 indicate that there are nine distinct products and one being shared amongst D_A and Z. Hence, we need to use a (4×4) PLA with nine product terms. Table 8.6 shows the program table for a (4×4) PLA with nine product terms.

Table 8.6 PLA program table for Example 8.3

Product terms		Inputs				Outputs			
		Q_A	Q_B	Q_C	x	D_A	D_B	D_C	Z
$\overline{Q}_A Q_C X$	1	0	–	1	1	1	–	–	1
$\overline{Q}_A \overline{Q}_C$	2	0	–	0	–	–	1	–	–
$\overline{Q}_A Q_B \overline{X}$	3	0	1	–	0	–	1	–	–
$\overline{Q}_A \overline{Q}_C \overline{X}$	4	0	–	0	0	–	–	1	–
$Q_A \overline{Q}_B \overline{Q}_C$	5	1	0	0	–	–	–	1	–
$\overline{Q}_A \overline{Q}_B \overline{X}$	6	0	0	–	0	–	–	1	–
$\overline{Q}_A \overline{Q}_B X$	7	0	0	–	1	–	–	–	1
$\overline{Q}_A Q_B \overline{Q}_C \overline{X}$	8	0	1	0	0	–	–	–	1
$Q_A \overline{Q}_B \overline{Q}_C \overline{X}$	9	1	0	0	0	–	–	–	1
						T	T	T	T

Figure 8.18 shows the diagram of a (4×4) PLA with nine product terms programmed according to Table 8.6 with D flip-flops.

8.3 PROGRAMMABLE ARRAY LOGIC

Programming array logic (PAL) devices are those PLDs which are very widel used and are readily available. PALs, like PLAs, consist of an array of AND gate and an array of OR gates. The major difference between PLA and PAL devices that, in case of PLA, both the arrays are programmable, whereas in PAL, OR arra is fixed and AND array is programmable. Because only the AND gates a programmable, the PAL is easier to program, but is not as flexible as the PLA

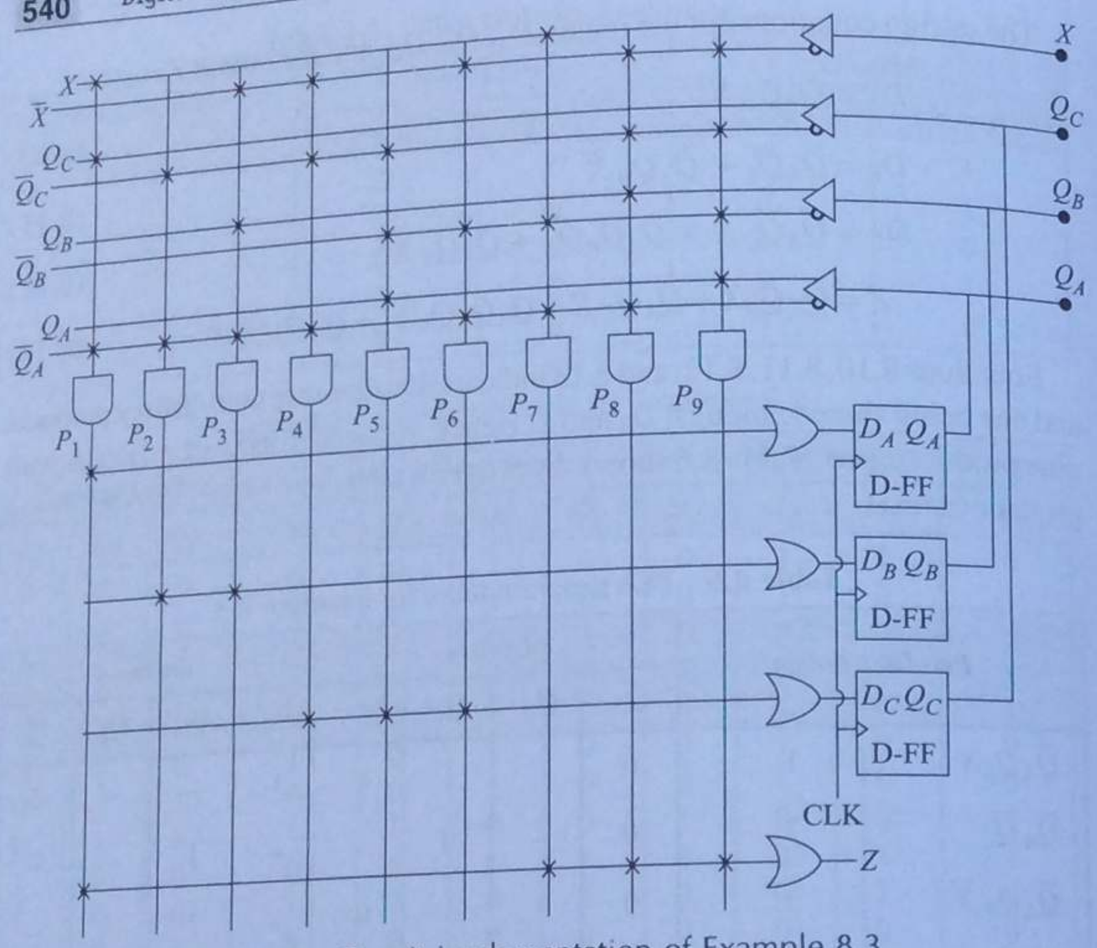


Fig. 8.18 Circuit implementation of Example 8.3

8.3.1 Block Diagram of PAL

PAL is a type of fixed architecture logic device with programmable AND gates and fixed OR gates. The block diagram of PAL is as shown in Fig. 8.19.

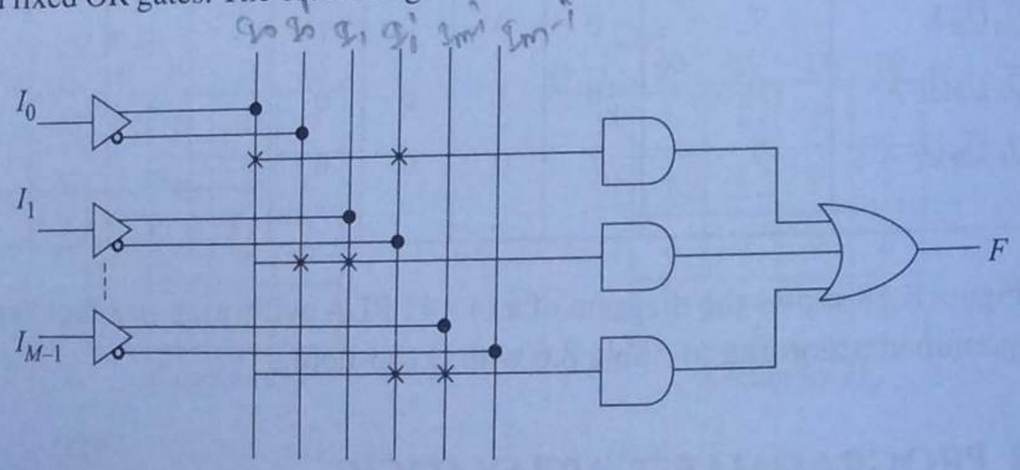


Fig. 8.19 Block diagram of PAL

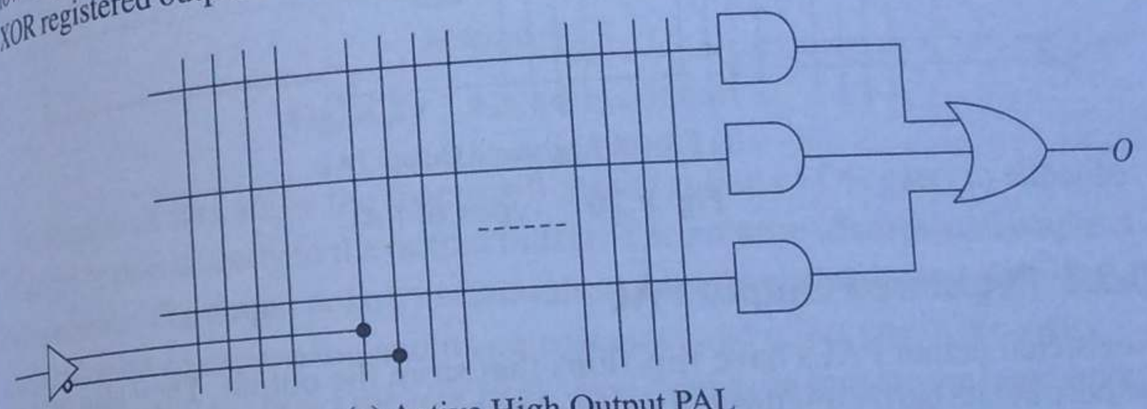
The block diagram of PAL, shown in Fig. 8.19, has input buffers, AND gates array, and OR gates array. It has M inputs (I_0 through I_{M-1}), and N outputs (O_0 through O_{N-1}).

An '×' indicates the programmed connection between an input signal and the input of AND gate. The programmed connections are made via EPROM cells or other programming technology.

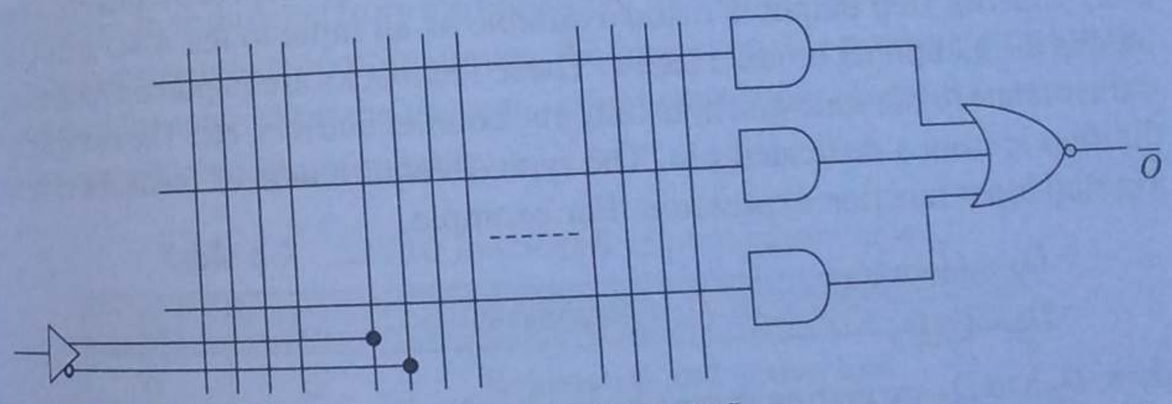
PAL devices are named as M type N, where M represents the number of inputs, N represents the number of outputs, and the middle letter dictates the type of output arrangement, which can be one of the following:

L : Active low outputs
H : Active High outputs
C : Both active high and active low outputs
R : Registered outputs
X : XOR registered outputs
V : Variable product terms per OR gate
P : Programmable output

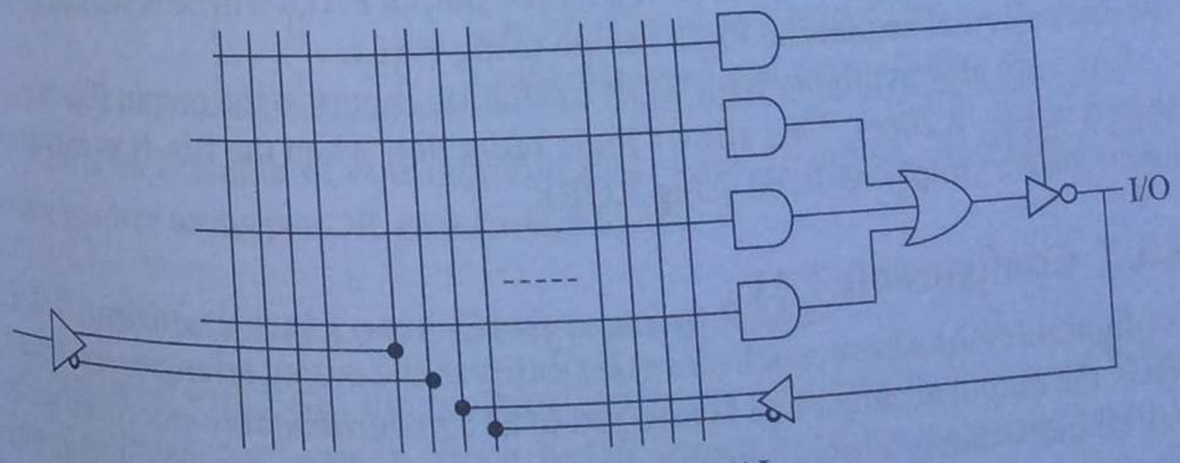
For example, 16R8 is a PAL with 16 inputs and 8 registered outputs. Figure 8.20 (a), (b), (c), (d) and (e) show the active high output PAL, the active low output PAL, the programmable I/O PAL, the registered output PAL, and the XOR registered output PAL, respectively.



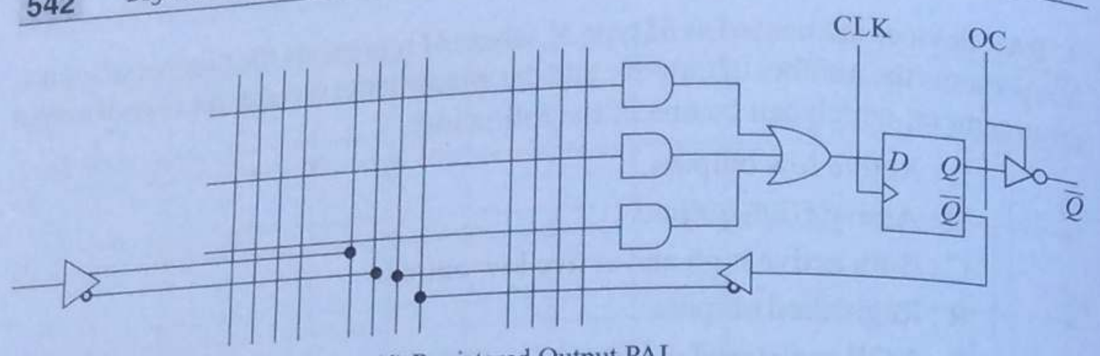
(a) Active High Output PAL



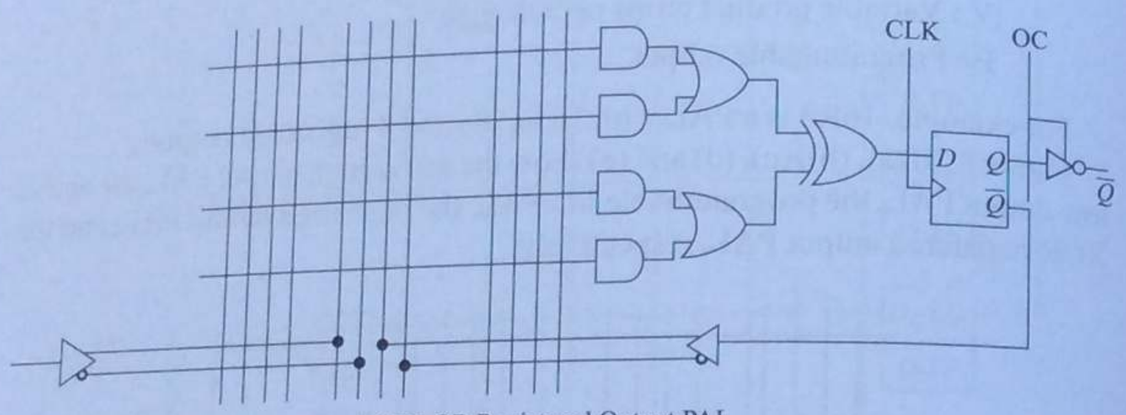
(b) Active Low Output PAL



(c) Programmable I/O PAL



(d) Registered Output PAL



(e) EX-OR Registered Output PAL

Fig. 8.20 Types of PAL

8.3.2 Registered Output PAL

Registered output PALs have flip-flops that store the output. They also have an output tri-state buffer. Figure 8.20(d) shows the arrangement of a registered output PAL. The flip-flop output is made available as an input to the AND array to be used in the additional product terms. These feedbacks are important for creating the next state logic, state machine outputs, counter shifters, etc. The clock for the flip-flop is from a dedicated pin. The typical requirement of feedback is in the flip-flop input function expression. For example,

$$D_A = \overline{Q}_A + Q_B$$
$$D_B = Q_A Q_B$$

where D_A and D_B are getting driven by the product terms in which the feedback is taken from Q_A and Q_B. The use of registered output PALs will be demonstrated in the discussion of sequential logic design using PALs.

PALs are also available with XOR excitation inputs to the output flip-flop, as shown in Fig. 8.20(e). This allows logic reduction when the flip-flop excitation functions are simply realized using XORs.

8.3.3 Configurable PAL

Configurable PALs have much more flexibility in the output arrangement. In such PALs, the output arrangement is referred to as a *programmable macrocell* which may be individually programmed by the state of configuration bits. Figure 8.21 shows a typical macrocell of a widely used PAL: **22V10.**

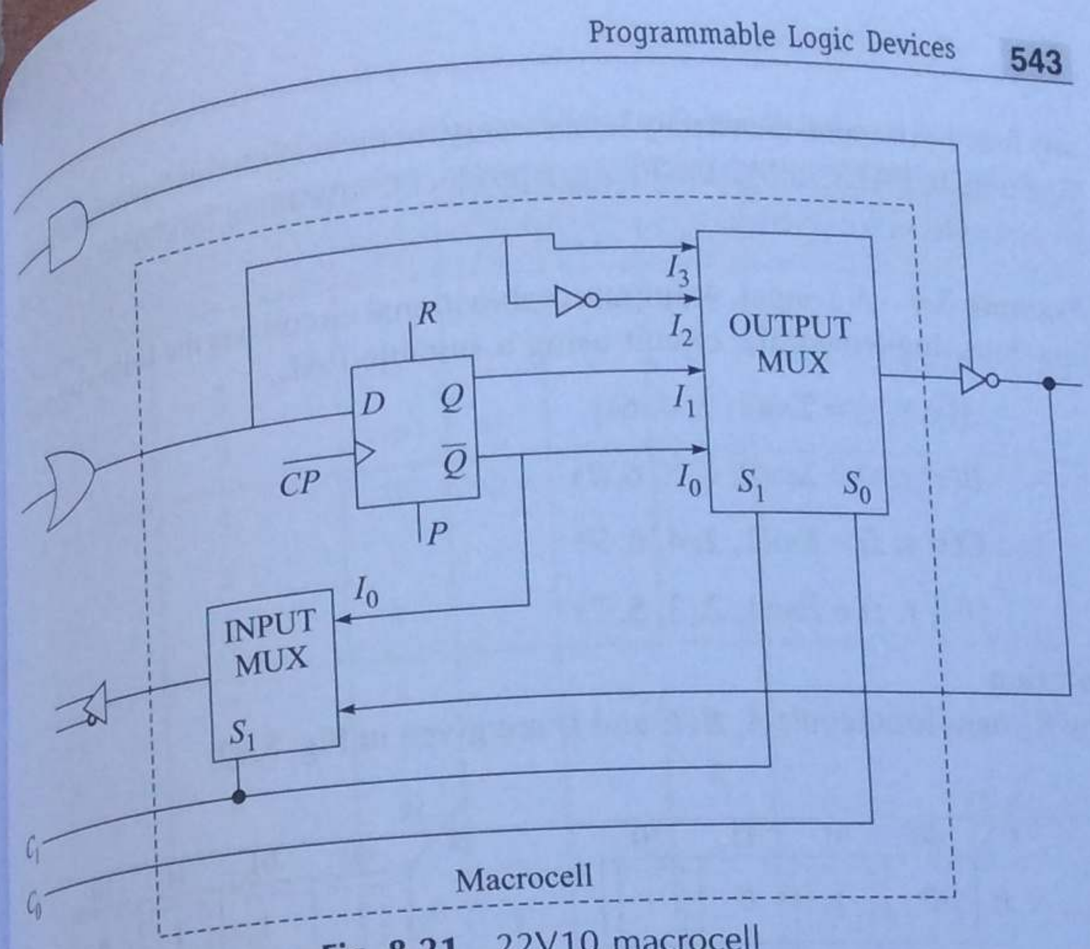


Fig. 8.21 22V10 macrocell

Configuration bits allow the macrocell input (output of OR gate) to either be registered or pass directly to the output buffer. The polarity of macrocell output is also selectable. The output polarity selection enables the software to perform logic optimization on an expression and its complement and select one of the logic.

The macrocell also allows the feedback to be used as an input to the logic array. The feedback can be directed from the register or I/O buffer, depending on whether the signal is registered or combinational. An output enable product term can be programmed for controlling the tri-state output enable pin. Table 8.7 shows the 22V10 macrocell configuration table.

Table 8.7 22V10 macrocell configuration table

C_1	C_0	*Description*
0	0	Registered and active low
0	1	Registered and active high
1	0	Combinational and active low
1	1	Combinational and active high

22V10 is an example of configurable PAL with 22 inputs, 10 outputs, and variable product terms per OR gate from 8 to 16.

8.3.4 Combinational Logic Design using PALs

While designing with PAL, the function in its sum of products form need to be simplified to fit into each section. Since the OR array is fixed, the number of product terms per OR gate cannot be changed. Hence, if the number of product terms in

any function is more, then it may be necessary for the sections to implement Boolean function. In PALs, unlike the PLAs, a product term cannot be shared among two or more OR gates.

Example 8.4 A 3-input, 4-output combinational circuit has the following output functions. Implement the circuit using a suitable PAL.

$A(x, y, z) = \Sigma m(1, 2, 4, 6)$

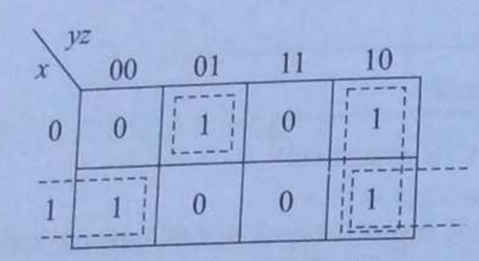
$B(x, y, z) = \Sigma m(0, 1, 3, 6, 7)$

$C(x, y, z) = \Sigma m(1, 2, 4, 6, 7)$

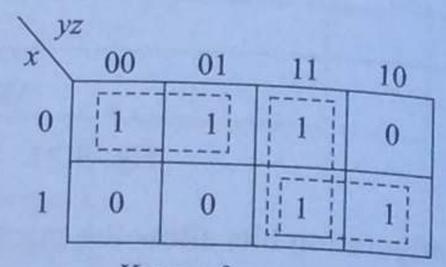
$D(x, y, z) = \Sigma m(1, 2, 3, 5, 7)$

Solution

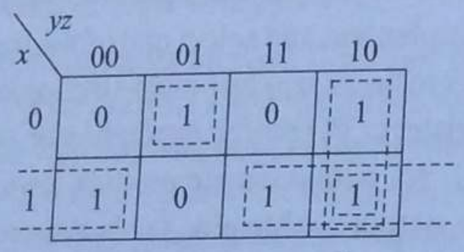
The K-maps for outputs A, B, C and D are given in Fig. 8.22.



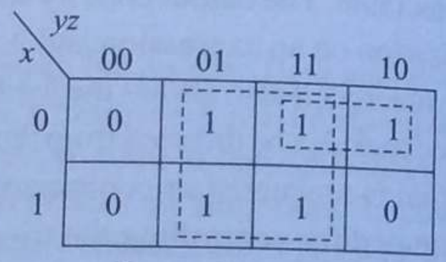
K-map for outputs A



K-map for outputs B



K-map for outputs C



K-map for outputs D

Fig. 8.22 K-maps for A, B, C, and D

$$A = x\bar{z} + y\bar{z} + \bar{x}\,\bar{y}\,z \tag{8.14}$$

$$B = \bar{x}\,\bar{y} + yz + xy \tag{8.15}$$

$$C = \bar{x}\,\bar{y}z + x\,\bar{z} + xy + y\,\bar{z} \tag{8.16}$$

$$D = z + \bar{x}y \tag{8.17}$$

From the expression of output C (8.16) and output A (8.14),

$$C = xy + A$$

From equations 8.14, 8.15, 8.16, and 8.17, it can be seen that there are 3 inputs, 4 outputs, and a maximum of 3 product terms per OR gate. Hence, a 4×4 PAL is required with programmable input/output so as to allow us the use of one output function (A) in the other output function (C) with feedback.

The PAL program table shown in Table 8.8 contains three columns specifying product terms (no), inputs and outputs. In this example, the input column also contains A, since the feedback is used for one AND gate input.

Table 8.8 PAL program table for Example 8.4

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

Figure 8.23 shows the 4×4 PAL with 3 product terms per OR gate programmed as per Table 8.8.

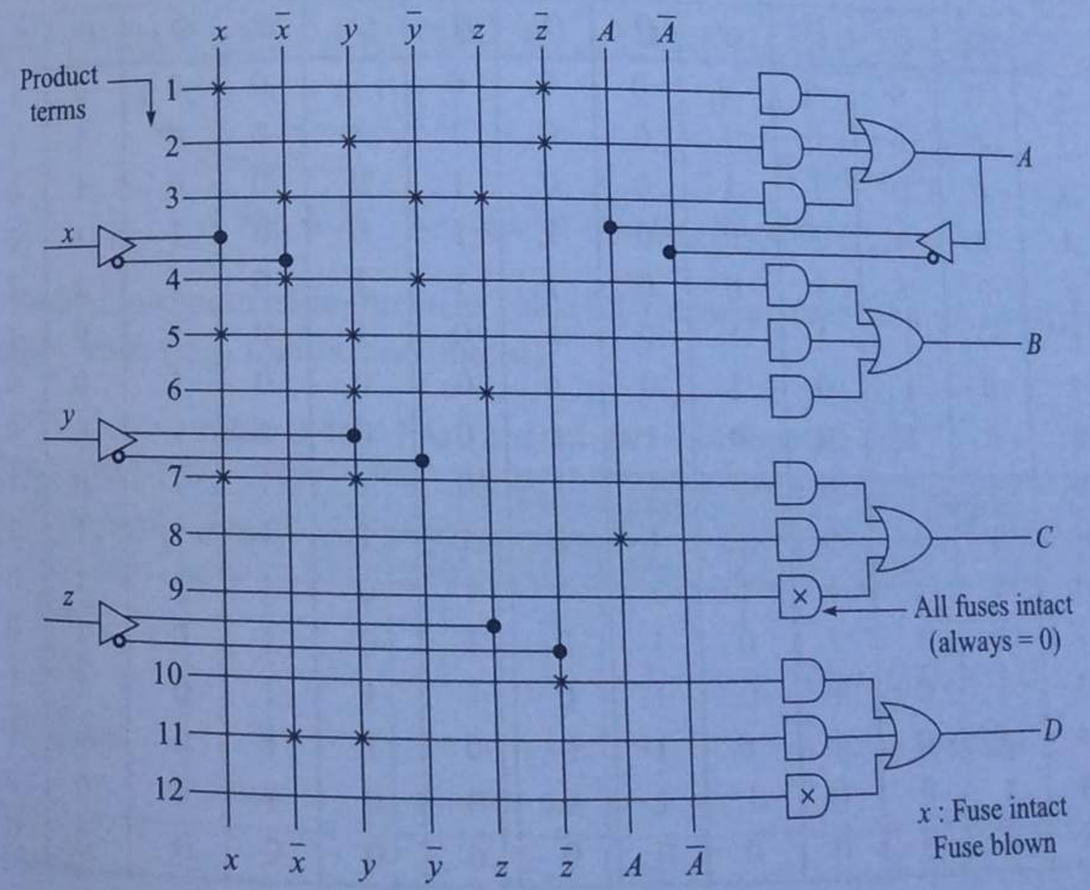


Fig. 8.23 Programmed PAL for Example 8.4

8.3.5 Sequential Logic Design using PAL

The PALs having registered outputs and feedback from output to input can be used in realizing sequential logic circuits. These PALs can be used for implementing

Mealy and Moore type of sequential circuits. The PALs which are designed for sequential circuits have bi-directional capacity. As shown in Fig. 8.20 (d), registered outputs have tri-state control. If they are disabled, these output pins can be used for giving additional external inputs and if the tri-state control is enabled, the pins give feedback to the AND gates. Since these PALs have flip-flops as well as AND-OR arrays, a complete sequential circuit can be fit into it without any external hardware.

Example 8.5 Design a 4-bit Gray code counter using a suitable PAL device. Show the coded logic diagram of the device.

Solution

The state diagram for a 4-bit gray counter is shown in Fig. 8.24.

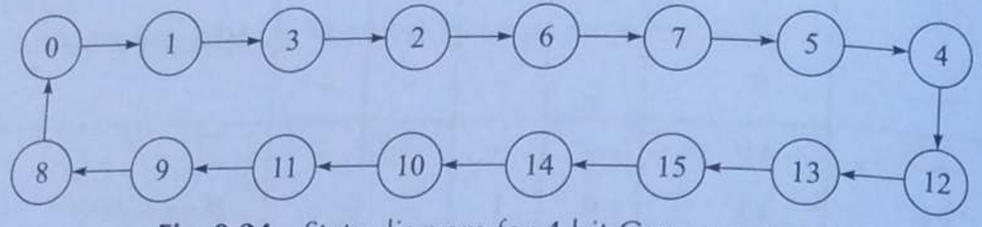


Fig. 8.24 State diagram for 4-bit Gray counter

Table 8.9 Excitation table for Example 8.5

CLK	Present State				Next State				Flip-flop i/p			
	Q_0	Q_1	Q_2	Q_3	Q_0	Q_1	Q_2	Q_3	D_0	D_1	D_2	D_3
1	0	0	0	0	0	0	0	1	0	0	0	1
2	0	0	0	1	0	0	1	1	0	0	1	1
3	0	0	1	1	0	0	1	0	0	0	1	0
4	0	0	1	0	0	1	1	0	0	1	1	0
5	0	1	1	0	0	1	1	1	0	1	1	1
6	0	1	1	1	0	1	0	1	0	1	0	1
7	0	1	0	1	0	1	0	0	0	1	0	0
8	0	1	0	0	1	1	0	0	1	1	0	0
9	1	1	0	0	1	1	0	1	1	1	0	1
10	1	1	0	1	1	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1	0	1	1	1	0
12	1	1	1	0	1	0	1	0	1	0	1	0
13	1	0	1	0	1	0	1	1	1	0	1	1
14	1	0	1	1	1	0	0	1	1	0	0	1
15	1	0	0	1	1	0	0	0	1	0	0	0
16	1	0	0	0	0	0	0	0	0	0	0	0

The K-maps for D_0, D_1, D_2, and D_3 are shown in Fig. 8.25.

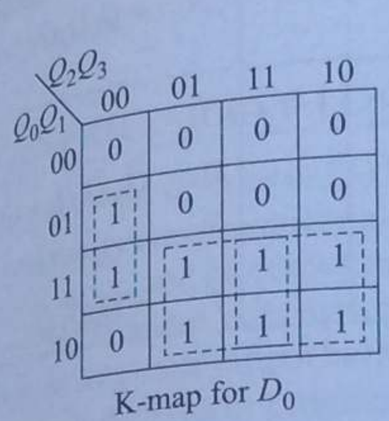
$$D_0 = Q_0Q_3 + Q_0Q_2 + Q_1\overline{Q}_2\overline{Q}_3 \tag{8.18}$$

$$D_1 = Q_1\overline{Q}_2 + Q_1Q_3 + \overline{Q}_0Q_2\overline{Q}_3 \tag{8.19}$$

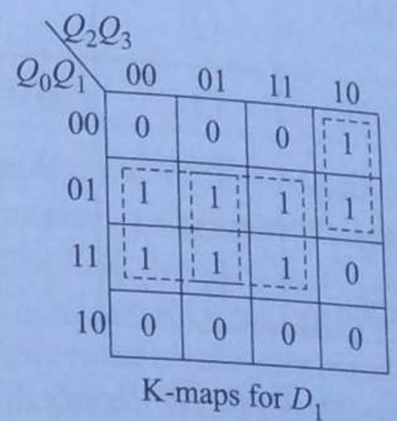
$$D_2 = Q_2\overline{Q}_3 + Q_3\overline{Q}_1\overline{Q}_0 + Q_3Q_1Q_0 \tag{8.20}$$

$$D_3 = \overline{Q}_0\overline{Q}_1\overline{Q}_2 + \overline{Q}_0Q_1Q_2 + Q_0Q_1\overline{Q}_2 + Q_0\overline{Q}_1Q_2 \tag{8.21}$$

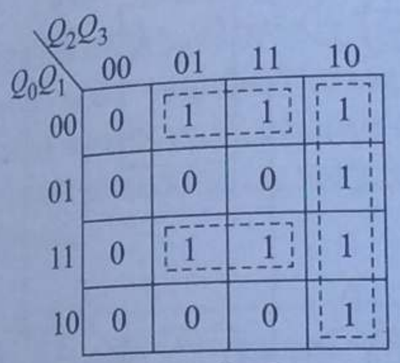
From equations 8.18, 8.19, 8.20, and 8.21, it can be seen that there are 4 inputs, 4 outputs, and a maximum of 13 product terms. Hence, we need a 4×4 output PAL.



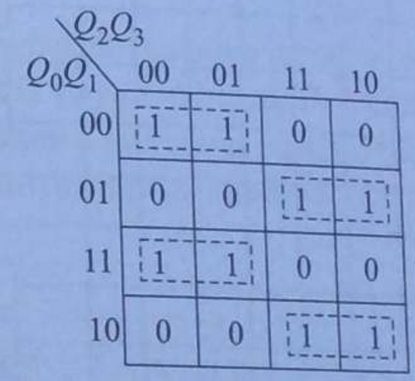
K-map for D_0



K-maps for D_1



K-map for D_2



K-map for D_3

Fig. 8.25 K-maps for D_0, D_1, D_2, and D_3

The PAL program table shown in Table 8.10 contains three columns, specifying product terms (no), inputs, and outputs

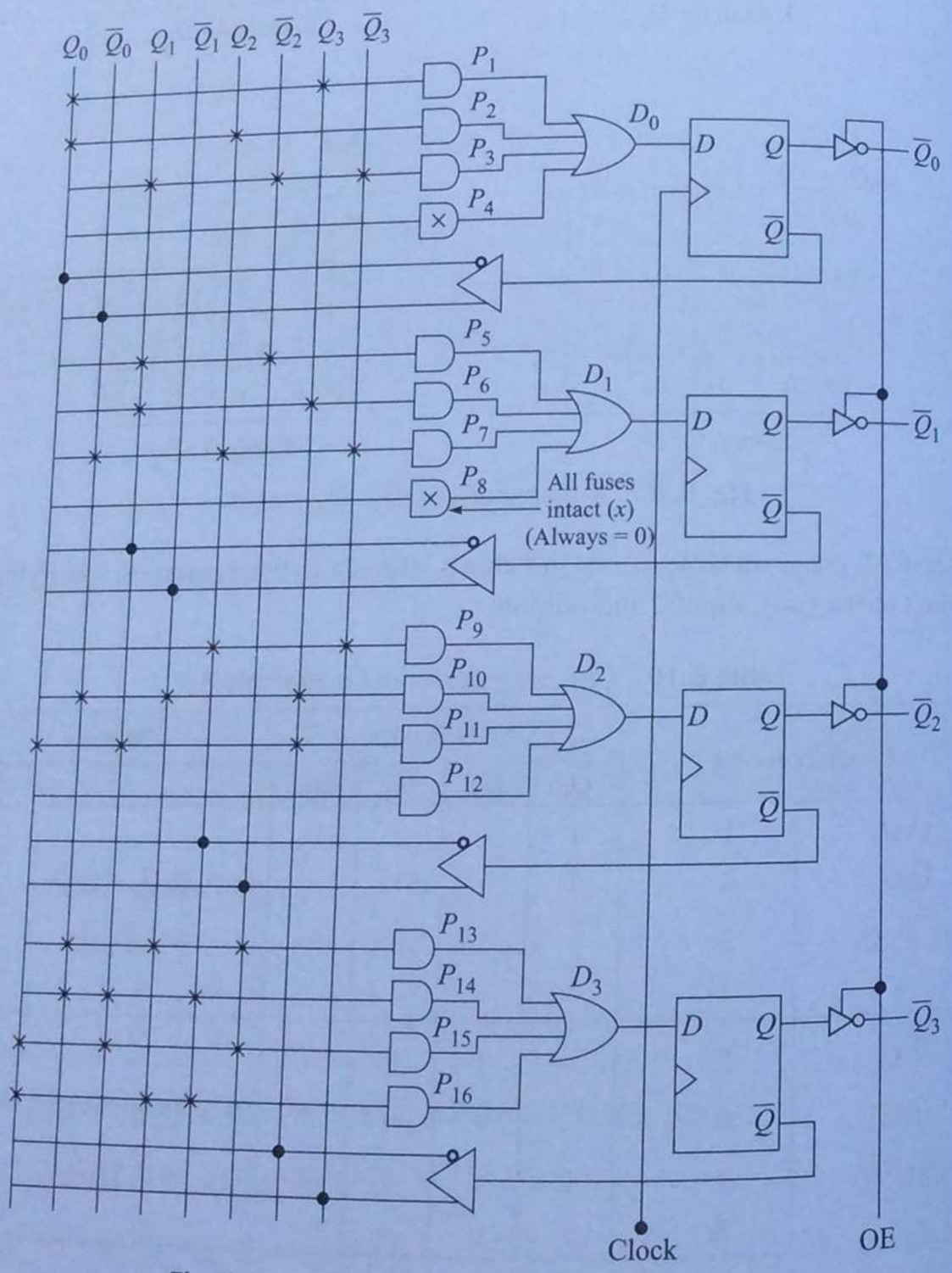
Table 8.10 PAL program table for Example 8.5

Product terms		AND inputs				Outputs
		Q_0	Q_1	Q_2	Q_3	
Q_0Q_3	1	1	–	–	1	
Q_0Q_2	2	1	–	1	–	$D_0 = Q_0Q_3 + Q_0Q_2$
$Q_1\overline{Q}_2\overline{Q}_3$	3	–	1	0	0	$+ Q_1\overline{Q}_2\overline{Q}_3$
–	4	–	–	–	–	
Q_1Q_2	5	–	1	0	–	
Q_1Q_3	6	–	1	–	1	$D_1 = Q_1\overline{Q}_2 + Q_1Q_3$
$\overline{Q}_0Q_2\overline{Q}_3$	7	0	–	1	0	$+ \overline{Q}_0Q_2\overline{Q}_3$
–	8	–	–	–	–	

Contd.

Contd.

Product terms		AND inputs Q_0	Q_1	Q_2	Q_3	Outputs
$Q_2\overline{Q}_3$	9	–	–	1	0	
$\overline{Q}_0\overline{Q}_1Q_3$	10	0	0	–	1	$D_2 = Q_2\overline{Q}_3 + \overline{Q}_0\overline{Q}_1Q_3$
$Q_0Q_1Q_3$	11	1	1	–	1	$+ Q_0Q_1Q_3$
–	12	–	–	–	–	
$\overline{Q}_0\overline{Q}_1\overline{Q}_2$	13	0	0	0	–	
$\overline{Q}_0Q_1Q_2$	14	0	1	1	–	$D_3 = \overline{Q}_0\overline{Q}_1\overline{Q}_2 + \overline{Q}_0Q_1Q_2$
$Q_0Q_1\overline{Q}_2$	15	1	1	0	–	$+ Q_0Q_1\overline{Q}_2$
$Q_0\overline{Q}_1Q_2$	16	1	0	1	–	$+ Q_0\overline{Q}_1Q_2$



From equations 8.18 to 8.21, it can be seen that there are 4 outputs to be generated (D_0 to D_3) from 4 inputs (Q_0 to Q_3) which are D flip-flop outputs. There are a maximum of 4 product terms per OR gate. Hence, we can select a 4×4 PAL with active high outputs and a set of four D flip-flops or we can select a 4×4 PAL with registered outputs with no external D flip-flops.

Table 8.10 shows the PAL program table and Fig. 8.26 shows the PAL diagram programmed according to the table.

8.4 GENERIC ARRAY LOGIC DEVICES

Generic array logic devices or GAL devices, introduced by Lattice Semiconductor Company, are a type of sequential PLDs. It contains a programmable AND array, a fixed OR array, and an output state similar to a PAL device. The output stage is referred to as the output logic macrocell (OLMC) by Lattice. It contains multiplexers, flip-flops, and tri-state output buffers. GAL devices are electrically erasable PLDs (EEPLDs), i.e. they can be programmed when they are connected on a circuit board.

8.4.1 Architecture of GAL16V8

A 16V8 device can have up to 20 inputs (with some pins as either inputs or registered feedbacks) and 8 outputs. The logic diagram is shown in Fig. 8.27. It contains an AND-array and the product terms generated from it is given as inputs to the OLMC block. These product terms are generated from the inputs or register outputs like a PAL device.

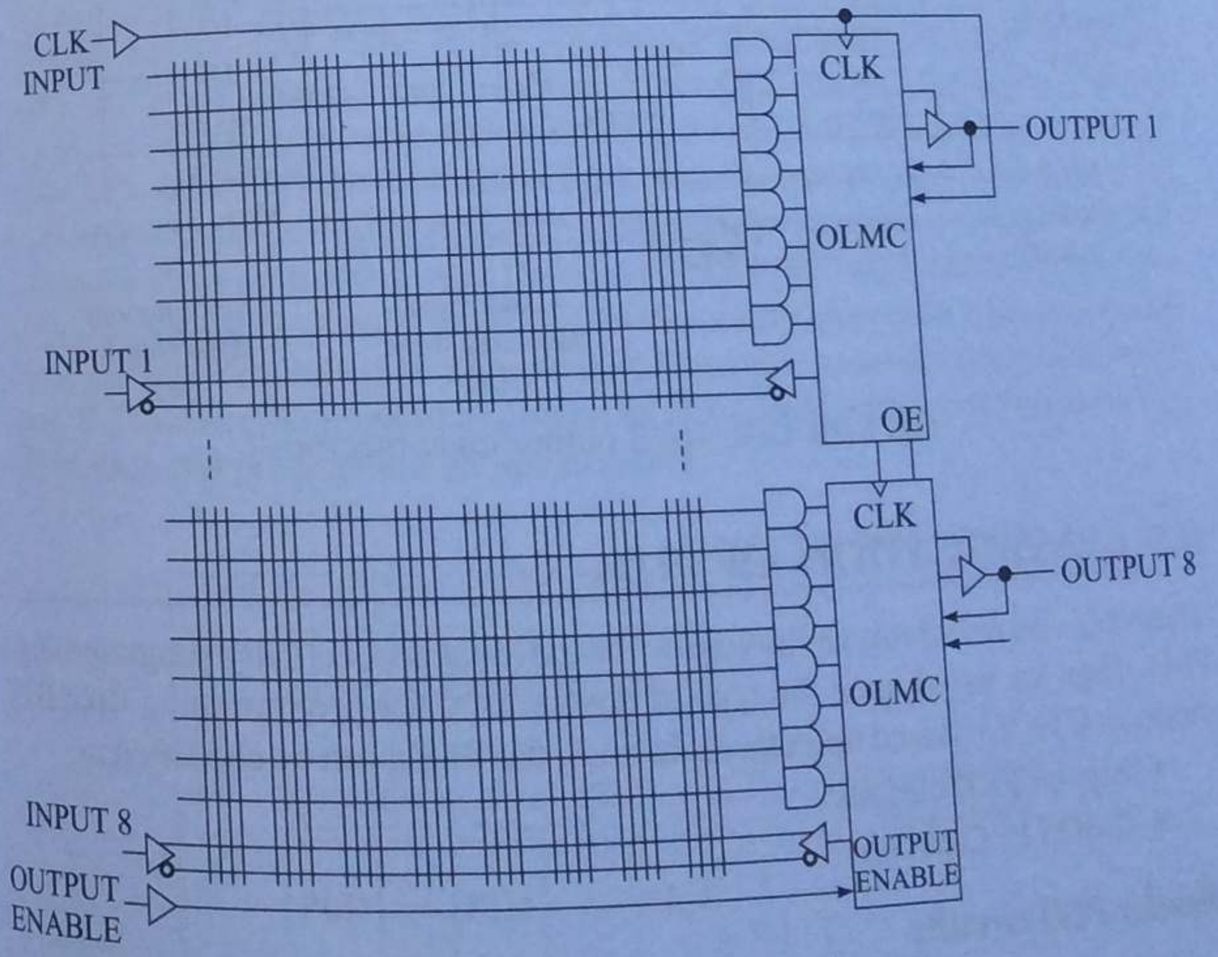


Fig. 8.27 GAL16V8 internal diagram

The OLMC, as shown in Fig. 8.28, has four multiplexers, one tri-state flip-flop, and a fixed OR array. In contrast to PAL, the output polarity of GAL is kept programmable by having a XOR gate in OLMC. The two control inputs AC_0 and AC_1 control the product term multiplexer (M_1) and the tri-state multiplexer (M_3). With the help of these control lines and multiplexers, the OLMC can be configured as a combinational function or registered function.

The feedback multiplexer is provided to have feedback from the registered output $\overline{Q}$, and the tri-state output from the output multiplexer or from an adjacent OLMC.

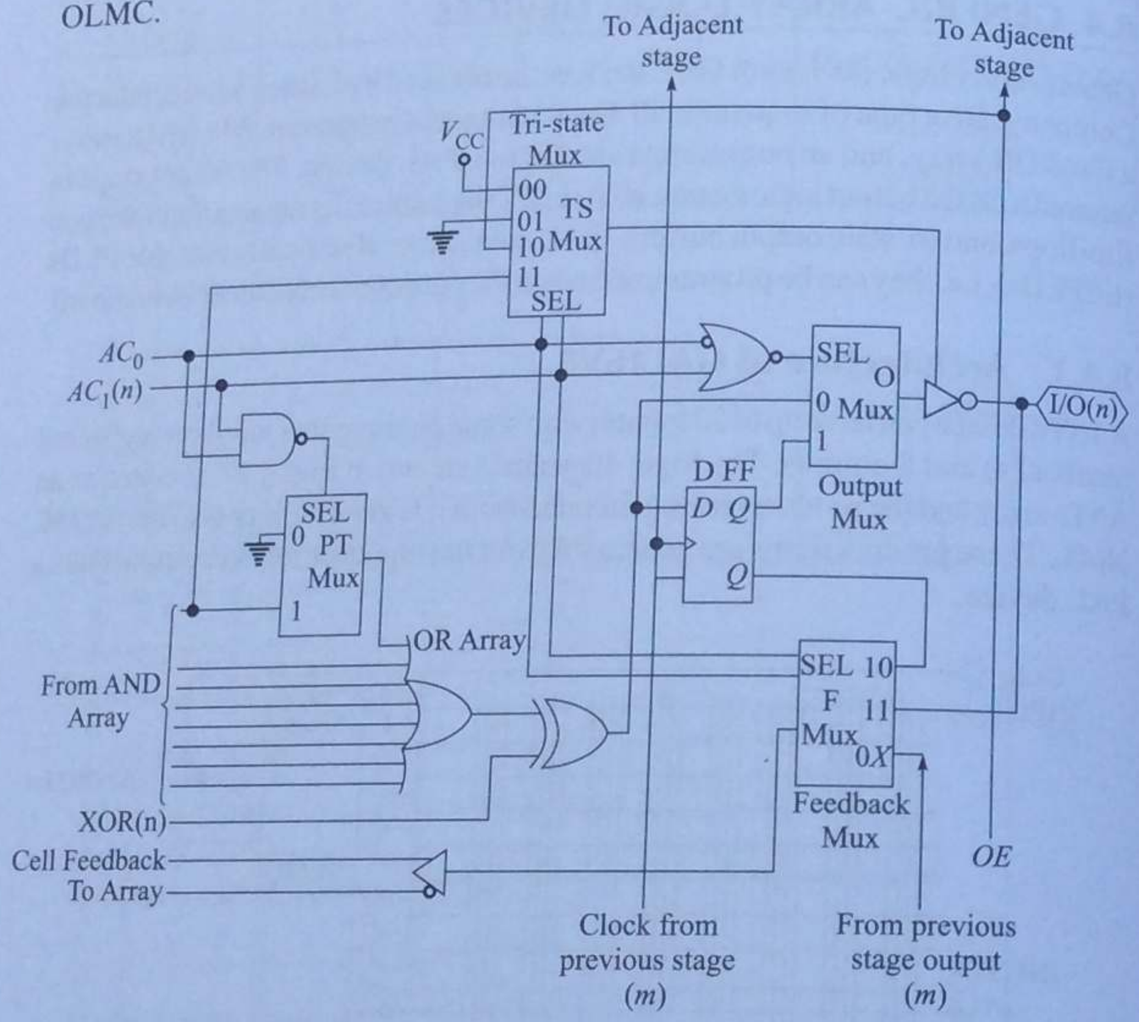


Fig. 8.28 GAL 16V8 output logic macrocell

8.5 CLASSIFICATION OF PLDs

There are various circuit technologies for building and actually programming a PLD. They are based upon the type of device used for programming (like BJT, diode, or CMOS). Based upon these devices, the PLDs can be classified as

- Bipolar PLD circuits
- CMOS PLD circuits

Bipolar PLD circuits

As discussed earlier, the AND-OR arrays have bipolar elements like diodes or transistors present at the row-column junction in a PLA. Earlier, PLA and PAL devices used these bipolar circuits. Such a bipolar PLD is manufactured with all

the diodes present but with a tiny fusible link in series with each. By applying special patterns to the device, one can select an individual link and by applying high voltage, these links are fused.

Earlier, these bipolar PLDs had reliability problems. These problems are eliminated by having more reliable fusible links in recent bipolar PLDs.

CMOS PLD circuits

In case of CMOS PLD circuits, instead of diode, an *n*-channel transistor with programmable connection is placed at each intersection between an input and word line (row-column of array). In CMOS PLDs, the programmable links are generally not fused. In non-field programmable devices, the links connection is removed or established as a part of the metal mask pattern generated by the manufacturer. In case of field-programmable devices, either floating gate programming technology is used or a read/write memory cell to control the state of each connection.

The different CMOS PLD circuits are:

- Erasable Programmable Logic Devices (EPLD)
- Electrically Erasable Programmable Logic Devices (EEPLD)
- Complex Programmable Logic Devices (CPLD)
- Field Programmable Gate Array Devices (FPGA)

8.6 COMPLEX PROGRAMMABLE LOGIC DEVICES

Programmable logic devices are basically LSI devices which are suitable in cases where the logic design has a larger number of inputs and outputs. When the number of inputs and outputs is still higher than the capacity of existing PLDs, it is necessary to increase the density. If the PLDs are simply made larger in terms of inputs, outputs, and product terms, they become slower due to capacitive effect, leakage currents, etc. At the same time, this solution does not make any cost-effective use of chip-area. Hence instead of increasing the inputs, outputs, and AND terms, it is better to have multiple PLDs structure having programmable connectivity.

Complex Programmable Logic Devices (CPLDs) are originated from this idea. CPLD is a collection of individual PLDs on a single chip, with a programmable interconnection structure that allows the PLDs to get connected as the user wants. Figure 8.29 shows the general architecture of a CPLD with input/output blocks, PLD blocks, and programmable interconnect.

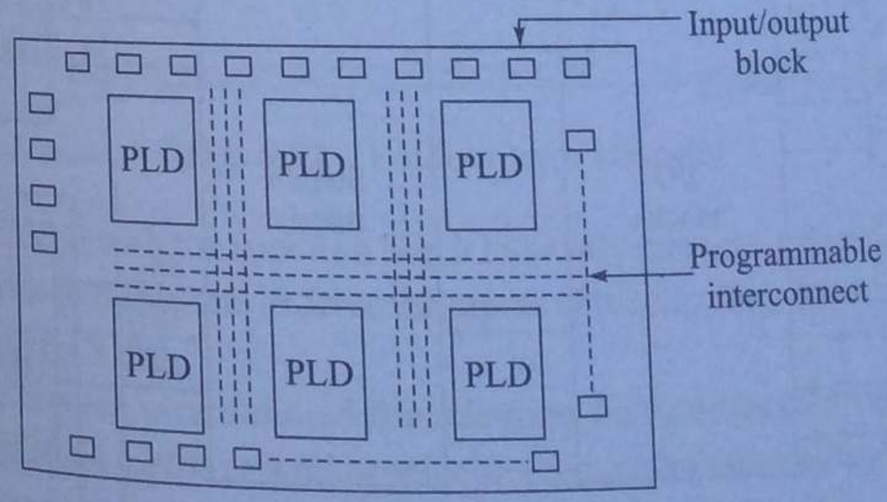


Fig. 8.29 CPLD architecture

CPLDs from different manufacturers differ in terms of individual PLDs, input/output blocks, and programmable interconnect. In this chapter, Xilinx 9500 series CPLD architecture is discussed.

8.6.1 Xilinx XC 9500 CPLD Family

The XC 9500 CPLD family is by Xilinx, with similar architecture for a series of devices which differ in terms of the number of external input-output (I/O) pins and the number of internal PLDs (referred as function block (FB) by Xilinx).

As shown in Table 8.11, the devices in the family are named according to the number of macrocells they contain. The number of I/O pins for each device varies according to their package type. The details about each part number and package can be found from Xilinx data book.

Table 8.11 FBs and Macrocells in Xilinx XC 9500

	Part Number				
	XC 9536	*XC 9572*	*XC 95108*	*XC 95144*	*XC 95288*
FB	2	4	6	8	16
Macrocells	36	72	108	144	288

Internal architecture

Figure 8.30 is a block diagram of the internal architecture of XC 9500 family CPLD. Each external I/O pin can be used as an input, output, or a bidirectional pin according to the device programming. Some of the pins (shown at the bottom) can also be used for special purposes like global clocks (GCK), global set/reset (GSR) and global three state controls (GTS).

The function blocks shown in the block diagram receives 36 signals from the switch matrix. The number of function blocks varies according to the specific device (minimum 2 is XC 9536 to maximum 16 in XC 95288).

The switch matrix receives 18 inputs from the macrocells within the FBs and the remaining inputs from external I/O pins. The 18 outputs from each FBs also get connected to external I/O pins through the switch matrix.

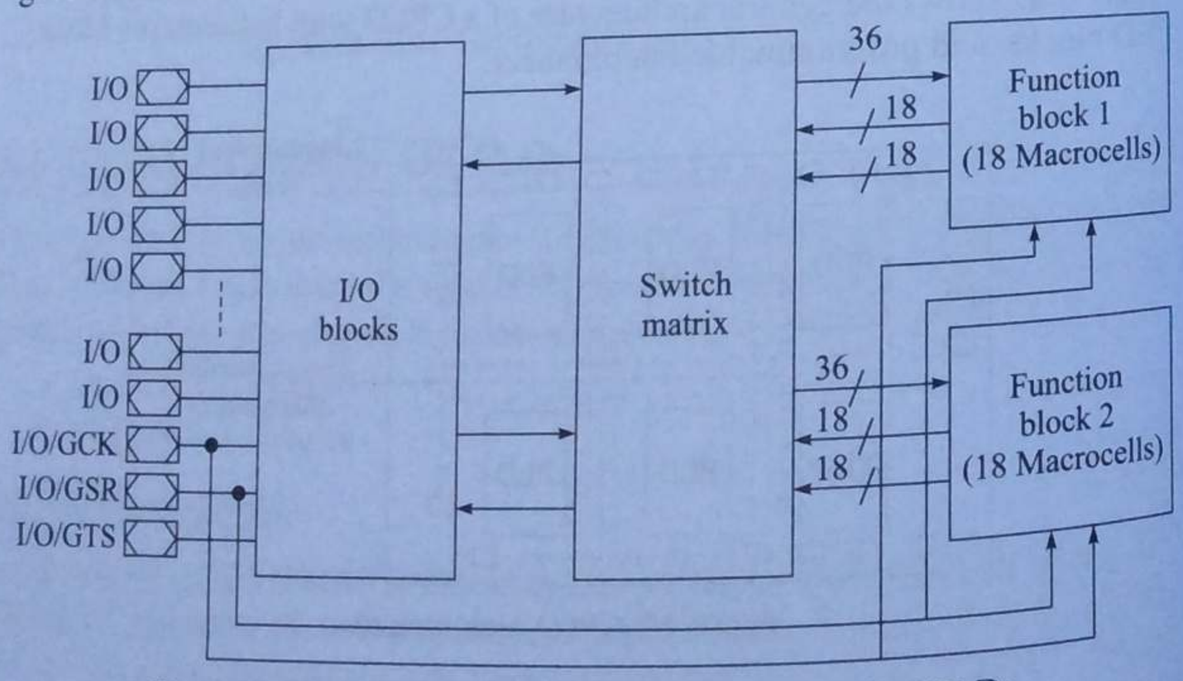


Fig. 8.30 Internal architecture of XC 9500 family CPLDs

Function block architecture

The basic structure of XC 9500 FB is shown in Fig. 8.31. This structure is similar to a PLD containing an AND matrix (programmable AND array), OR matrix (product term allocators), and programmable O/P arrangement (macrocell).

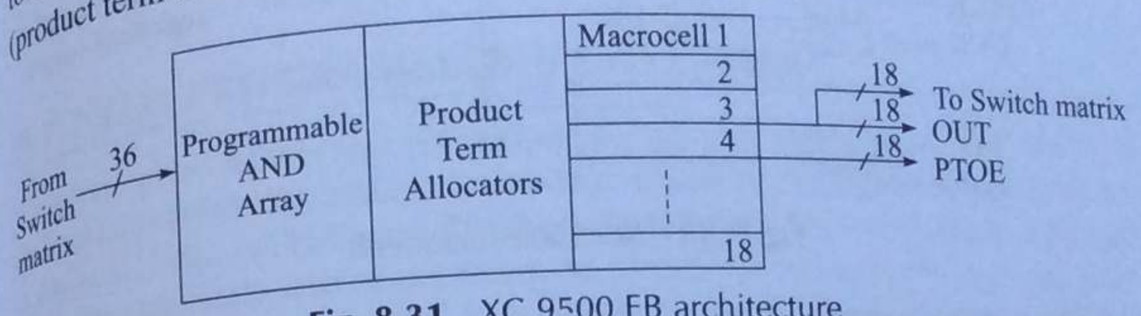


Fig. 8.31 XC 9500 FB architecture

XC 9500 function block has 5 product terms per macrocell. Hence, there are 90 product terms (18×5) in the AND array.

Even though fewer product terms are associated with each macrocell, with the help of a product term allocator, it is possible to use the unused product terms from the nearby macrocells with the help of a *product term allocator*. It consists of some programmable switches and gates to pass on or accept the product terms to or from the nearby macrocells as well as to create the sum terms by ORing the desired product terms. The switches in the product terms allocator steer the input product terms to one of their functions.

The product term allocator is shown in Fig. 8.32.

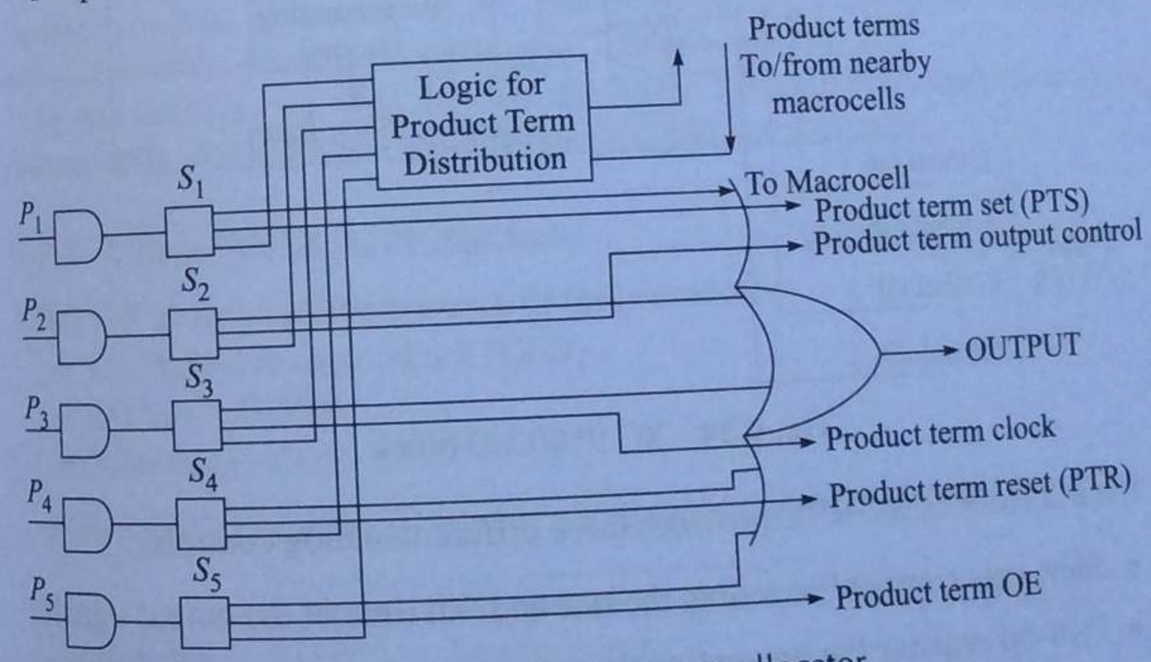


Fig. 8.32 Product term allocator

The macrocell with the function block basically contains a flip-flop with some programmable multiplexers to drive the inputs of the flip-flop. It can be represented as shown in Fig. 8.33.

Multiplexers are programmed for selecting the sources to drive the flip-flop set/reset and clock inputs as well as selecting the combinational/registered output. The flip-flop can be programmed to behave as a T or D flip-flop.

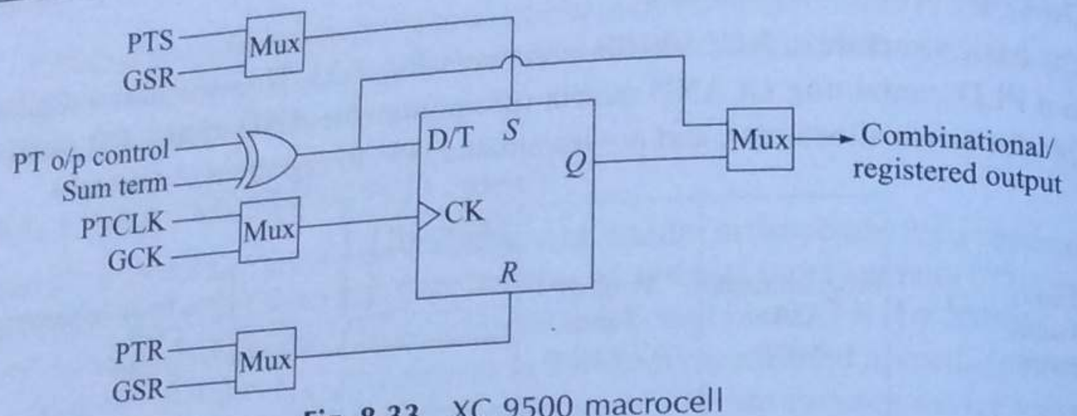


Fig. 8.33 XC 9500 macrocell

I/O block of XC 9500

All the external I/O pins of XC 9500 family get routed through an I/O block before getting connected to the switch matrix. An I/O block is shown in Fig. 8.34.

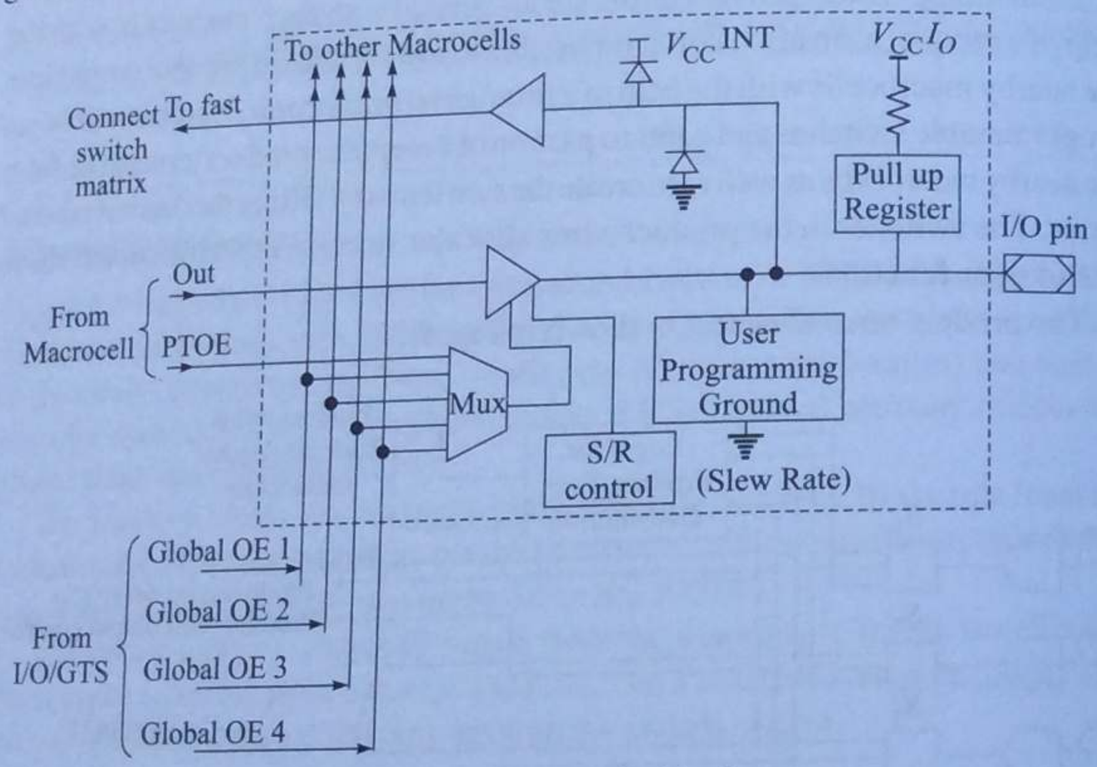


Fig. 8.34 XC 9500 I/O block

The I/O block of XC 9500 provides three different analog controls.

- **Slew rate control** For setting the rise and fall time of the output signals.
- **Pull-up register** For preventing the output pins from floating at power-up.
- **User programmable ground** For providing extra ground pins to handle the high dynamic currents at switching.

I/O block also provides various digital controls, such as the output can be always on, always off, controlled by PTOE or global OE pins.

This block provides compatibility of devices operating at different supply voltages.

Switch matrix

The switch matrix is built in a chip as a rectangular structure, with a column for each input, a row for each output and a pass transistor at each cross point to control the connection between the given input and output.

8.7 FIELD PROGRAMMABLE GATE ARRAY

Programmable Gate Arrays (PGAs) are the devices which are similar to CPLDs having more flexibility and can be used to implement a complete digital system on a single IC chip. A PGA is an IC which contains an array of identical logic cells with programmable interconnections. An FPGA's programmable logic cell is less than a typical PLD, but an FPGA chip contains much more logic than a CPLD of the same size. In PGA IC, the user can program the functions realized by each logic cell and connections between the cells. Such PGAs are called as FPGA, since they are field programmable.

Even though Xilinx invented FPGAs, two basic FPGA architectures have become popular which are from Xilinx and Actel. Both the approaches have blocks or modules that are configurable and both of them have switching matrix to interconnect these blocks.

Xilinx FPGA architecture typically has configurable logic blocks and 710 blocks with a programmable switching matrix. The switching matrix configuration data is stored in an external memory chip. By selecting this memory chip as EPROM or RAM, the configuration data can be changed, hence the device is reprogrammable.

In Actel FPGAs, there are simple basic cells called as *logic modules* and a complex switching matrix. This architecture, due to the complex switching matrix, is not reprogrammable. Once it is configured, it cannot be changed.

In this section, we will discuss the basic Xilinx FPGA architecture and subsequently, the popular FPGA series (XC 4000) architecture.

8.7.1 Xilinx FPGA Architecture

Xilinx FPGA is basically made up of three modules:

- Configurable logic block (CLB)
- I/O block (IOB)
- Switching matrix

Figure 8.35 shows the general block diagram of FPGA structure.

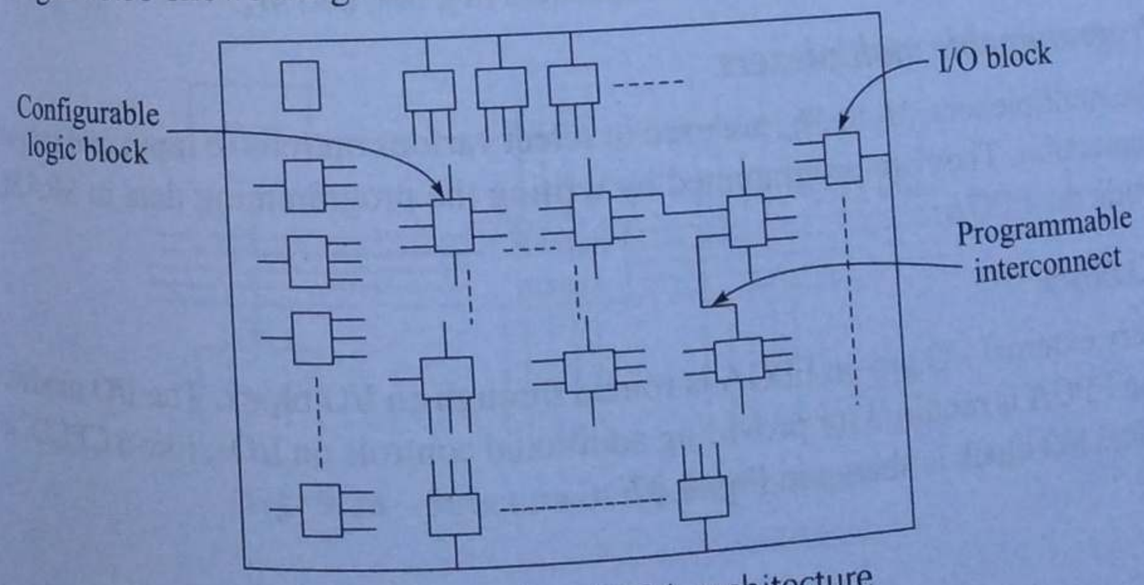


Fig. 8.35 Xilinx FPGA architecture

Configurable logic block

A configurable logic block consists of data multiplexers, flip-flops, and a combinational logic array. A combinational function is performed by a $2^n \times 1$ look-up table entries that realize any n-variable Boolean logic function, where n depends upon the FPGA family (i.e. XC 2000, XC 3000, XC 4000, etc.). The programmable multiplexers are used to route the data internally with the configurable logic block. Figure 8.36 shows the block diagram of a simple configurable logic block of XC 2000 series FPGA.

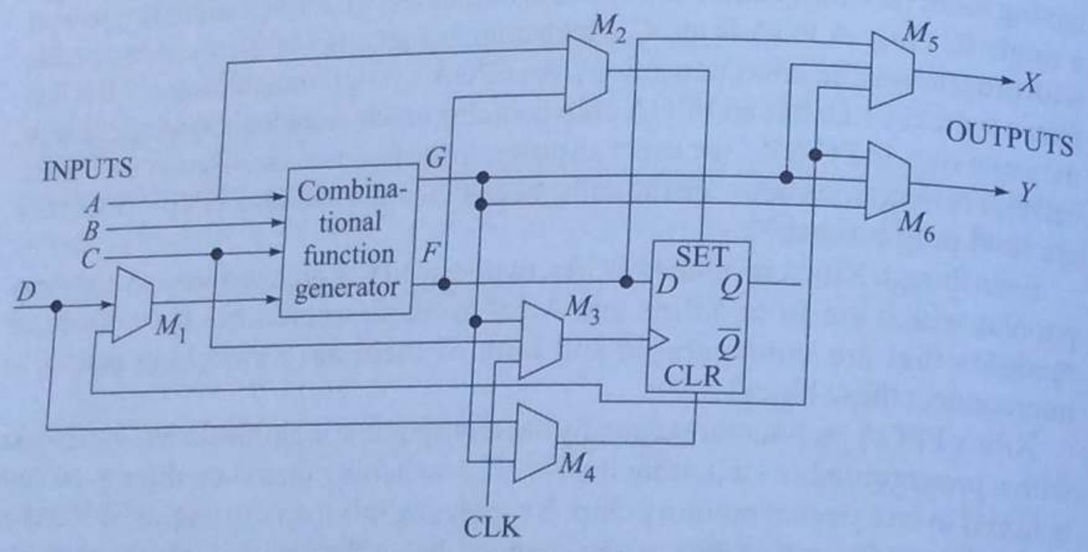


Fig. 8.36 Xilinx XC 2000 CLB

Combinational function generator

It accepts four external inputs A, B, C, D, and the output of D can be external or feedback from the output selected by multiplexer M_1. The combinational function generator produces two outputs, F and G. With the help of multiplexers M_5 and M_6, either combinational or registered outputs from F and G are available at the outputs, X and Y.

Flip-flop

The configurable logic block of XC 2000 contains a D flip-flop whose input D is driven by F output. Its SET, CLR, and CLK inputs can be driven by one of the different sources as selected by multiplexers M_2, M_3, and M_4.

Programmable multiplexers

The multiplexers, M_1 to M_6, are used to select various options of input and output connection. They are programmed by writing the programming data in SRAM, inside the FPGA.

I/O block

Every external I/O pin on FPGA is routed through an I/O block. The I/O module in an FPGA is required for providing additional controls on I/Os, like a CPLD. A typical I/O block is shown in Fig. 8.37.

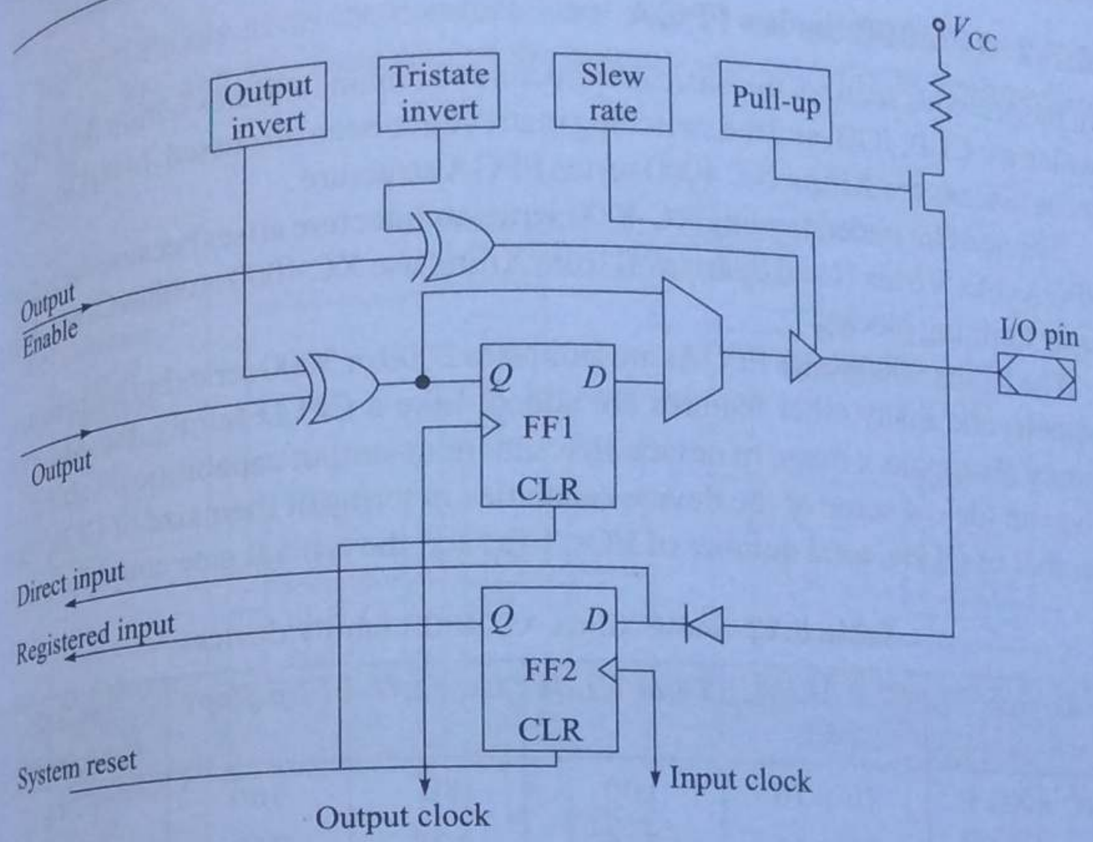


Fig. 8.37 Xilinx FPGA I/O block

The Xilinx FPGA programmable I/O block contains two flip-flops for registered input and registered output, which are connected to only one I/O pin, since it is bidirectional. The important controls which are provided in the I/O block of every I/O pin are:

- Output inversion control
- Slew rate control
- Tri-state control
- Active/Passive pull-up control

Switching matrix structure

A switching matrix provides inconnection among CLBs and IOBs inside an FPGA. Every switch provides direct interconnection between adjacent CLBs or IOBs. In addition, a set of vertical and horizontal lines provides facility to connect any two CLBs within the FPGA. The general structure of a switch matrix is shown in Fig. 8.38.

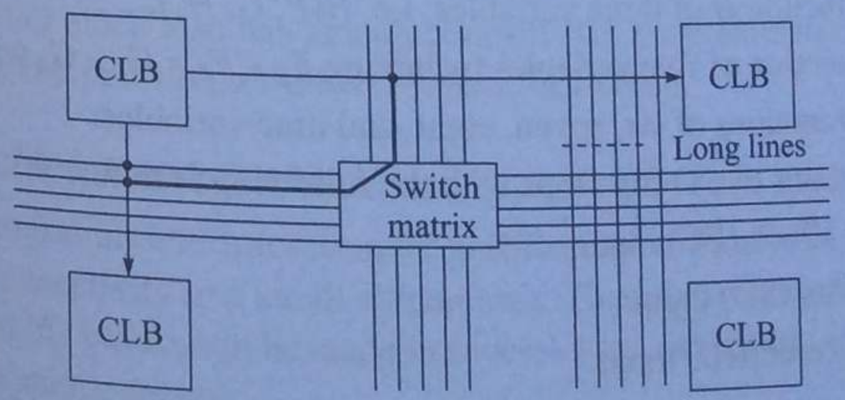


Fig. 8.38 Structure of the switch matrix

8.7.2 XC 4000 Series FPGA

In the previous section, the basic architectures of main modules within the FPGA which are CLB, IOB, and the switching matrix have been discussed. In this section, let us discuss the Xilinx XC 4000 series FPGA structure.

The need for understanding XC 4000 series architecture arises because the recent FPGAs like Virtex II and Spartan–II from Xilinx use XC 4000 architecture as their basic building blocks.

The Xilinx 4000 series FPGAs are similar to 2000 or 3000 series but more inputs, outputs, and many other features are added. Like a CPLD family, the XC 4000 family also spans a range of device size and input-output capabilities. Table 8.12 gives an idea of some of the device capacities in terms of their size of CLB, total number of CLBs, total number of I/Os, FFs, and the typical gate count for each.

Table 8.12 Some Xilinx XC 4000 family devices

Device	*CLB Matrix size*	*Total CLBs*	*User I/O*	*Flip-flops*	*Typical Gate count*
XC 4003 E	10 × 10	100	80	360	2k–5k
XC 4006 E	16 × 16	256	128	768	4k–12k
XC 4008 E	18 × 18	324	144	936	7k–15k
XC 4062 XL	48 × 48	2,304	384	5,376	40k–130k
XC 4085 XL	52 × 52	3,136	44	87,168	55k–180k

Let us see the main blocks of XC 4000 architecture, which are:

- Configurable logic block
- I/O block
- Switching matrix

Configurable logic block of XC 4000

Figure 8.39 shows a simplified block diagram of an XC 4000 CLB. It basically has nine logic inputs ($F_1 - F_4$, $G_1 - G_4$, and H_1) and three outputs F, G, and H. The combinational function generator can generate the outputs in the following manner:

- Two independent functions G and F of four variables, i.e. $G(G_1, G_2, G_3, G_4)$ and $F(F_1, F_2, F_3, F_4)$.
- One function H of three variables, i.e. $H(F, G, H_1)$.
- Any function of five variables by having $F_1 - F_4 = G_1 - G_4$ with H_1.
- Some functions of six, seven, eight, and nine variables.

The CLB contains two D flip-flops with the following features:

- Enable Clock (EC) input
- Direct Set (SD) input
- Direct Reset (RD) input

The above inputs and the inputs like D and CLK can be selected from multiple sources with the help of programmable multiplexers M_1, M_3, and $M_5 - M_8$, as shown in the block diagram.

The CLB finally gives the combinational as well as registered output on the pins, X, Y, XQ, and YQ, from one of the outputs selected with the multiplexers $M_1 - M_4$.

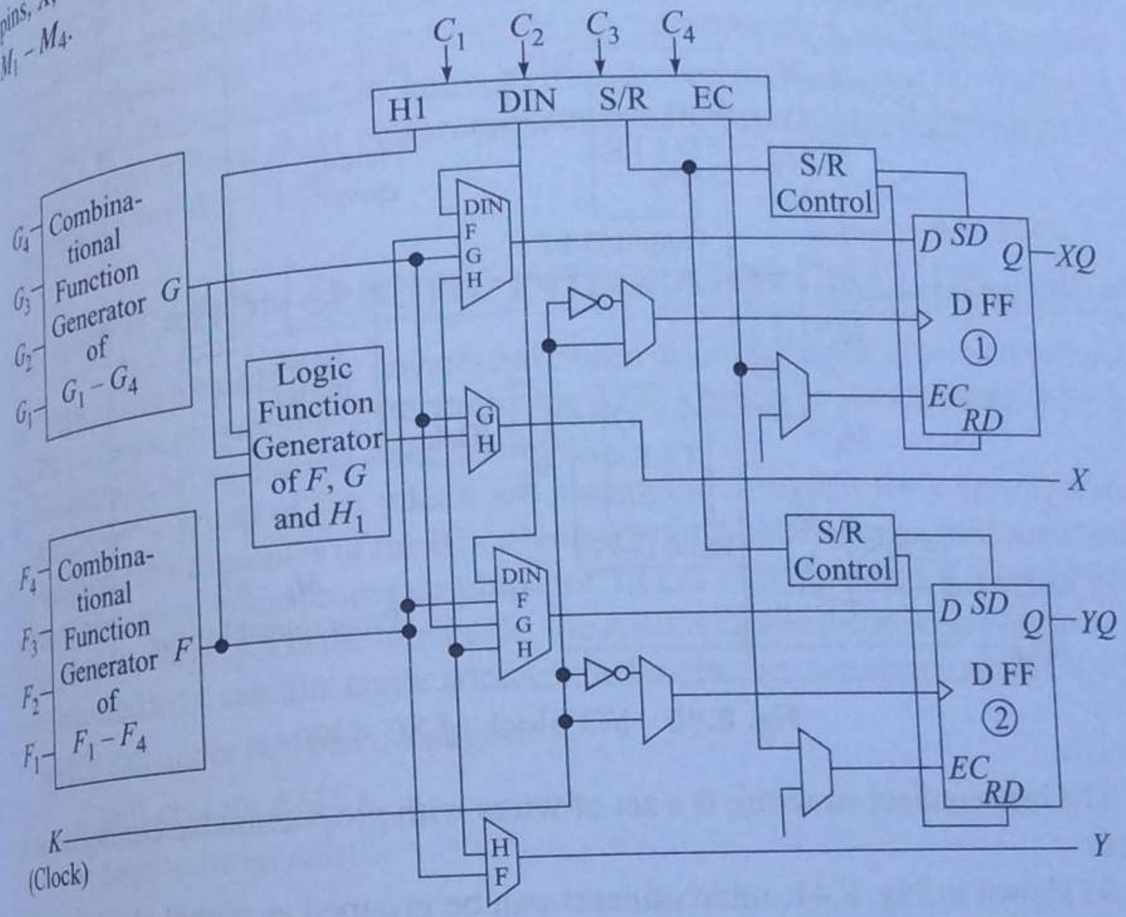


Fig. 8.39 XC 4000 series CLB structure

I/O Block of XC 4000

Any I/O pin on the FPGA can be used for input or output or both. Hence, like CPLD XC 9500 family, XC 4000 series FPGA also contains an IOB having more logic control than CPLD.

Figure 8.39 shows the I/O block of XC 4000 series FPGA. Every I/O pin has two D flip-flops associated with it. The input and output paths within the FPGA contain these D flip-flops selectable by multiplexers M_5 and M_7

The I/O block's other logic controls are polarity selection of the four signals which are output bits (OUT), three-state enable signal (T), an output clock (OCLK), and an input clock (ICLK) enable with the help of multiplexers $M_1 - M_4$.

XC 4000 I/O block also has analog controls like programmable slew rate of output driver and pull-up/pull-down resistor for I/O pin.

Programmable interconnects

XC 4000 programmable interconnect architecture provides very dense and symmetric connectivity in a small silicon area. These interconnections between the CLBs and I/O blocks can be made in several ways, which are:

- Direct interconnects
- General-purpose interconnects
- Long lines

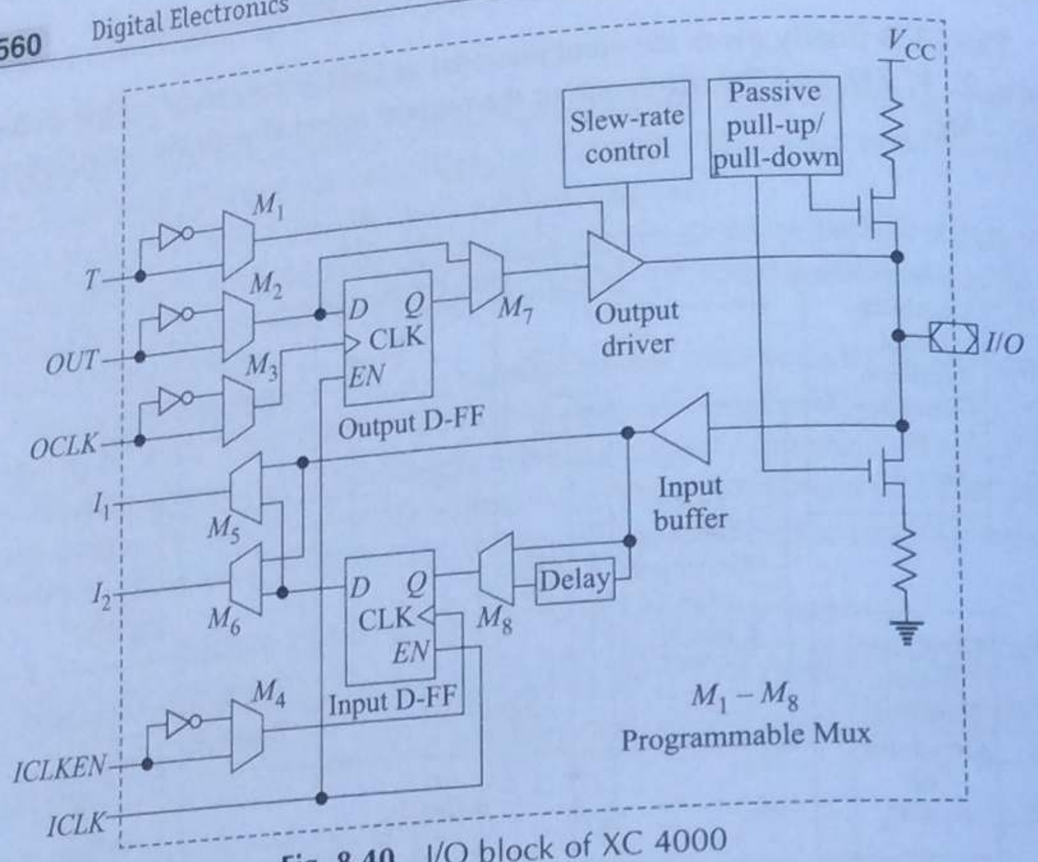


Fig. 8.40 I/O block of XC 4000

The interconnect structure is a set of wires with programmable connections to them.

As shown in Fig. 8.41, interconnects can be grouped as global clocks, single lines, double lines, and long lines. The Global Clocks are optimized as clock inputs to CLBs which provide short delay and minimal slew. The single lines are made for adjacent CLB connection. Double and long lines are optimized for shorter delay for longer connections.

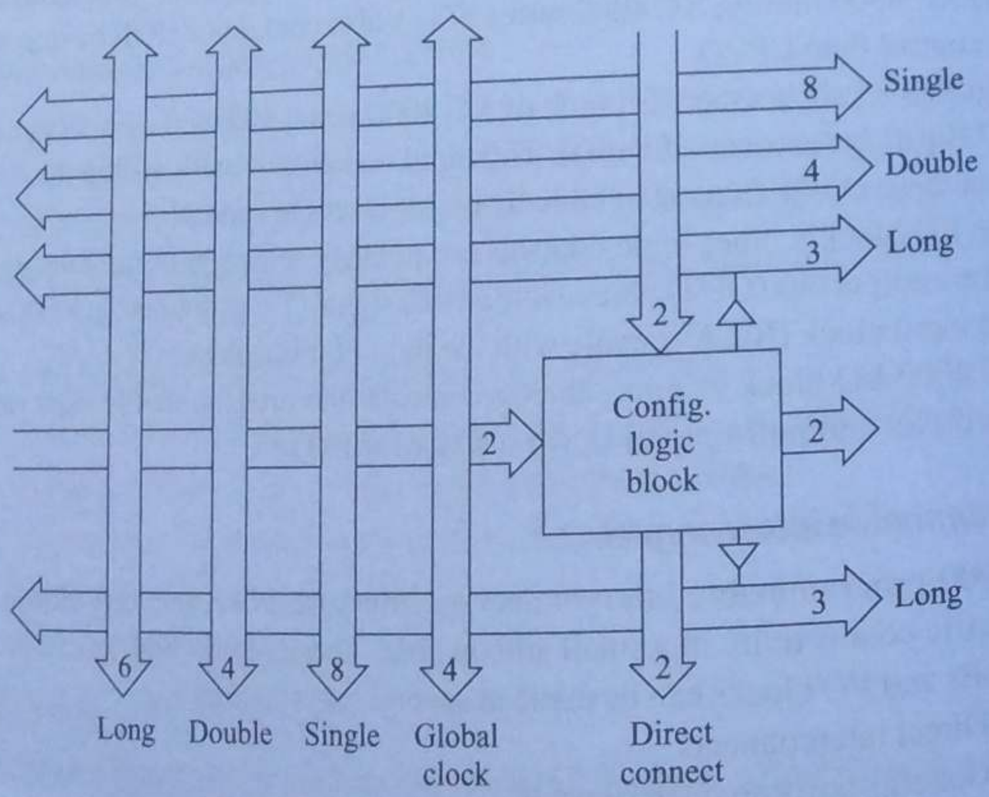


Fig. 8.41 Programmable interconnects

Additionally, a specialized programmable interconnect is provided at the edges of the CLB array for connections to the I/O blocks.

FPGA programming is done with advanced software tools from the manufacturers, which is referred to as loading the desired bit pattern in the configuration memory cells. Before it is done, the software tools called as *place and route* or *fitter* are used to optimize the device performance by finding a placement that allows short connections and then, routing the connections by searching for suitable CLBs.

8.8 APPLICATION-SPECIFIC INTEGRATED CIRCUITS (ASICs)

Different programmable logic devices have been discussed in the previous sections. All PLDs, once programmed, become type from a group of semi-custom ASICs. In this section, let us discuss ASICs in brief.

ASICs are a group of ICs which are meant and designed for a very specific application. The examples of the ICs which are not ASICs are standard parts such as memory chips, microprocessor chips, or TTL/TTL equivalent ICs under SSI, MSI, or LSI category. On the other side, the ASICs can be those which are used in sophisticated toys, satellite applications, interface between memory and CPU, etc.

ASICs fall under two basic categories:

- Full-custom ASICs
- Semi-custom ASICs

8.8.1 Full-custom ASICs

Basically, a full-custom IC includes almost all logic cells (like PGA/CPLD) and interconnects that are customized. This allows designers to include analog circuits, optimized memory cells, or even mechanical structures on an IC. Hence, in a full-custom ASIC, an engineer designs some or all of the logic cells, circuits, and layout, specifically for one ASIC. These ICs are expensive to manufacture and design. Hence, very few full-customs ASICs are designed. A popular type of full-custom application is the mixed analog/digital ASIC.

8.8.2 Semi-custom ASICs

In semi-custom ASICs, some of the logic cells are predesigned and some of the interconnects are customized. The cell library of predesigned logic cells is very convenient for implementing the design.

The two basic types of semi-custom ASICs are:

- Standard cell-based ASICs
- Gate array-based ASICs

Programmable ASICs

Programmable ASICs are the ones in which all the logic cells are predesigned and none of the interconnects are customized. The basic types of programmable ASICs are:

- Programmable logic devices
- Field programmable gate arrays

Standard cell-based ASICs

The cell-based ASIC uses predesigned logic cells (AND gates, OR gates, multiplexers, flip-flops, etc.) which are called as *standard cells*. A standard cell area is made up of rows of standard cells and these areas may be used in combination with larger predesigned cells. The designer only defines the placement of standard cells and interconnects.

Figure 8.42 shows the internal block diagram (Die) of a standard cell-based ASIC.

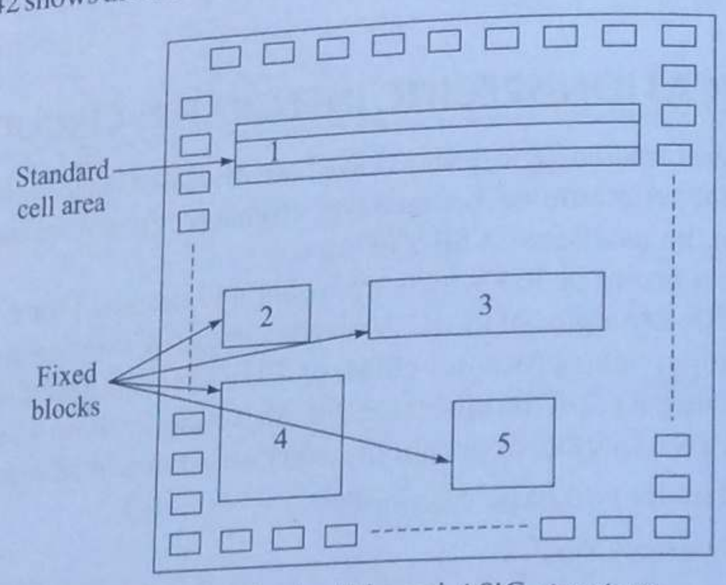


Fig. 8.42 Standard cell-based ASIC structure

Gate array-based ASICs

In a gate array-based ASIC, the transistors are predefined on the silicon wafer. The designer chooses from a gate array library of predesigned and pre-characterized logic cells. The logic cells in the library are called *macros*.

There are three different types of gate array-based ASICs:

- Channelled gate arrays
- Channel-less gate arrays
- Structured gate arrays

A *channelled gate array* is similar to a cell-based IC which uses rows of cells separated by channels used for interconnect. Only in this type, the space for interconnect is fixed. Figure 8.41 shows the arrangement in a channelled gate array.

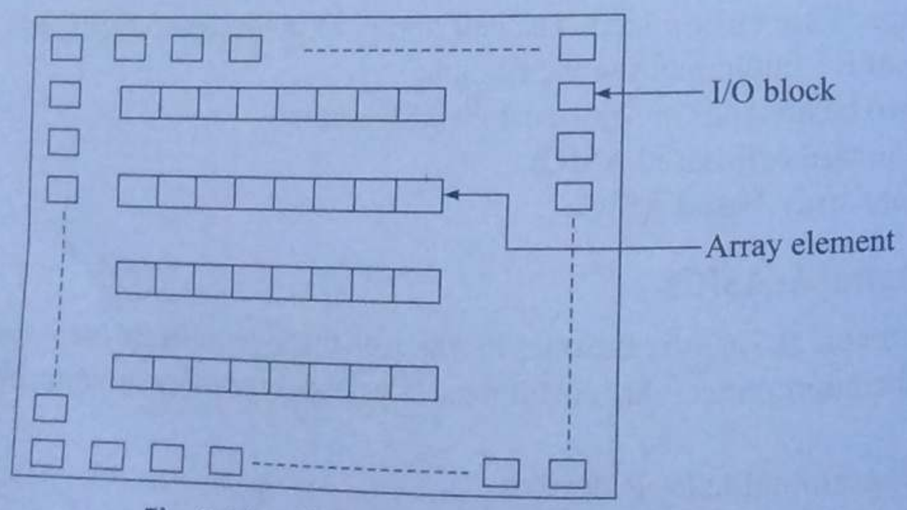


Fig. 8.43 Channelled gate array structure

In *channel-less gate array*, there are no predefined areas set aside for routing between cells; instead, the routing is done over the top of gate array devices. This structure is also referred to as the *sea-of-gates* or *SOG array*.

Figure 8.42 shows the arrangement in a channelless gate array.

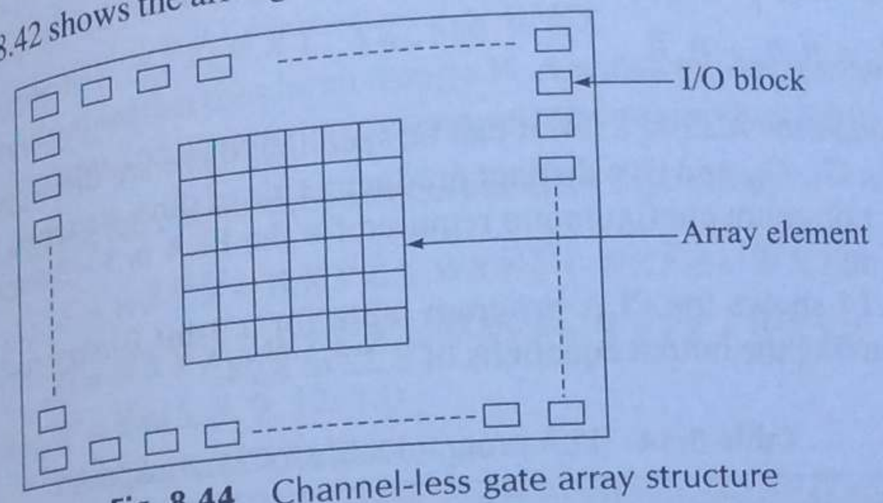


Fig. 8.44 Channel-less gate array structure

In a *structured gate array*, some area of the IC is set aside and dedicated to a specific function. This is referred to as the *embedded area*.

SOLVED EXAMPLES

Example 8.6 Design a 3-bit Binary-to-Gray code converter using a suitable PLA.

Solution

The truth table for a 3-bit Binary-to-Gray code converter is shown in Table 8.13.

Table 8.13 Truth table for Binary-to-Gray code converter

Binary Code			*Grey Code*		
B_2	B_1	B_0	G_2	G_1	G_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

K-maps for outputs G_2, G_1, and G_0 in terms of B_2, B_1, and B_0 are shown in Fig. 8.45.

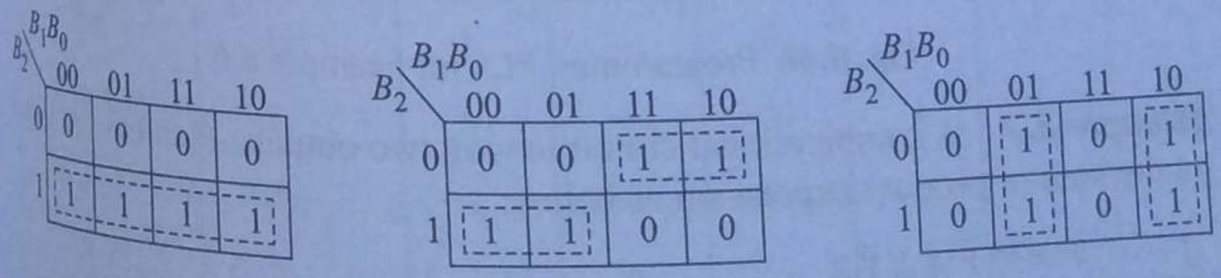


Fig. 8.45 K-maps for G_2, G_1, and G_0

The simplified expression in SOP form for the outputs G_2, G_1, and G_0 are given by the following equations.

$$G_2 = B_2 \tag{8.22}$$

$$G_1 = B_2\overline{B}_1 + \overline{B}_2 B_1 \tag{8.23}$$

$$G_0 = \overline{B}_1 B_0 + B_1 \overline{B}_0 \tag{8.24}$$

From equations 8.22 to 8.24, it can be seen that there are three outputs sum-terms of G_2, G_1, G_0, and five distinct product terms of three variables B_2, B_1, B_0. Hence, the minimum configuration required for the PLA is 3×3 outputs with 5 product terms.

Table 8.14 shows the PLA program table for a 3-bit Binary-to-Gray code converter having the output equations of 8.22 to 8.24.

Table 8.14 PLA program table for Example 8.6

Product terms		*Inputs*			*Outputs*		
		B_2	B_1	B_0	G_1	G_2	G_0
B_2	1	1	–	–	1	–	–
$B_2\overline{B}_1$	2	1	0	–	–	1	–
$\overline{B}_2 B_1$	3	0	1	–	–	1	–
$\overline{B}_1 B_0$	4	–	0	1	–	–	1
$B_1\overline{B}_0$	5	–	1	0	–	–	1
					T	*T*	*T*

Figure 8.45 shows the diagram of a 3×3 PLA with 5 product terms programmed.

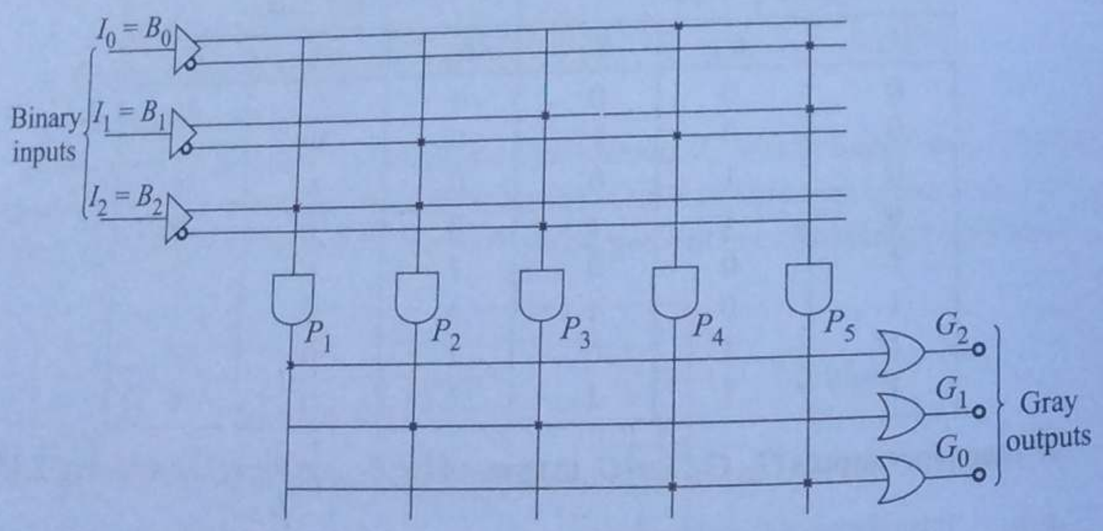


Fig. 8.46 Programmed PLA for Example 8.6

Example 8.7 A combinational circuit having two outputs, F and G, are given by the sum of product expression as below:

$$F = W\overline{Y} + X\overline{Y}Z$$

$$G = W\overline{X}\,\overline{Y} + \overline{X}Z + \overline{W}Z$$

where W, X, Y, Z are inputs to the system. Implement this circuit using PLA having no more than four product terms.

Solution

From the equations of the outputs, F and G, it is observed that there are five distinct product terms $W\overline{Y}$, $X\overline{Y}Z$, $W\overline{X}\,\overline{Y}$, $\overline{X}Z$, and $\overline{W}\,\overline{Y}Z$

In order to implement the circuit using a PLA with 4 product terms, we need to change this expression so as to have common (shared) product terms.

By converting the output equations in standard SOP form,

$$F = W\overline{X}\,\overline{Y}\,\overline{Z} + W\overline{X}\,\overline{Y}Z + WX\overline{Y}\,\overline{Z} + WX\overline{Y}Z + \overline{W}X\overline{Y}Z$$

$$G = W\overline{X}\,\overline{Y}Z + W\overline{X}\,\overline{Y}\,\overline{Z} + \overline{W}\,\overline{X}YZ + W\overline{X}YZ + \overline{W}X\overline{Y}Z$$

That is, $F = \Sigma m(5, 8, 9, 12, 13)$

$G = \Sigma m(1, 3, 5, 8, 9, 11)$

K-maps for outputs F and G in terms of X, Y, Z, and W are shown in Fig. 8.47.

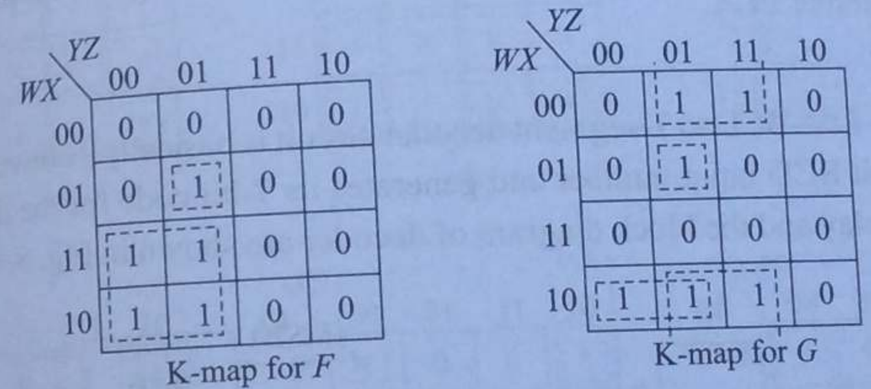


Fig. 8.47 K-maps for F and G

The expressions for F and G having only 4 distinct product terms are:

$$F = W\overline{Y} + \overline{W}X\overline{Y}Z \tag{8.25}$$

$$G = W\overline{X}\,\overline{Y} + \overline{X}Z + \overline{W}X\overline{Y}Z \tag{8.26}$$

The product terms are: $W\overline{Y}$, $\overline{W}X\overline{Y}Z$, $W\overline{X}\,\overline{Y}$, and $\overline{X}Z$.

Table 8.15 shows the PLA program table for PLA having 4 inputs, 2 outputs, and 4 product terms. Figure 8.48 shows the programmed PLA.

Table 8.15 PLA program table for Example 8.7

Product terms		*Inputs*				*Outputs*	
		W	X	Y	Z	F	G
$W\overline{Y}$	1	1	–	0	–	1	–
$\overline{W}X\overline{Y}Z$	2	0	1	0	1	1	1
$W\overline{X}\,\overline{Y}$	3	1	0	0	–	–	1
$\overline{X}Z$	4	–	0	–	1	–	1
						T	*T*

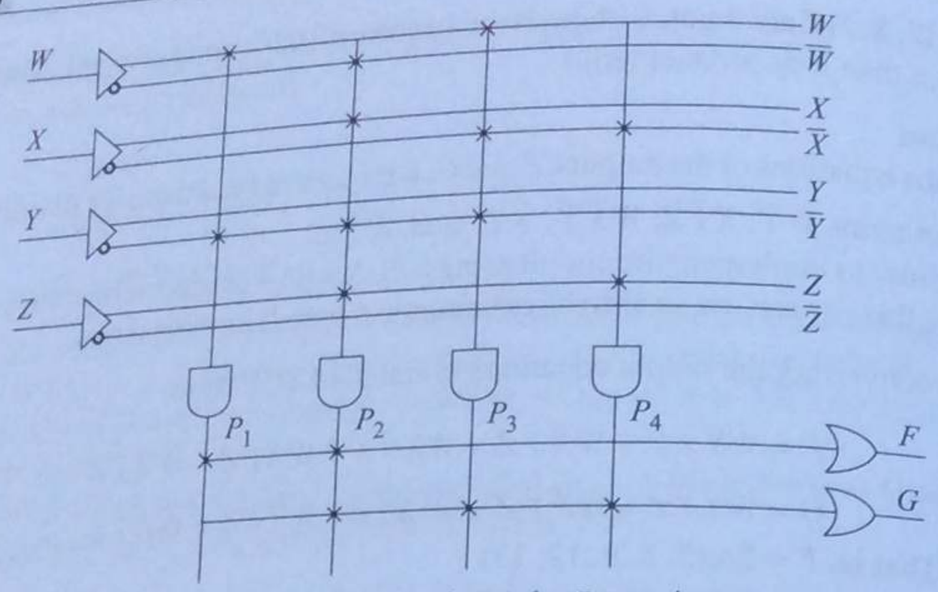


Fig. 8.48 Programmed PLA for Example 8.7

Example 8.8 Design a BCD to 7-segment decoder for common cathode display, using a suitable PLA.

Solution

Approach 1 : A BCD to 7-segment decoder circuit is basically a converter which takes a 4-bit BCD input number and generates its 7-bit code for the driver. The type of display and the block diagram of decoder are shown in Fig. 8.49.

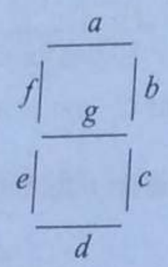


Fig. 8.49(a) Common cathode display

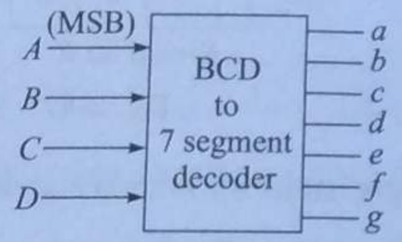


Fig. 8.49(b) Block diagram of decoder

The truth table of the display is given in Table 8.16.

Table 8.16 Truth table of BCD to 7-segment decoder

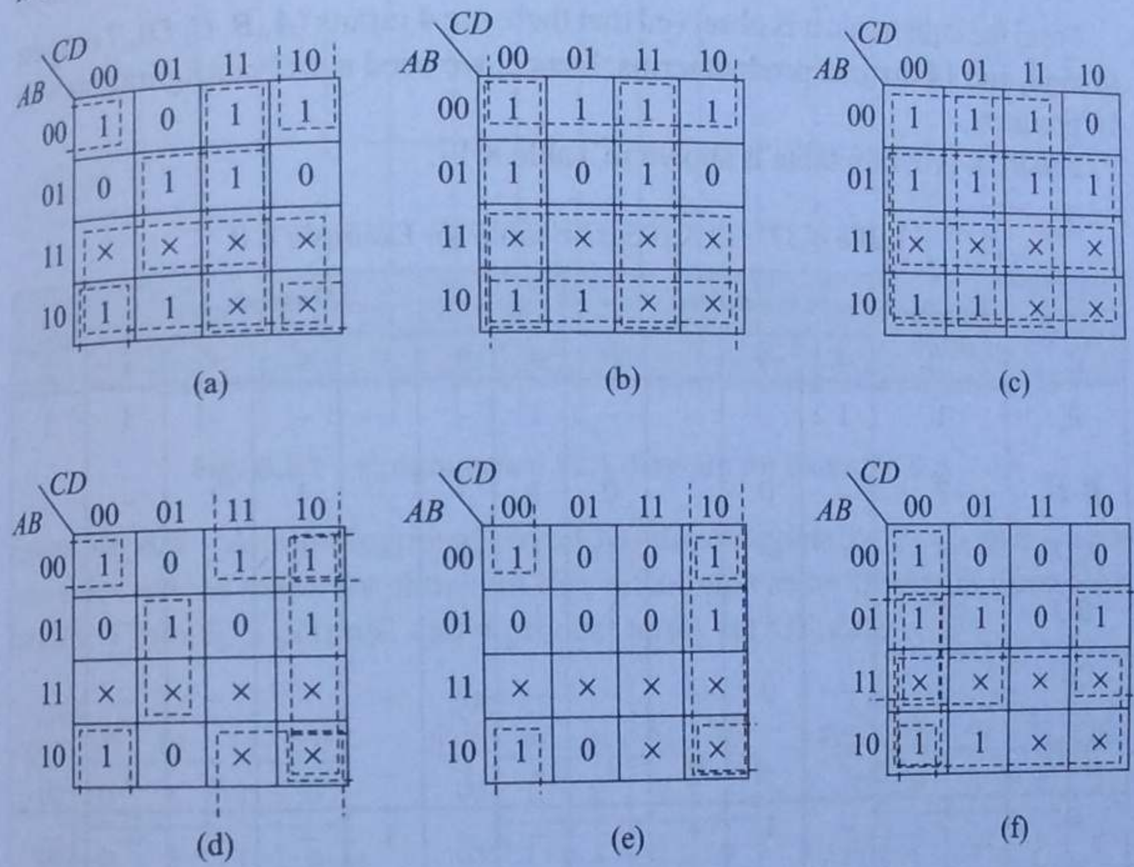
Decimal digit	Inputs				Outputs						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1

Contd.

Contd.

Decimal digit	Inputs				Outputs						
	A	B	C	D	a	b	c	d	e	f	g
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

Figure 8.50 shows the K-maps for outputs *a* to *g* by considering the output to be *X* for invalid BCD inputs.



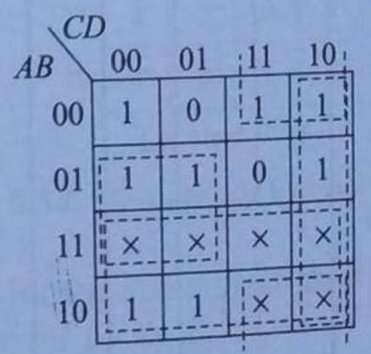


Fig. 8.50 K-maps for outputs

The K-maps are simplified and the minimum expressions are derived, as given in equations 8.27 to 8.33.

$$a = A + \overline{B}\,\overline{D} + CD + BD \tag{8.27}$$

$$b = \overline{B} + \overline{C}\,\overline{D} + CD \tag{8.28}$$

$$c = B + \overline{C} + D \tag{8.29}$$

$$d = \overline{B}\,\overline{D} + C\overline{D} + \overline{B}C + B\overline{C}D \tag{8.30}$$

$$e = \overline{B}\,\overline{D} + C\overline{D} \tag{8.31}$$

$$f = A + \overline{C}\,\overline{D} + B\overline{C} + B\overline{D} \tag{8.32}$$

$$g = A + C\overline{D} + B\overline{C} + \overline{B}C \tag{8.33}$$

From the equations, it is observed that there are 4 inputs (A, B, C, D), 7 outputs (a to g), and 14 distinct product terms. Hence, we need a 4×7 configuration with 14 products.

The PLA program table is shown in Table 8.17.

Table 8.17 PLA program table for Example 8.8

	Product terms	Inputs				Outputs						
		A	B	C	D	a	b	c	d	e	f	g
A	1	1	–	–	–	1	–	–	–	–	1	1
$\overline{B}\,\overline{D}$	2	–	0	–	0	1	–	–	1	1	–	–
CD	3	–	–	1	1	1	1	–	–	–	–	–
BD	4	–	1	–	1	1	–	–	–	–	–	–
$\overline{B}$	5	–	0	–	–	–	1	–	–	–	–	–
$\overline{C}\,\overline{D}$	6	–	–	0	0	–	1	–	–	–	1	–
B	7	–	1	–	–	–	–	1	–	–	–	–
$\overline{C}$	8	–	–	0	–	–	–	1	–	–	–	–
D	9	–	–	–	1	–	–	1	–	–	–	–
$C\overline{D}$	10	–	–	1	0	–	–	–	1	1	–	1
$\overline{B}C$	11	–	0	1	–	–	–	–	1	–	–	1
$B\overline{C}D$	12	–	1	0	1	–	–	–	1	–	–	–
$B\overline{C}$	13	–	1	0	–	–	–	–	–	–	1	1
$B\overline{D}$	14	–	1	–	0	–	–	–	–	–	1	–
						T	T	T	T	T	T	T

Figure 8.51 shows the diagram of a 4×7 PLA with 14 product terms, implementing the BCD to 7-segment decoder.

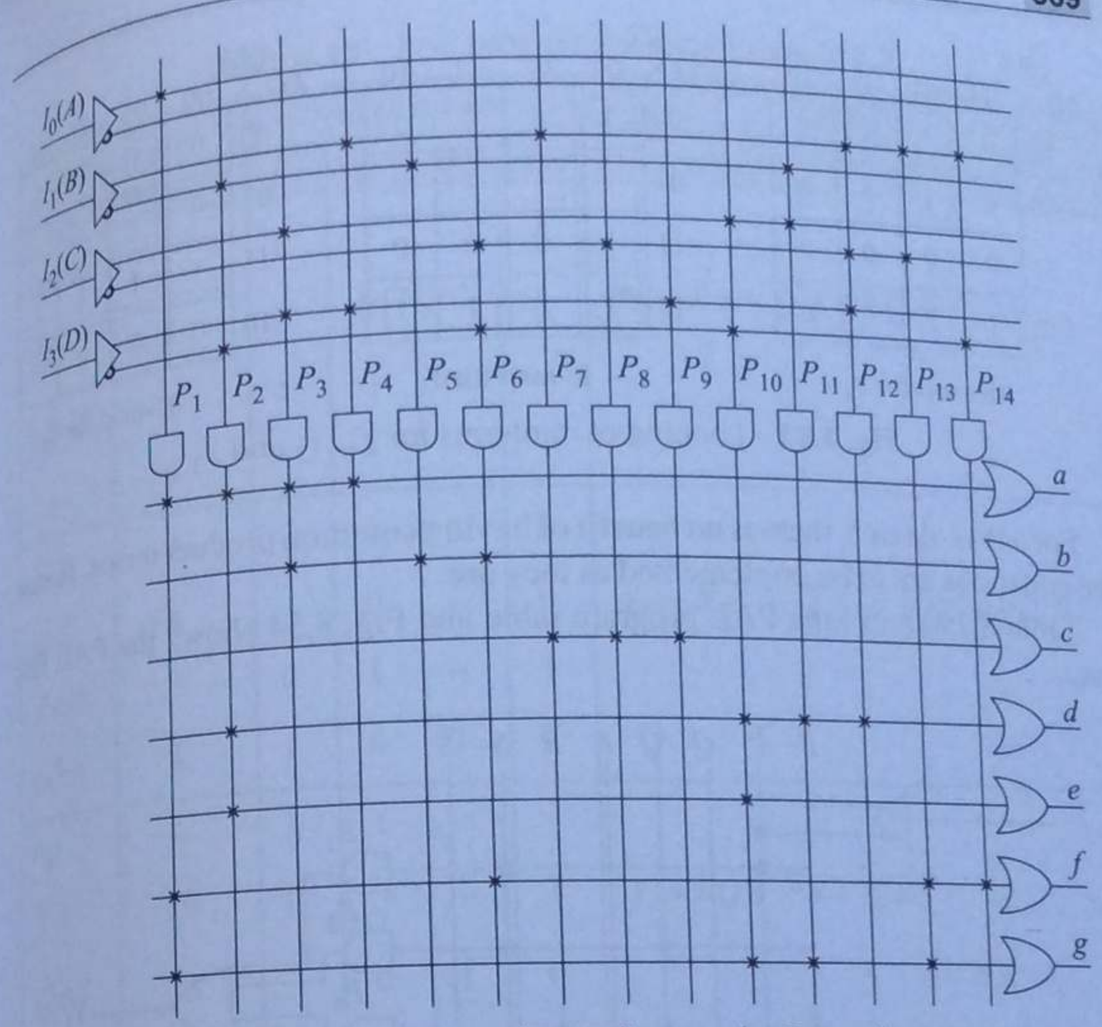


Fig. 8.51 Programmed PLA diagram for Example 8.8

Example 8.9 A combinational circuit has three outputs Y_1, Y_2, Y_3, and 4 input variables whose maps are shown in Fig. 8.52. Implement the circuit using PAL having 4 inputs, 3 outputs, and 4 product terms per OR gate.

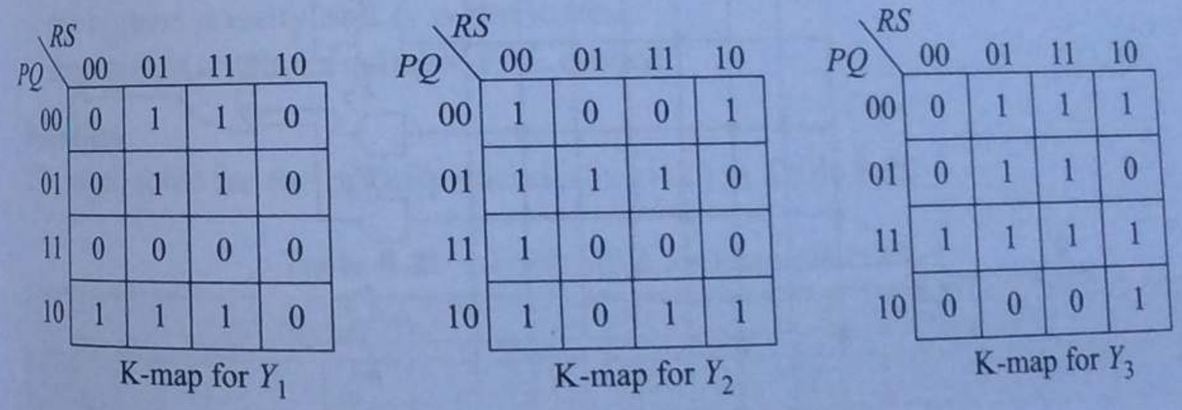


Fig. 8.52 K-maps for Example 8.9

Solution

The K-maps with looping for the minterms is shown in Fig. 8.53, to obtain the sum of products expression for Y_1, Y_2, Y_3.

Equations for Y_1, Y_2, Y_3 are:

$$Y_1 = \overline{P}S + \overline{Q}S + P\overline{Q}\,\overline{R} \tag{8.34}$$

$$Y_2 = \overline{R}\,\overline{S} + \overline{Q}\,\overline{S} + \overline{P}QS + P\overline{Q}R \tag{8.35}$$

$$Y_3 = PQ + \overline{P}S + \overline{Q}R\overline{S} \tag{8.36}$$

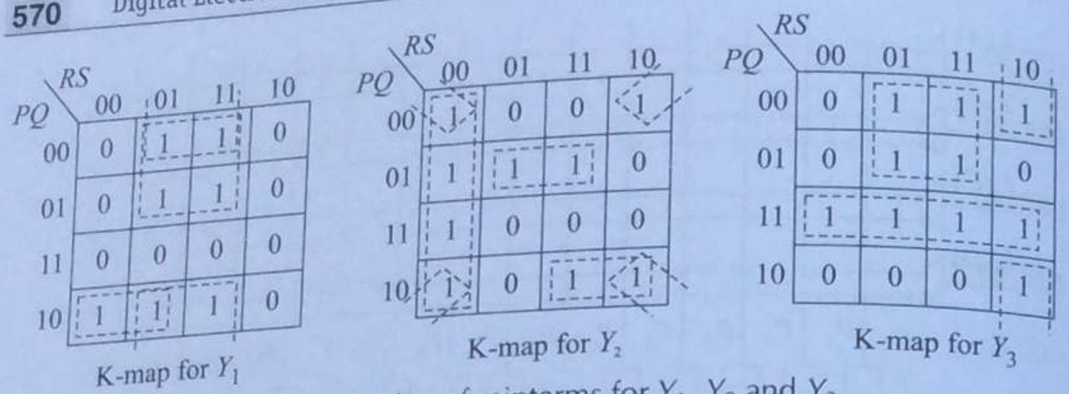


Fig. 8.53 Looping of minterms for Y_1, Y_2 and Y_3

For a PAL device, there is no benefit of having common product terms. Hence, the equations are to be implemented as they are.

Table 8.19 shows the PAL program table and Fig. 8.54 shows the PAL fuse map.

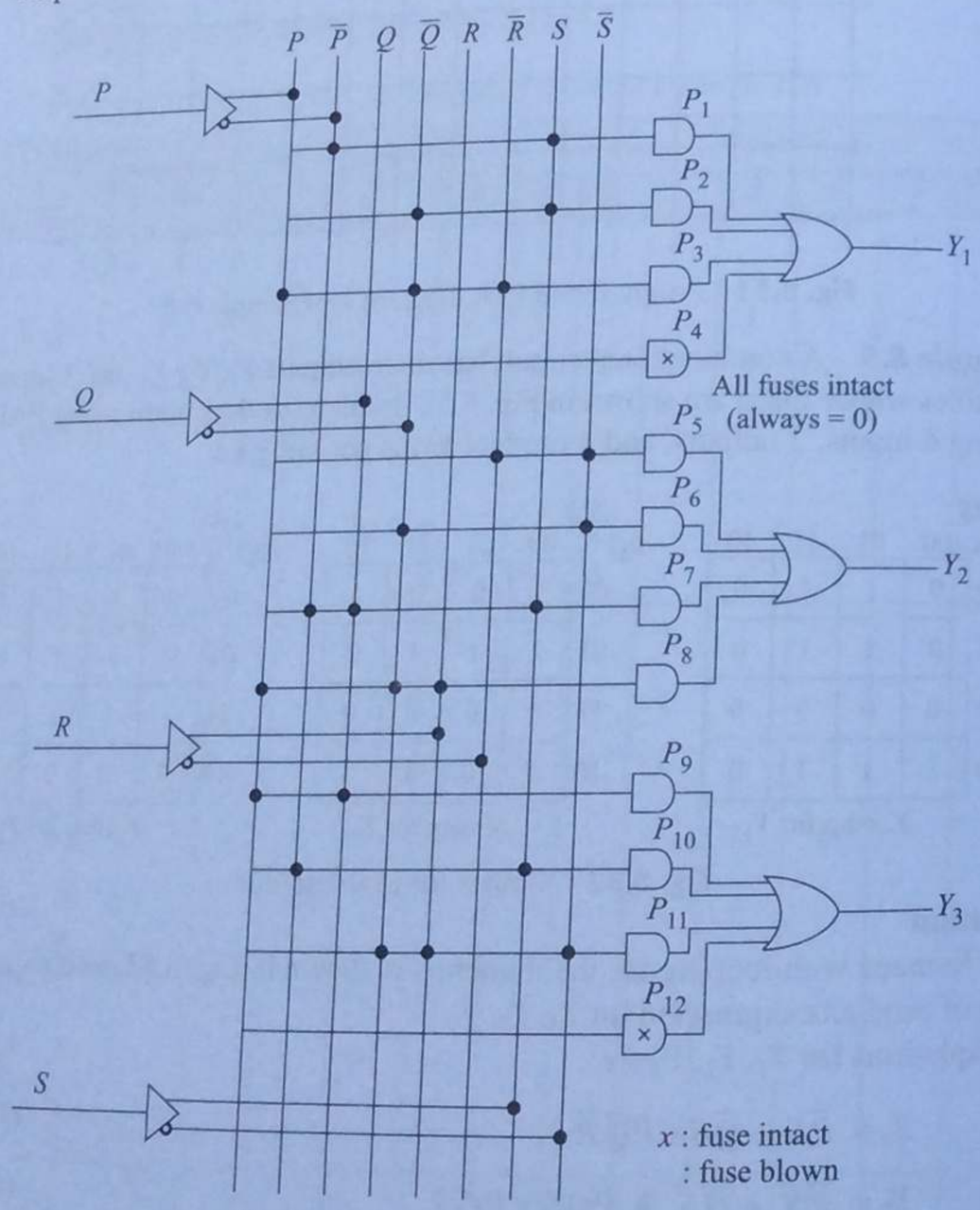


Fig. 8.54 PAL fuse map diagram

Table 8.19 PAL program table for Example 8.9

Product terms		AND inputs P	Q	R	S	Outputs
$\overline{P}S$	1	0	–	–	1	$Y_1 = \overline{P}S + \overline{Q}S + P\overline{Q}\,\overline{R}$
$\overline{Q}S$	2	–	0	–	1	
$P\overline{Q}\,\overline{R}$	3	1	0	0	–	
–	4	–	–	–	–	
$\overline{R}\,\overline{S}$	5	–	–	0	0	$Y_2 = \overline{R}\,\overline{S} + \overline{Q}\,\overline{S} + \overline{P}QS + P\overline{Q}R$
$\overline{Q}\,\overline{S}$	6	–	0	–	0	
$\overline{P}QS$	7	0	1	–	1	
$P\overline{Q}R$	8	1	0	1	–	
PQ	9	1	1	–	–	$Y_3 = PQ + \overline{P}S + \overline{Q}R\overline{S}$
$\overline{P}S$	10	0	–	–	1	
$\overline{Q}R\overline{S}$	11	–	0	1	0	
–	12	–	–	–	–	

Example 8.10 Design a priority encoder circuit with 4 active high inputs I_0, I_1, I_2, I_3 and 3 active high outputs, A and B indicating the number of highest priority device requesting service and N, indicating no active requests. Input I_0 is the highest priority and I_3 is the lowest.

Implement it using a suitable PAL device.

Solution

The truth table for the priority encoder is given in Table 8.20.

Table 8.20 Truth table for Example 8.10

Inputs I_0	I_1	I_2	I_3	Outputs A	B	N
0	0	0	0	X	X	1
1	X	X	X	0	0	0
0	1	X	X	0	1	0
0	0	1	X	1	0	0
0	0	0	1	1	1	0

From the table, we have

$$N = \overline{I}_0\,\overline{I}_1\,\overline{I}_2\,\overline{I}_3$$

The K-maps for A and B are shown in Fig. 8.55.

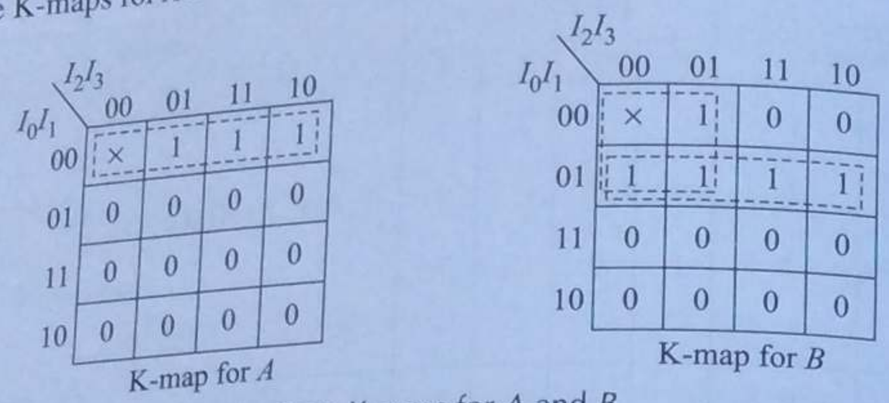


Fig. 8.55 K-maps for A and B

The equations for A and B are:

$$A = \bar{I}_0 \bar{I}_1 \tag{8.37}$$

$$B = \bar{I}_0 I_2 + \bar{I}_0 I_1 \tag{8.38}$$

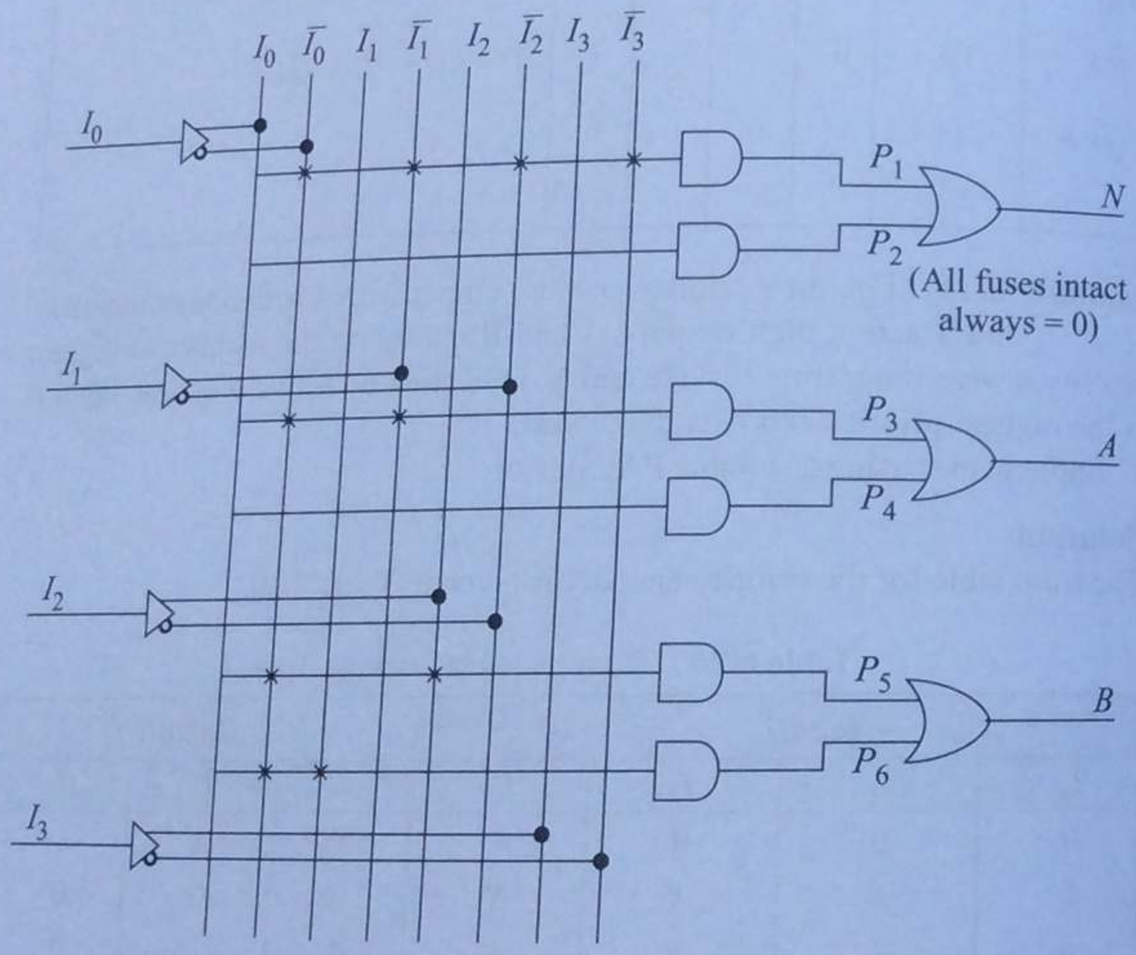


Fig. 8.56 Programmed PAL for Example 8.10

Hence, there is need to use a PAL having 3 inputs, 3 outputs with minimum 2 product terms (AND gates) per output OR gate. Table 8.21 shows the PAL programming table for the 3×3 PAL and Fig. 8.56 shows its fuse map.

Table 8.21 PAL program table for Example 8.10

Product terms		Inputs I_0	I_1	I_2	I_3	Outputs
$\bar{I}_0\bar{I}_1\bar{I}_2\bar{I}_3$	1	0	0	0	0	$N = \bar{I}_0\bar{I}_1\bar{I}_2\bar{I}_3$
	2	–	–	–	–	
$\bar{I}_0\bar{I}_1$	3	0	0	–	–	$A = \bar{I}_0\bar{I}_1$
	4	–	–	–	–	
$\bar{I}_0 I_2$	5	0	–	1	–	$B = \bar{I}_0 I_2 + \bar{I}_0 I_1$
$\bar{I}_0 I_1$	6	0	1	–	–	

Example 8.11 Derive PLA and PAL programming tables for a combinational circuit that squares a 3-bit number.

Solution

Let us consider a 3-bit input number having the bits as I_2, I_1 and I_0 and the 6 bit-output number ($7^2 = 49$ or 110001) to be Y_5 to Y_0.

The truth table for the circuit is given in Table 8.22.

Table 8.22 Truth table for Example 8.11

Decimal No.	Inputs I_2	I_1	I_0	Outputs Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	Decimal No.
0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	1	1
2	0	1	0	0	0	0	1	0	0	4
3	0	1	1	0	0	1	0	0	1	9
4	1	0	0	0	1	0	0	0	0	16
5	1	0	1	0	1	1	0	0	1	25
6	1	1	0	1	0	0	1	0	0	36
7	1	1	1	1	1	0	0	0	1	49

The K-maps for Y_5 to Y_0 are given in Fig. 8.56 and equations in equation 8.13.

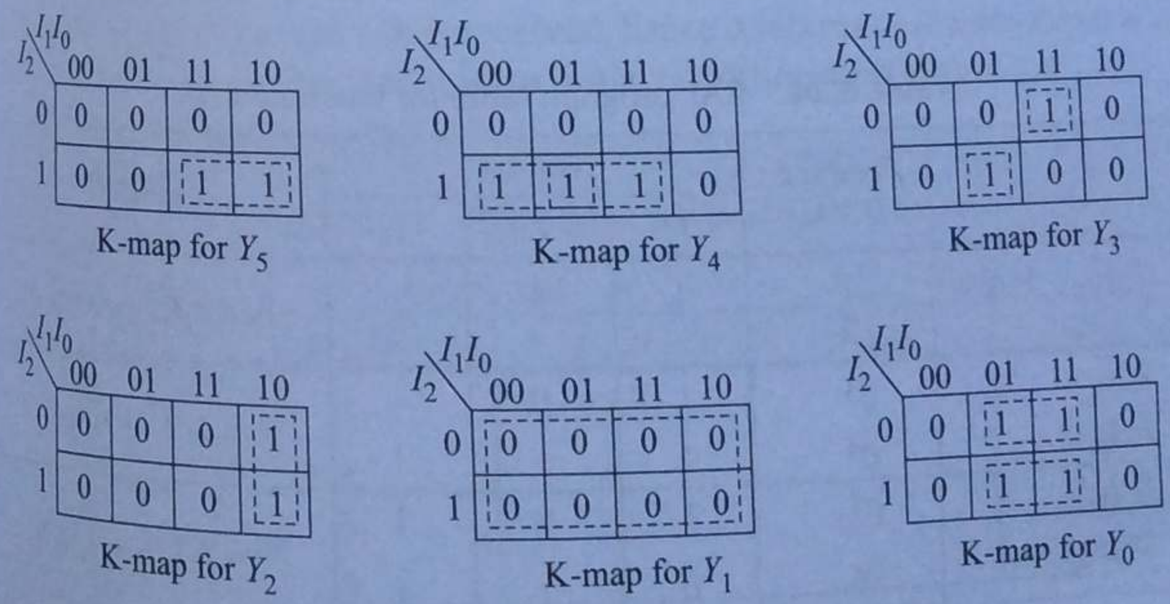


Fig. 8.57 K-maps Y_5 to Y_0

The sum of product expressions for Y_5 to Y_0 are as follows:

$$Y_5 = I_2 \cdot I_1 \tag{8.39}$$

$$Y_4 = I_2\bar{I}_1 + I_2I_0 \tag{8.40}$$

$$Y_3 = I_2\bar{I}_1I_0 + \bar{I}_2I_1I_0 \tag{8.41}$$

$$Y_2 = I_1\bar{I}_0 \tag{8.42}$$

$$Y_1 = 0^* \tag{8.43}$$

$$Y_0 = \bar{I}_2I_0 + I_2I_0 \tag{8.44}$$

There are 8 distinct product terms considering the product term *(= 0) as AND gate with all the outputs intact so as to have its output always '0'. Table 8.23 shows the PLA program table and Table 8.24 shows the PAL program table. The size of the required PLA = 3 inputs × 6 outputs with 8 product terms. The size of the required PAL = 3 inputs × 6 outputs with minimum 2 AND gates per output or gate.

Table 8.23 PLA program table for Example 8.11

	Product terms	Inputs			Outputs				
		I_2	I_1	I_0	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
I_2I_1	1	1	1	–	1	–	–	–	–	–
$I_2\bar{I}_1$	2	1	0	–	–	1	–	–	–	–
I_2I_0	3	1	–	1	–	1	–	–	–	–
$I_2\bar{I}_1I_0$	4	1	0	1	–	–	1	–	–	1
$\bar{I}_2I_1I_0$	5	0	1	1	–	–	1	–	–	–
$I_1\bar{I}_0$	6	–	1	0	–	–	–	1	–	–
$I_2\bar{I}_2I_1\bar{I}_1I_0\bar{I}_0$	7	–	–	–	–	–	–	–	1	–
$I_0\bar{I}_2$	8	0	–	1	–	–	–	–	–	1
					T	T	T	T	T	T

Table 8.24 PAL program table for Example 8.11

	Product terms	Inputs			Outputs
		I_2	I_1	I_0	
I_2I_1	1	1	1	–	$Y_5 = I_2I_1$
*	2	–	–	–	
$I_2\bar{I}_1$	3	1	0	–	$Y_4 = I_2\bar{I}_1 + I_2I_0$
I_2I_0	4	1	–	1	
$I_2\bar{I}_1I_0$	5	1	0	1	$Y_3 = I_2\bar{I}_1I_0 + \bar{I}_2I_1I_0$
$\bar{I}_2I_1I_0$	6	0	1	1	

Contd.

Contd.

	Product terms	Inputs			Outputs
		I_2	I_1	I_0	
	7	–	1	0	$Y_2 = I_1\bar{I}_0$
$I_1\bar{I}_0$	8	–	–	–	
*	9	–	–	–	$Y_1 = 0$
*	10	–	–	–	
*	11	0	–	1	$Y_0 = \bar{I}_2I_0 + I_2I_0$
$\bar{I}_2I_0$	12	1	–	1	
I_2I_0					

* Product terms with all inputs intact.

Example 8.12 Design a sequence detector to detect the sequence 1101, using a suitable flip-flop. Give state diagram and state table.

Solution

Let us assume A to be the initial (reset) state, X as input and Z as output.

The state diagram is shown in Fig. 8.58 by considering the sequence to be overlapping, i.e.

(Input) X .....1101101011101 10.....

(Output) Y .....0001001000001.....

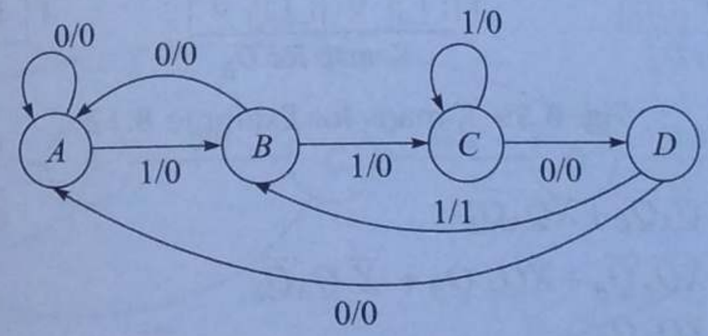


Fig. 8.58 State diagram for Example 8.12

In the above state diagram, state B indicates occurrence of one 1. State C indicates occurrence of two 1s. Hence, the circuit remains in state C till it receives a 0. State D indicates that 110 is received, hence it returns to B with output = 1 if *i/p* bit received is 1, else it returns to initial condition with output = 0.

The state table is given in Table 8.25.

Table 8.25 State table for Example 8.12

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	B	0	1

Let the state assignments be

$A = 0\,0, \quad B = 0\,1, \quad C = 1\,0, \quad D = 1\,1$

We need two flip-flops. Let us have two D flip-flops with D_A and D_B as inputs and Q_A and Q_B as outputs.

Table 8.26 Excitation table for Example 8.12

Present state		*I/P*	*Next state*		*FF inputs*		*Output*
Q_A	Q_B	X	Q_A^+	Q_B^+	D_A	D_B	Z
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	0	0	0	0
0	1	1	1	0	1	0	0
1	0	0	1	1	1	1	0
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	0
1	1	1	0	1	0	1	1

The K-maps for D_A, D_B and Z are shown in Fig. 8.59 and their equations are given by equations 8.46 to 8.48.

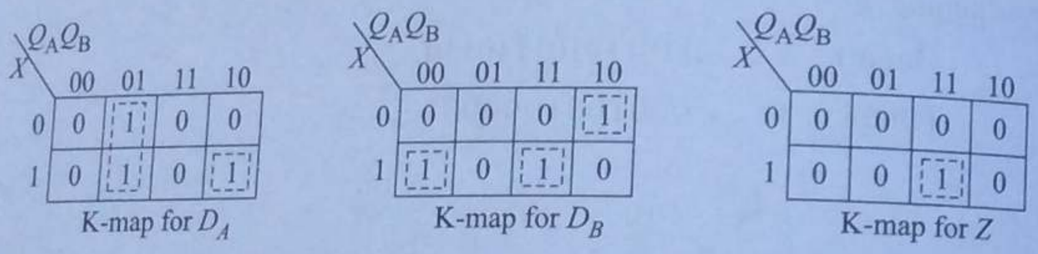


Fig. 8.59 K-maps for Example 8.12

$$D_A = \overline{Q}_A Q_B + X Q_A \overline{Q}_B \tag{8.46}$$

$$D_B = X\overline{Q}_A \overline{Q}_B + X Q_A Q_B + \overline{X}\, Q_A \overline{Q}_B \tag{8.47}$$

$$Z = X Q_A Q_B \tag{8.48}$$

From the equations, there are 5 distinct product terms. Hence, we need a PLA with 3 inputs, 3 outputs and 5 product terms. Also, we need two D flip-flops connected at the outputs of PLA, as given in Fig. 8.60.

Table 8.27 PLA program table for Example 8.12

	Product terms	*AND Inputs*			*Outputs*		
		X	Q_A	Q_B	D_A	D_B	Z
$\overline{Q}_A Q_B$	1	–	0	1	1	–	–
$XQ_A\overline{Q}_B$	2	1	1	0	1	–	–
$X\overline{Q}_A\overline{Q}_B$	3	1	0	0	–	1	–
XQ_AQ_B	4	1	1	1	–	1	1
$\overline{X}Q_A\overline{Q}_B$	5	0	1	0	–	1	–
					T	*T*	*T*

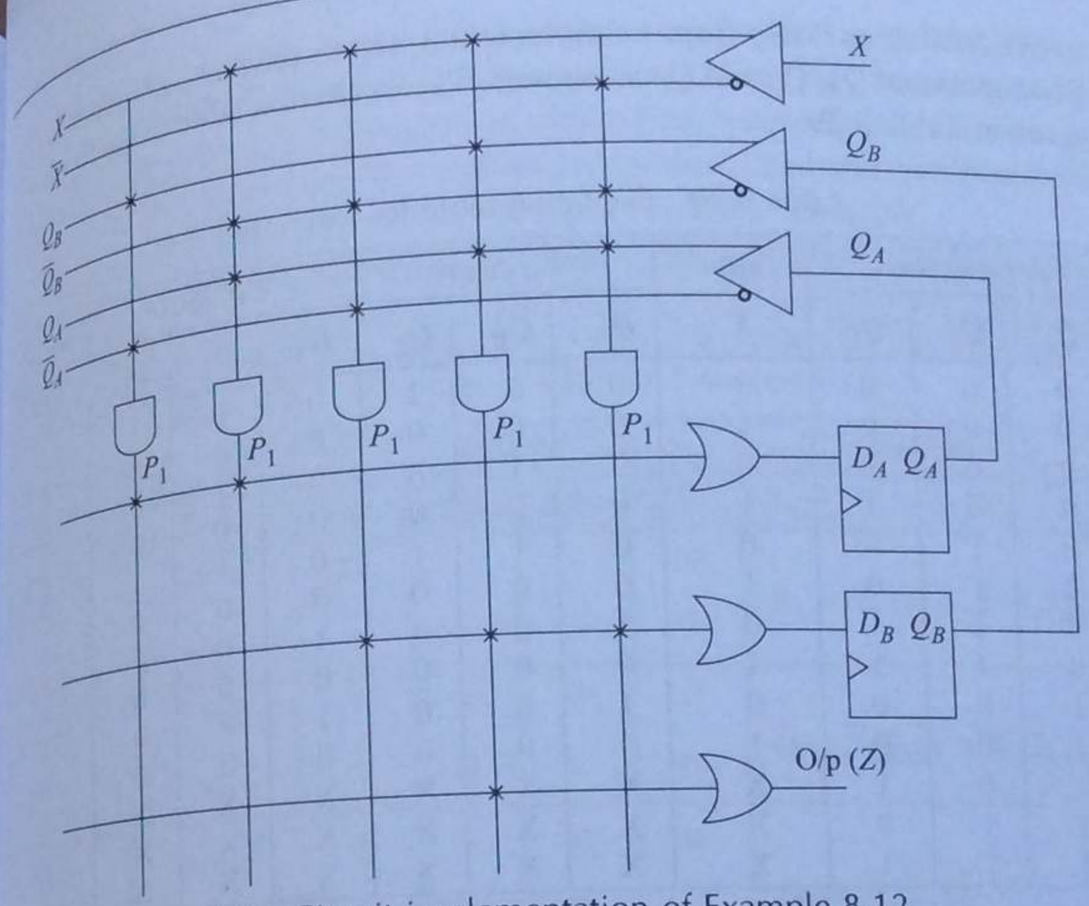


Fig. 8.60 Circuit implementation of Example 8.12

Example 8.13 Implement the state diagram shown in Fig. 8.61 using PAL. Give the size of PAL required.

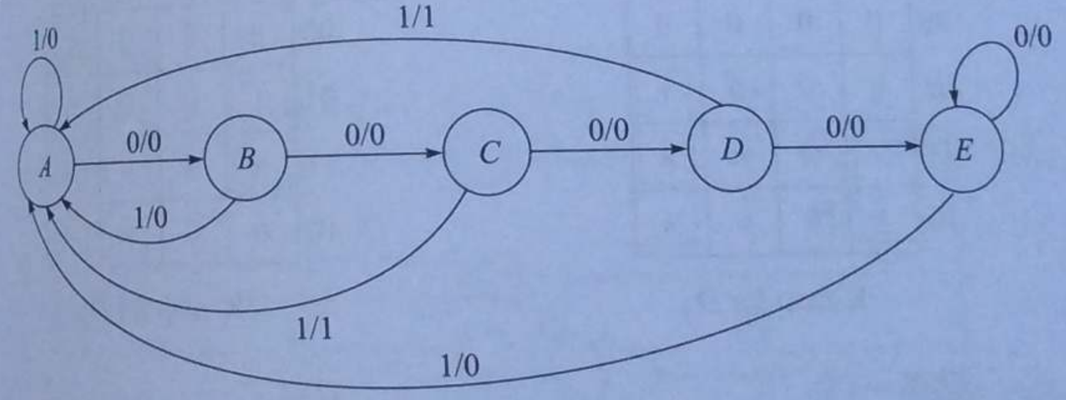


Fig. 8.61 State diagram

Solution

The state table for the state diagram of Fig. 8.61 is given in Table 8.28.

Table 8.28 State table for Example 8.13

Present State	*Next State*		*Output*	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
A	B	A	0	0
B	C	A	0	0
C	D	A	0	1
D	E	A	0	1
E	E	A	0	0

Let us have the state assignments as

$A = 000, B = 001, C = 010, D = 011, E = 100$

We need three D flip-flops to implement it. Let us have D_A, D_B and D_C as flip-flop inputs and Q_A, Q_B and Q_C as outputs. The excitation table of the flip-flops is given in Table 8.29.

Table 8.29 Excitation table for Example 8.13

Present State			*Input*	*Next State*			*FF i/ps*			*Output*
Q_A	Q_B	Q_C	X	Q_A^+	Q_B^+	Q_C^+	D_A	D_B	D_C	Z
0	0	0	0	0	0	1	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0
0	1	0	0	0	1	1	0	1	1	0
0	1	0	1	0	0	0	0	0	0	1
0	1	1	0	1	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	0	0	0
1	0	0	1	0	0	0	0	0	0	0
1	0	1	X	X	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X

The K-maps for D_A, D_B, D_C and Z are shown in Fig. 8.62.

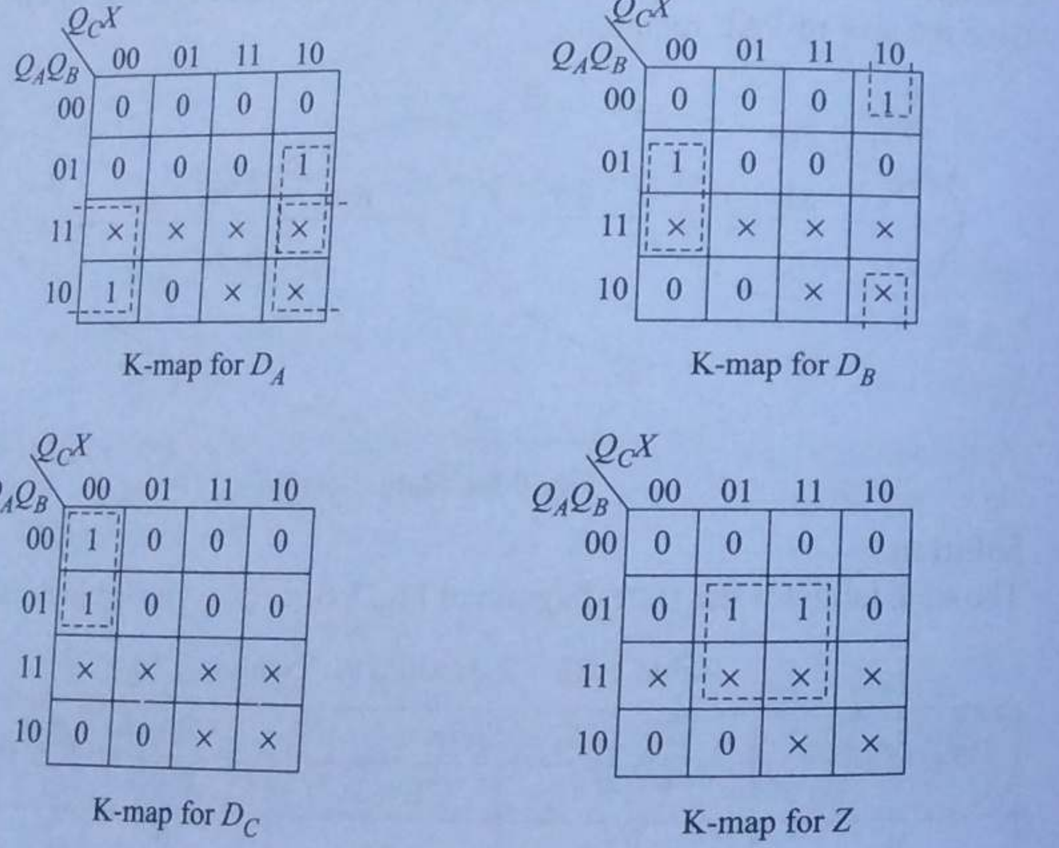


Fig. 8.62 K-maps for D_A, D_B, D_C, Z

$$D_A = \bar{X}Q_A + \bar{X}Q_BQ_C \quad (8.49)$$

$$D_B = Q_B\bar{Q}_C\bar{X} + \bar{X}\bar{Q}_BQ_C \quad (8.50)$$

$$D_C = \bar{X}\bar{Q}_A\bar{Q}_C \quad (8.51)$$

$$Z = XQ_B \quad (8.52)$$

The size of required PAL is 4 inputs, 4 outputs with minimum 2 product terms (AND gates) per output OR gate. Since we need three D flip-flops, instead of using standard I/O PAL, we can use a registered output PAL whose outputs can be used as feedbacks to the input side. We need three registered and one combinational output. Hence, we select a PAL with both types of outputs available.

Table 8.30 shows the PAL program table and Fig. 8.63 shows the PAL logic diagram.

Table 8.30 PAL program table for Example 8.13

	Product terms	*Inputs*				*Outputs*
		Q_A	Q_B	Q_C	X	
$\bar{X}Q_A$	1	1	–	–	0	$D_A = \bar{X}Q_A + \bar{X}A_BQ_C$
$\bar{X}Q_BQ_C$	2	–	1	1	0	
$\bar{X}Q_B\bar{Q}_C$	3	–	1	0	0	$D_B = \bar{X}Q_B\bar{Q}_C + \bar{X}\bar{Q}_BQ_C$
$\bar{X}\bar{Q}_BQ_C$	4	–	0	1	0	
$\bar{X}\bar{Q}_A\bar{Q}_C$	5	0	–	0	0	$D_C = \bar{X}\bar{Q}_A\bar{Q}_C$
-	6	–	–	–	–	
XQ_B	7	–	1	–	1	$Z = xQ_B$
-	8	–	–	–	–	

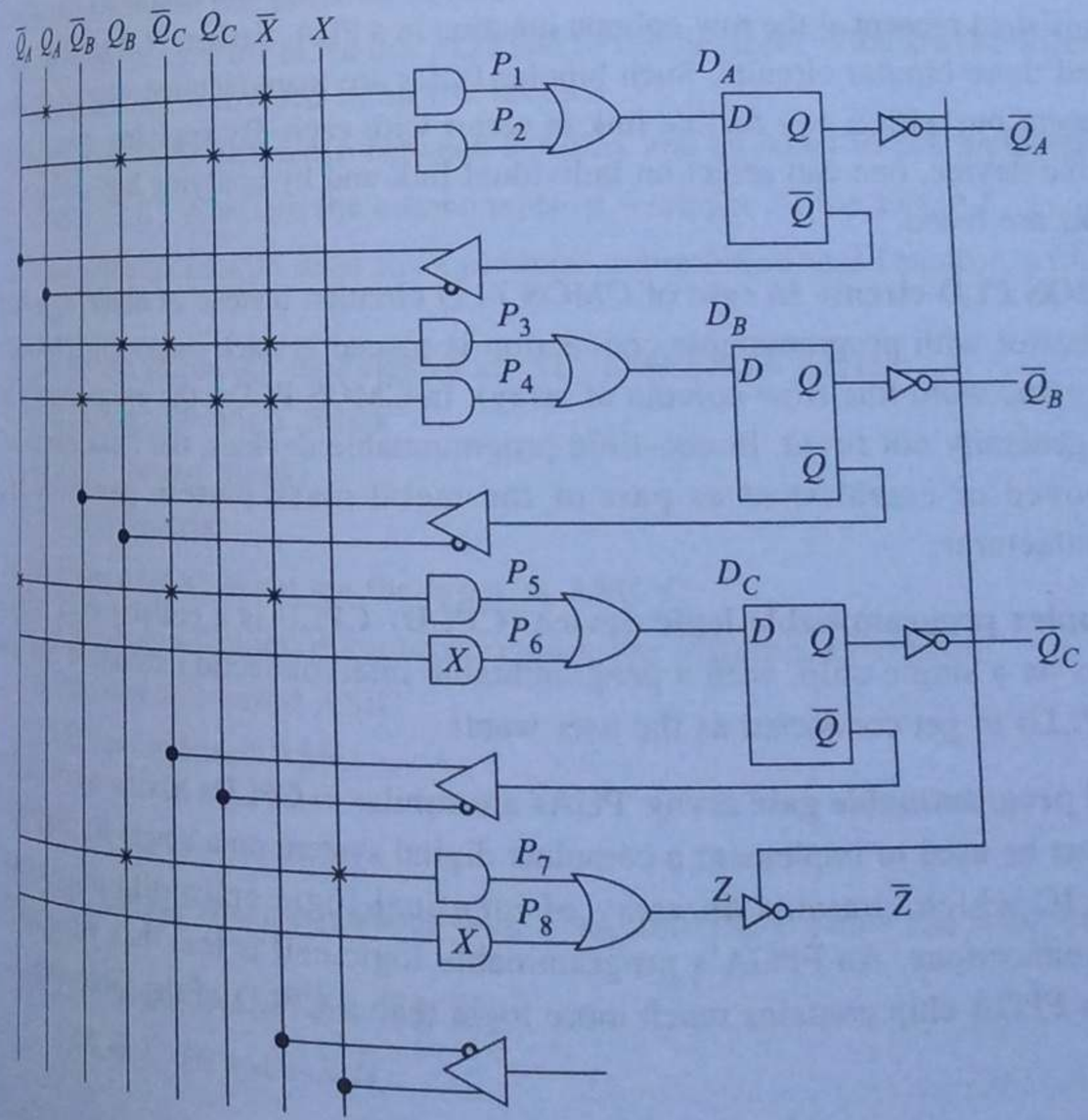


Fig. 8.63 PAL fuse map for Example 8.13

SUMMARY

In this chapter, we have looked at the different programmable logic devices like PLA, PAL and GAL and their applications in implementing combinational and sequential circuits in a digital system. In addition, we discussed some popular devices like PAL22V10 and GAL16V8 as examples.

Programmable logic device It is a group of integrated circuits that can be configured to perform logic functions. The term 'programmable' means that the function may be specified after the chip has been manufactured.

Programmable logic array (PLA) A PLA device basically contains an array of AND and OR functions whose configuration is done by the user. It typically consists of a set of fusible links which establish or remove the contact of a literal in the AND function or contact of a product term in the OR function.

Programming array logic (PAL) PAL devices are those PLDs which are very widely used and are readily available in today's market. PALs consist of an array of AND gates and an array of OR gates like PLAs. The major difference between PLA and PAL devices is that, in case of PLA, both the arrays are programmable; whereas in PAL, OR array is fixed and AND array is programmable.

Generic array logic device (GAL) It contains a programmable AND array, a fixed OR array and an output state similar to a PAL device. The output stage is referred to as the output logic macrocell (OLMC).

Bipolar PLD circuit The AND-OR arrays have bipolar elements like diode or transistors present at the row-column junction in a PLA. Earlier, PLA and PAL devices used these bipolar circuits. Such bipolar PLDs are manufactured with all the diodes present but with a tiny fusible link in series with each. By applying special patterns to the device, one can select an individual link and by applying high voltage, these links are fused.

CMOS PLD circuit In case of CMOS PLD circuits, instead of diode, an *n*-channel transistor with programmable connection is placed at each intersection between an input and word line (row-column of array). In CMOS PLDs, the programmable links are generally not fused. In non-field programmable devices, the links connection is removed or established as part of the metal mask pattern generated by the manufacturer.

Complex programmable logic device (CPLD) CPLD is a collection of individual PLDs on a single chip, with a programmable interconnection structure that allows the PLDs to get connected as the user wants.

Field programmable gate array PGAs are similar to CPLDs having more flexibility and can be used to implement a complete digital system on a single IC chip. A PGA is an IC which contains an array of identical logic cells with programmable interconnections. An FPGA's programmable logic cell is less than a typical PLD, but an FPGA chip contains much more logic than a CPLD of the same size.

EXERCISE

Review Questions

1. Explain the need and advantages of using programmable logic devices in digital system design.
2. Describe the differences between the following:
   (a) PLA and PAL
   (b) Registered output PAL and GAL
   Use logic diagrams for explanation.
3. Explain how to use a PAL having only five product terms to realize a function having six product terms.
4. Discuss the output arrangements of standard I/O, programmable I/O, and registered I/O PAL.
5. With a suitable diagram, explain the structure of Output Logic Macrocell in a GAL device (e.g. GAL 16V8).
6. What is the difference between a simple programmable logic device and a complex programmable logic device?
7. With the help of a neat block diagram of XC 9500 CPLD, explain the roles of the following modules in brief:
   (a) Switch matrix
   (b) Function block
8. With the help of a neat diagram, explain the features of XC 9500 CPLD macrocell within the function block.
9. What is the need of I/O block in CPLD and FPGA devices? What are the different typical controls provided in an I/O block?
10. Discuss the basic difference between a Xilinx and an Actel FPGA architecture.
11. What is CLB? What are the components of a simple Xilinx FPGA?
12. What is the technique used to implement a combinational function within a CLB?
13. Explain the following modules of an XC 4000 FPGA device:
   (a) Configurable logic block
   (b) Input-Output block
   (c) Switch matrix
14. What are ASICs? What are the types of ASICs?
15. With the help of block diagrams, explain the difference between:
   (a) Standard cell-based ASIC
   (b) Gate array-based ASIC

Problems

1. Realize the following equations with a suitable PLA. Draw the logic diagram using PLA.
   (a) $f_1(A, B, C, D) = A\overline{B}D + \overline{A}B\overline{D}$
   $f_2(A, B, C, D) = A + B\overline{D}$

(b) $W(A, B, C) = \Sigma m(0, 1, 4)$
$X(A, B, C) = \Sigma m(0, 3, 4, 7)$
$Y(A, B, C) = \Sigma m(1, 2, 6)$
$Z(A, B, C) = \Sigma m(2, 3, 6, 7)$

2. Realize the following functions using PAL and draw the logic diagram using PAL.
(a) $F(A, B, C) = \Sigma m(3, 4, 5\ 7)$
$G(A, B, C) = \Sigma m(1, 3, 5, 6, 7)$
$H(A, B, C) = \Sigma m(1, 4, 5)$
(b) $Y_1(P, Q, R, S) = \Sigma m(3, 5, 6, 7, 8, 11, 13, 14, 15)$
$Y_2(P, Q, R, S) = \Sigma m(0, 1, 5, 6, 8, 9, 11, 13, 14)$

3. Design and implement the following multiple output combinational circuits using PAL and PLA devices:
(a) Full-subtractor
(b) 2-bit comparator with less than, greater than, and equality output
(c) Excess-3 to BCD code converter
(d) 4-bit Grey-to-Binary code converter

4. Design a combinational circuit for an 8-segment display driver shown in Fig. 8.64. It has 4 inputs, a, b, c, d and 8 outputs, Y_1 to Y_8.
This should display the decimal equivalent of a 4-bit binary number in 1's complement form. For example, the codes are shown below:

0 0 0 0 (0)	1 0 0 0 (–7)
0 0 0 1 (1)	1 0 0 1 (–6)
0 0 1 0 (2)	1 0 1 0 (–5)
.	.
.	.
.	.
0 1 1 1 (7)	1 1 1 1 (0)

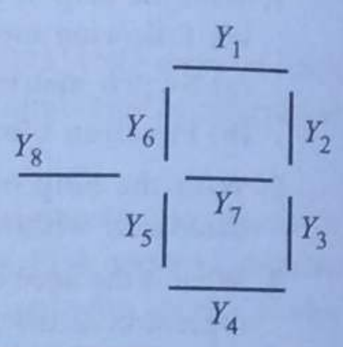


Fig. 8.64 8-segment display

Segment Y_8 should indicate the sign (it is lit for –1 to –7 only)
(a) Write the truth table for the circuit.
(b) Show the maps and the equations suitable for implementation using PLA.
(c) Implement these functions using PLA with minimum product terms. Draw the logic diagram.

5. Design a 4-bit synchronous binary counter using
(a) D flip-flops
(b) JK flip-flops
Implement them using a suitable PLA. Draw the logic diagrams for (a) and (b).

6. Design and implement a BCD counter using a PAL with 16 inputs and 4 registered outputs. Specify the number of product terms per output.
(a) Write the PAL program table.
(b) Draw the logic diagram.

7. Design Mealy circuit whose output is 1 if and only if the input has been 1 for at least three consecutive cycles.
(a) Draw the state diagram.
(b) Write the state table.
Implement the circuit using a suitable PAL device with standard I/O configuration and D flip-flops.

8. Design a synchronous counter that goes through the sequence
$1 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 4 \rightarrow 2 \rightarrow 0 \rightarrow 6$
(a) Implement it using JK flip-flops and a suitable PLA.
(b) Implement it using a suitable PAL device with registered output.

Chapter

9

A/D AND D/A CONVERTERS

Chapter Outline

- Basic concepts of analog-to-digital conversion and vice versa
- Types of ADC and DAC
- Examples of ADC and DAC

9.1 INTRODUCTION

Real world processes produce analog signals that carry information. An analog signal is defined over a continuous period of time and its amplitude may assume a continuous range of values. It is difficult to store, compare, calculate, and manipulate this information with good accuracy using purely analog technology.

Digital signal processing technology has a number of advantages over analog signal processing. Therefore in most of the applications, digital systems are preferred over analog systems. But it is found that most of the real data available is in analog form, for example, voice signal, output of the sensors, etc. Therefore it is necessary to convert analog signals into digital signals. An analog-to-digital converter (ADC) is used for this purpose. After processing these digital data by a digital system, it is given to a control device or transmitted through an analog channel or measured by an analog equipment. And hence, the output of a digital system should be converted into an analog signal. A digital-to-analog converter (DAC) is used for this purpose. The block diagram of a digital system with analog input and output is shown in Fig. 9.1.

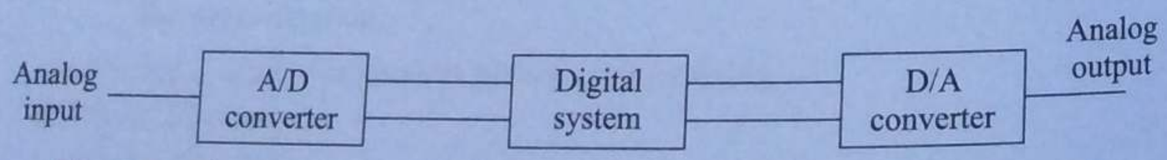


Fig. 9.1 Block diagram of a digital system with analog input and output

This chapter discusses the most common A/D and D/A conversion techniques.

9.2 BASIC PRINCIPLE OF DAC

The digital-to-analog (D/A) conversion involves translating digital information into its equivalent analog information. After processing the digital data by a digital system, it is given to an analog control device to drive the cursor arms of a plotter or transmitted through an analog channel or measured by an analog equipment. Hence the output of a digital system should be converted into analog signal. The DAC is used to convert digital signals into analog signals.

Figure 9.2 shows a block diagram of a 3-bit DAC. It has three input lines, D_2, D_1, and D_0, and one output line which provides the analog signal. A 3-bit digital signal has eight combinations, from 000 to 111. The output analog voltage level is a function of the input combinations of 0 and 1; it is minimum for 000 and maximum for 111. Let us assume the output analog signal is in the range of 0 to 1V and the input combinations are 000 to 111. Its equivalent analog is given in Table 9.1 and the relation between the digital signal input and the analog signal output is shown in Fig. 9.3.

Table 9.1 Digital input and its equivalent analog

Digital signal	*Analog signal*
000	0 V
001	1/8 V
010	2/8 V
011	3/8 V
100	4/8 V
101	5/8 V
110	6/8 V
111	7/8 V

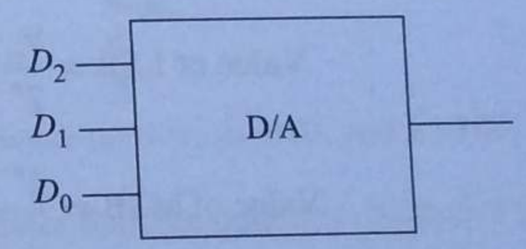


Fig. 9.2 Block diagram of 3-bit DAC

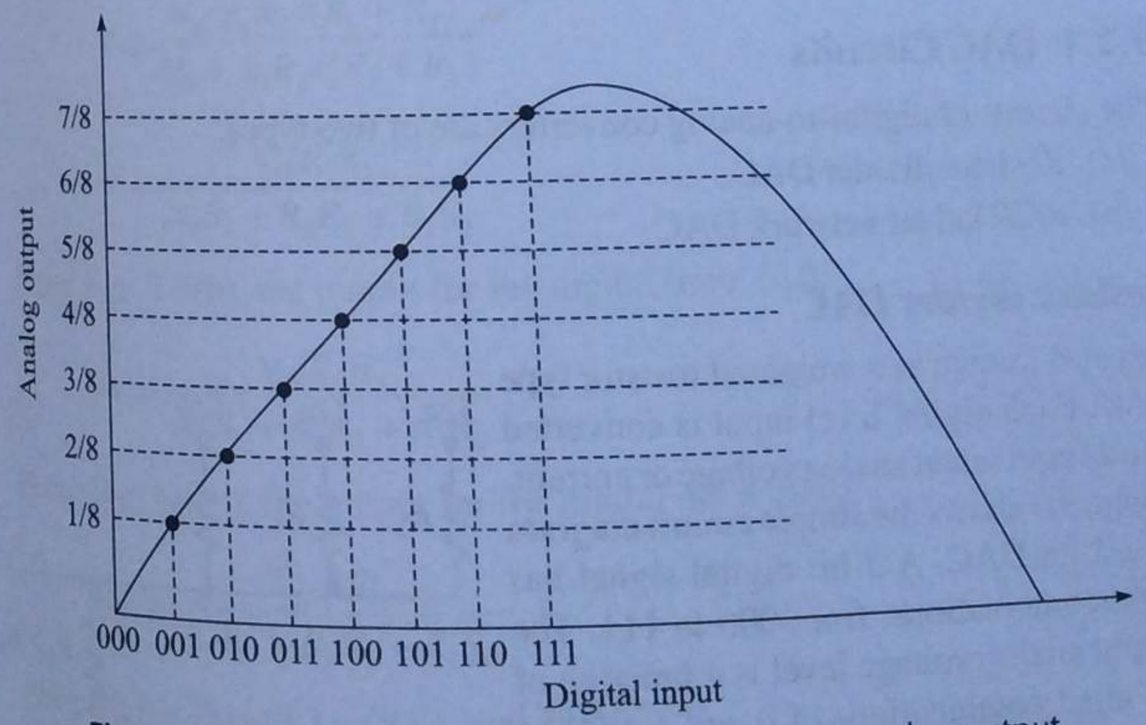


Fig. 9.3 The relation between digital input and analog output

From the above table and Fig. 9.3,

1. The value of LSB is $\frac{V_m}{2^n}$
2. The value of second LSB is $\frac{2^1 \cdot V_m}{2^n}$
3. The value of *N*th LSB is $\frac{2^{n-1} \cdot V_m}{2^n}$
4. Full scale output = V_m – Value of LSB

where n is the number of bits of a digital signal and V_m is the maximum amplitude. The output analog voltage V_A is

$$V_A = \frac{V_m[V_{D0}2^0 + V_{D1}2^1 + V_{D2}2^2 + \ldots + V_{Dn-1}2^{n-1}]}{2^n}$$

where V_{D0}, V_{D1}, V_{D2}, ..., V_{Dn-1} are the digital inputs of n-bit DAC.

Example 9.1 Calculate the values of the LSB, MSB, and the full-scale output for an 8-bit DAC for 0 to 10 V analog output range.

Solution

Given data: $n = 8$, $V_m = 10$ V

$$\text{Value of LSB} = \frac{V_m}{2^n} = \frac{10}{2^8} = \frac{10}{256} = 0.039 \text{ V}$$

$$\text{Value of MSB} = \frac{2^{n-1} \cdot V_m}{2^n} = \frac{2^{8-1} \cdot 10}{2^8} = 5 \text{ V}$$

Full-scale output = V_m – Value of LSB
= 10 – 0.039 = 9.961 V

9.2.1 DAC Circuits

The circuits of digital-to-analog converters are of two types:

(a) Resistor divider DAC
(b) *R*/2*R* ladder network DAC

Resistor divider DAC

It is also known as a weighted resistor type DAC. Each digital level input is converted into an equivalent analog voltage or current. Figure 9.4 shows the simple circuit diagram of a 3-bit DAC. A 3-bit digital signal has eight combinations, from 000 to 111. The output analog voltage level is a function of the input combinations of 0 and 1 and the amplitude or reference voltage; it is minimum for 000 and maximum for 111.

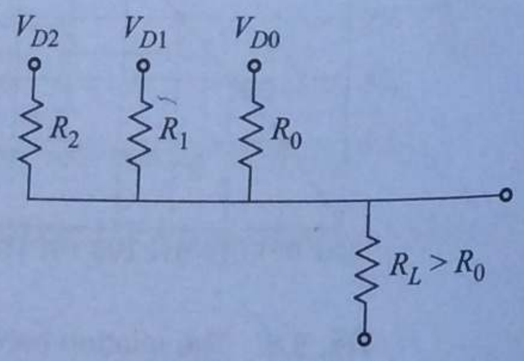


Fig. 9.4 3-bit DAC

In Fig. 9.4, V_{D2}, V_{D1}, and V_{D0} are the 3-bit digital inputs reference voltages and V_A is the analog output. Assume logic 0 corresponds to 0V and logic 1 corresponds to 7V. For the input 001, the output should be $V_m/2^n = 7/2^3 \cong 1$ V; for the input 010, the output should be $2^1 \cdot V_m/2^n = 2^1 \cdot 7/2^3 \cong 2$ V, and for the input 100, the output should be $2^2 \cdot V_m/2^n = 2^2 \cdot 7/2^3 \cong 4$ V. Let us consider that the digital input signals 001, 010, and 100 are applied to the network, as shown in Figs 9.5 (a), (b), and (c), respectively, and assume $V_{D2} = V_{D1} = V_{D0} = V_D$.

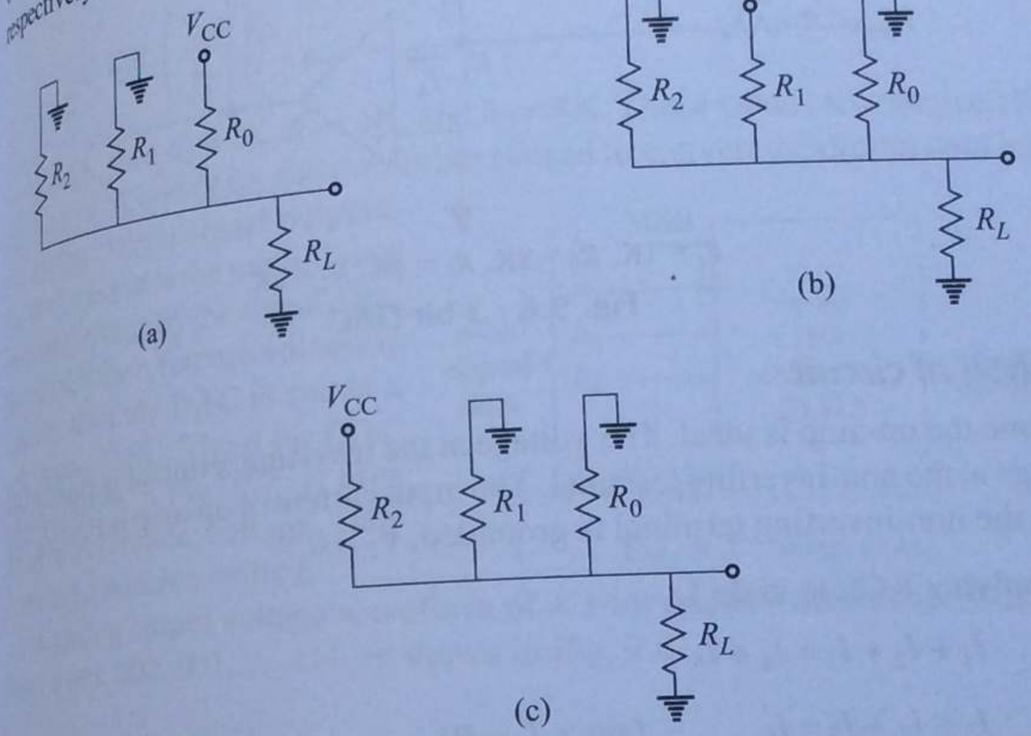


Fig. 9.5 Digital input applied to the network (a) 001, (b) 010, and (c) 100

Assume that the load resistance R_L is greater than the output resistance of the network.

Form Fig. 9.5(a), the output for the digital input 001,

$$1 \text{ V} = \frac{V_D R_1 R_2/(R_1 + R_2)}{R_0 + R_1 R_2/(R_1 + R_2)}$$

$$1 \text{ V} = \frac{V_D R_1 R_2}{R_0 R_1 + R_0 R_2 + R_1 R_2} \tag{9.1}$$

Form Fig. 9.5(b), the output for the digital input 010,

$$2 \text{ V} = \frac{V_D R_0 R_2}{R_0 R_1 + R_0 R_2 + R_1 R_2} \tag{9.2}$$

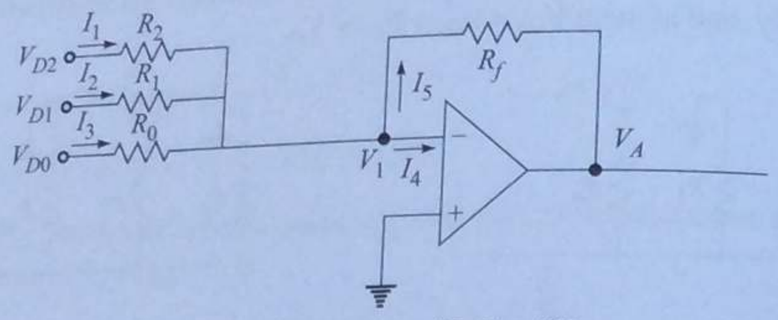
Form Fig. 9.5(c), the output for the digital input 100,

$$4 \text{ V} = \frac{V_D R_1 R_0}{R_2 R_0 + R_1 R_2 + R_1 R_0} \tag{9.3}$$

Simplifying Eqs (9.1), (9.2), and (9.3),

$R_1 = R_0/2$ and $R_2 = R_0/4$

The circuit will work as a DAC and provide more accurate result under the assumption that the load resistance is higher than the output resistance of the circuit. If it is low, the circuit cannot provide accurate result. To avoid this problem, an op-amp is used which isolates the load from the resistive network. The circuit of a DAC with op-amp is shown in Fig. 9.6.



$R_f = 1\text{K}, R_0 = 8\text{K}, R_1 = 4\text{K}, R_2 = 2\text{K}$

Fig. 9.6 3-bit DAC

Analysis of circuit

Assume the op-amp is ideal. The voltage at the inverting terminal is equal to the voltage at the non-inverting terminal. The input current of op-amp is zero ($I_4 = 0$), since the non-inverting terminal is grounded, $V_1 = 0$.

Applying KCL to node 1,

$$I_1 + I_2 + I_3 = I_4 + I_5$$

$$I_1 + I_2 + I_3 = I_5 \qquad (\text{since } I_4 = 0)$$

$$\frac{V_{D2} - V_1}{R_2} + \frac{V_{D1} - V_1}{R_1} + \frac{V_{D0} - V_1}{R_0} = \frac{V_1 - V_A}{R_f}$$

$$\frac{V_{D2}}{R_2} + \frac{V_{D1}}{R_1} + \frac{V_{D0}}{R_0} = -\frac{V_A}{R_f}$$

$$V_A = R_f\left(\frac{V_{D2}}{R_2} + \frac{V_{D1}}{R_1} + \frac{V_{D0}}{R_0}\right) \qquad (\text{Considering the magnitude only})$$

Assume $V_{D2} = V_{D1} = V_{D0} = V_D = 7\text{V}$

For the input 001, the output $V_A = \frac{V_m}{2^n} = \frac{7}{2^3} = \frac{7}{8}$

$$\frac{7}{8}V = R_f\left(\frac{V_{D0}}{R_0}\right) = \frac{7R_f}{R_0} \qquad (9.4)$$

For the input 010, the output,

$$V_A = \frac{2^2 \cdot V_m}{2^n} = \frac{2^1 \cdot 7}{2^3} = \frac{14}{8} = \frac{7}{4}$$

$$\frac{7}{4}V = R_f\left(\frac{V_{D1}}{R_1}\right) = \frac{7R_f}{R_1} \qquad (9.5)$$

For the input 100, the output,

$$V_A = \frac{2^2 \cdot V_m}{2^n} = \frac{2^1 \cdot 7}{2^3} = \frac{28}{8} = \frac{7}{2}$$

$$\frac{7}{2}V = R_f\left(\frac{V_{D2}}{R_2}\right) = \frac{7R_f}{R_2} \qquad (9.6)$$

If $R_f = 1\text{K}$, then $R_2 = 2\text{K}$, $R_1 = 4\text{K}$, and $R_0 = 8\text{K}$. These values are obtained from Eqs (9.4), (9.5), and (9.6), and the circuit is used to convert the digital data inputs into analog current signal outputs. The DAC output is the multiplying reference voltage by the signal data. Therefore if the reference voltage is variable, then the DAC is called a multiplying DAC. The n-bit DAC circuit shown in Fig. 9.7. It uses negative reference voltage, thus producing a positive voltage.

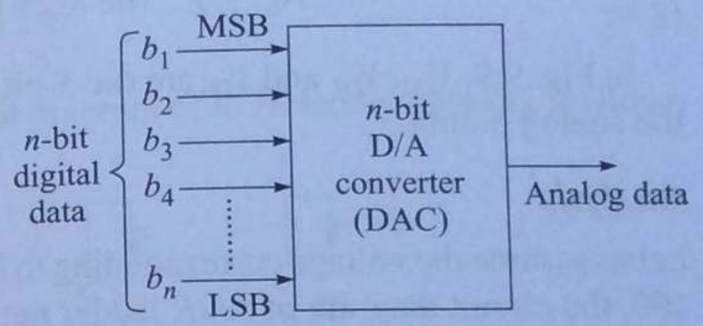


Fig. 9.7 n-bit DAC

The analog output voltage waveform of a 3-bit resistor divider DAC for the digital word, 000, 001,.....111, is shown in Fig. 9.8.

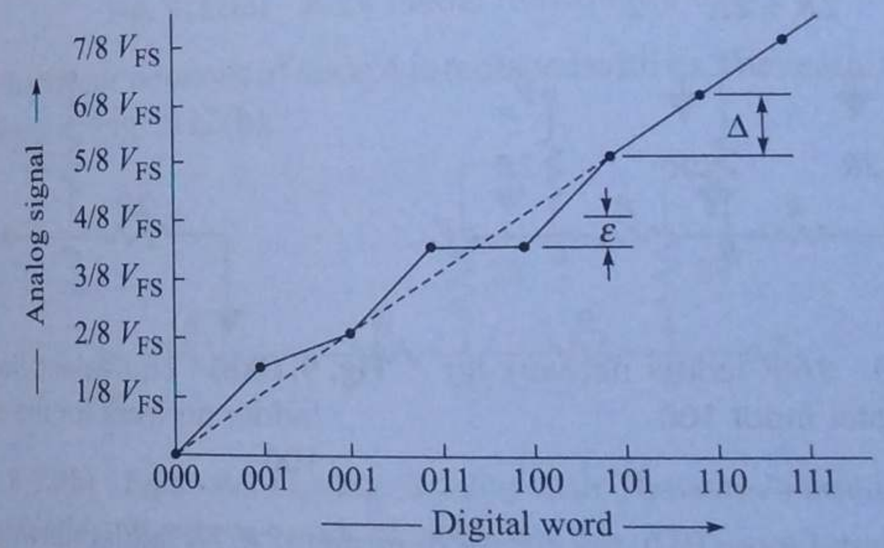


Fig. 9.8 Analog output voltage waveform for a 3-bit resistor divider DAC

Higher the value of n, finer the resolution of conversion and less is the step size of staircase. The accuracy and stability of DAC are based on the accuracy of the resistors and their temperature dependence.

A weighted DAC requires a wide range of resistor values. As the length of binary inputs increases, the range of resistor values needed also increases. For an 8-bit DAC, the larger values of resistor are 128 times the value of the smallest resistor.

R/2R ladder network DAC

The major drawback of a resistor divider DAC is the requirement of a wide range of resistor values. In contrary, *R*-2*R* ladder DAC uses only two resistor values, i.e. *R* and 2*R*. The *R*/2*R* ladder network is shown in Fig. 9.9.

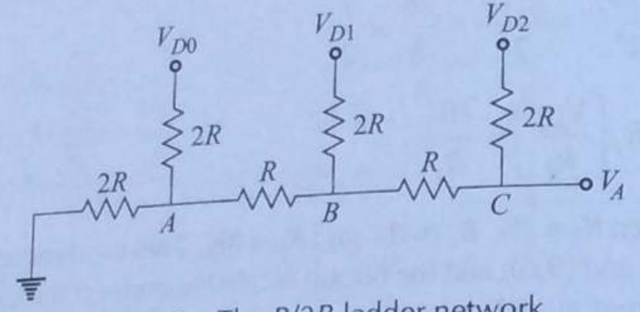


Fig. 9.9 The *R*/2*R* ladder network

In Fig. 9.9, V_{D2}, V_{D1} and V_{D0} are the 3-bit digital inputs voltage levels and V_A is the analog output.

Analysis

Let us assume the voltage corresponding to logic level 1 is V_m. For the digital input 100, the circuit diagram of *R*/2*R* ladder network is shown in Fig. 9.10(a) and its equivalent circuit is shown in Fig. 9.10(b). For the digital input signal 100, the output analog voltage is,

$$V_A = \frac{V_m \times 2R}{2R + 2R} = \frac{V_m}{2} \tag{9.7}$$

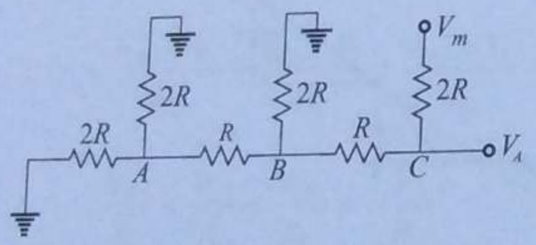


Fig. 9.10(a) R/2R ladder network for the digital input 100

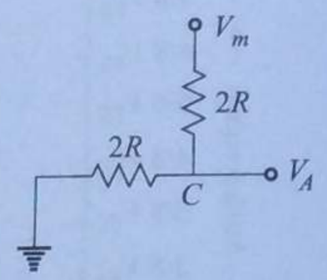


Fig. 9.10(b) Equivalent circuit of R/2R ladder network for the digital input 100

For the digital input 010, the circuit diagram of *R*/2*R* ladder network is shown in Fig. 9.11(a) and its equivalent circuit is shown in Fig. 9.11(b).

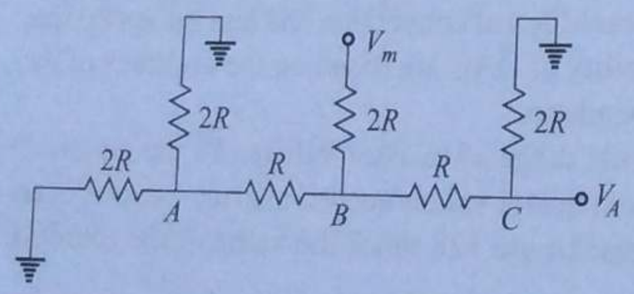


Fig. 9.11(a) R/2R ladder network for 010

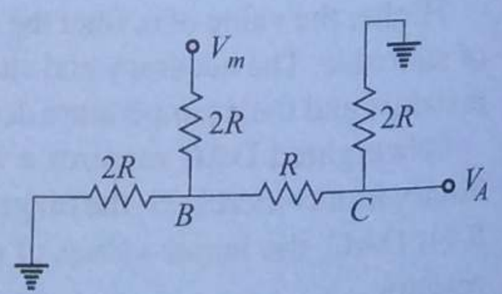


Fig. 9.11(b) Equivalent circuit of R/2R ladder network for 010

The left-hand side network of node *B* is replaced with its Thevenin's equivalent and it is shown in Fig. 9.11(c).

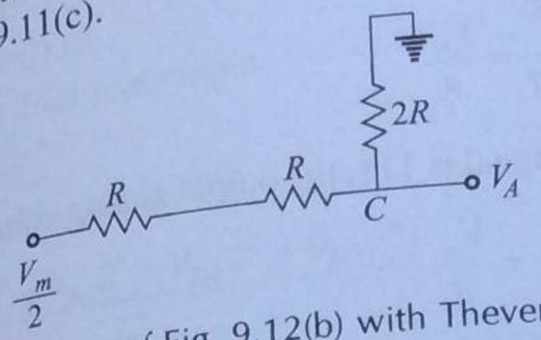


Fig. 9.11(c) Equivalent of Fig. 9.12(b) with Thevenin's equivalent of left-hand side network of node *B*

For the digital input signa l 010, the output analog voltage is,

$$V_A = \frac{V_m}{2} \times \frac{2R}{2R + 2R} = \frac{2V_m}{8} = \frac{V_m}{4} \tag{9.8}$$

For the digital input 001, the circuit diagram of *R*/2*R* ladder network is shown in Fig. 9.12(a).

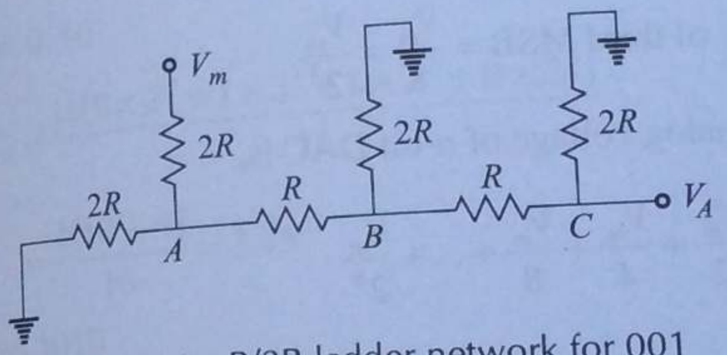


Fig. 9.12(a) R/2R ladder network for 001

The left-hand side network of node *A* is replaced with its Thevenin's equivalent and it is shown in Fig. 9.12(b).

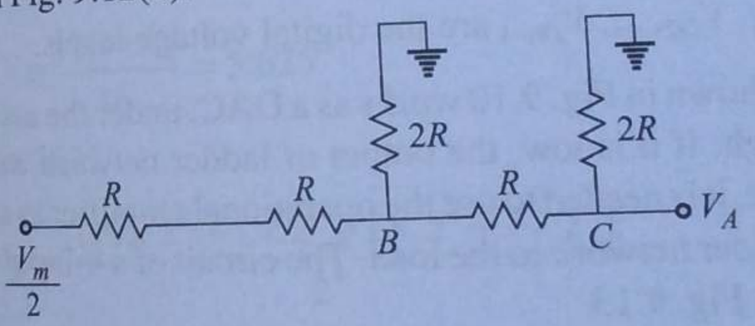


Fig. 9.12(b) Equivalent of Fig. 9.12(a) with Thevenin's equivalent of left-hand side network of node *A*

The left-hand side network of node B is replaced with its Thevenin's equivalent and it is shown in Fig. 9.12(c).

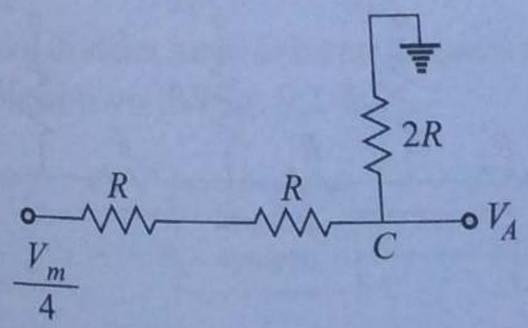


Fig. 9.12(c) Equivalent of Fig. 9.13(b) with Thevenin's equivalent of left-hand side network of node *B*

For the digital input signal 001, the output analog voltage is,

$$V_A = \frac{\frac{V_m}{4} \times 2R}{2R + 2R} = \frac{V_m}{8} \tag{9.9}$$

When the applied signal is 111, the output analog voltage will be,

$$V_m = \frac{V_m}{2} + \frac{V_m}{4} + \frac{V_m}{8} \tag{9.10}$$

From the analysis, it is observed that for a 3-bit DAC,

- The value of MSB = $\frac{V_m}{2}$
- The value of second MSB = $\frac{V_m}{4} = \frac{V_m}{2^2}$
- The value of third MSB = $\frac{V_m}{8} = \frac{V_m}{2^3}$

The output analog voltage of n-bit DAC is,

$$V_A = \frac{V_m}{2} + \frac{V_m}{4} + \frac{V_m}{8} + \ldots + \frac{V_m}{2^n} \tag{9.11}$$

$$= \frac{V_m[V_{Dn-1}\, 2^{n-1} \ldots + V_{D3}2^3 + V_{D2}2^2 + V_{D1}2^1 + V_{D0}2^0]}{2^n} \tag{9.12}$$

where V_{D0}, V_{D1}, V_{D2}, ..., V_{Dn-1} are the digital voltage levels.

The circuit shown in Fig. 9.10 works as a DAC, under the assumption that load resistance is high. If it is low, the output of ladder network will be an incorrect value. Therefore, it is needed to use the operational amplifier as a buffer to connect the output of ladder network to the load. The circuit of a 4-input DAC with an op-amp is shown in Fig. 9.13.

In Fig. 9.13, the op-amp is used in inverting mode. The amplifier acts as an inverting current to the voltage converter. The output voltage $V_A = -IR$. The analysis of the circuit is same as above.

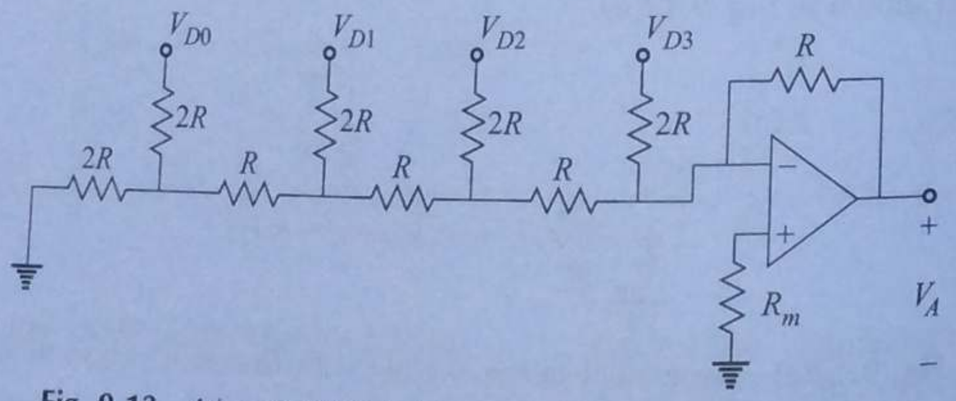


Fig. 9.13 4-inputs R/2R ladder network DAC with op-amp

Example 9.2 Find the output voltage of a 4-bit ladder, having the following digital inputs $V_m = 10$ V
(a) 1010 (b) 0110
(c) 1001 (d) Full-scale input

Solution
Assume logic 0 = 0V and logic 1 = 10V

$$V_A = \frac{V_m[V_{D0}\, 2^0 + V_{D1}2^1 + V_{D2}2^2 + V_{D3}2^3]}{2^4}$$

(a) Digital input 1010

$$V_A = \frac{10[0 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3]}{2^4}$$

$$= \frac{10[2+8]}{16} = 6.25$$

(b) Digital input 0110

$$V_A = \frac{10[0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 0 \times 2^3]}{2^4}$$

$$= \frac{10[2+4]}{16} = 3.75$$

(c) Digital input 1010

$$V_A = \frac{10[0 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 1 \times 2^3]}{2^4}$$

$$= \frac{10[1+8]}{16} = 5.625$$

(d) Digital input 1111

$$V_A = \frac{10[0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3]}{2^4}$$

$$= \frac{10[1+2+4+8]}{16} = 9.375$$

9.2.2 DAC

Either a ladder or a resistive divider network can be used as a DAC. The complete block diagram of a DAC is shown in Fig. 9.14.

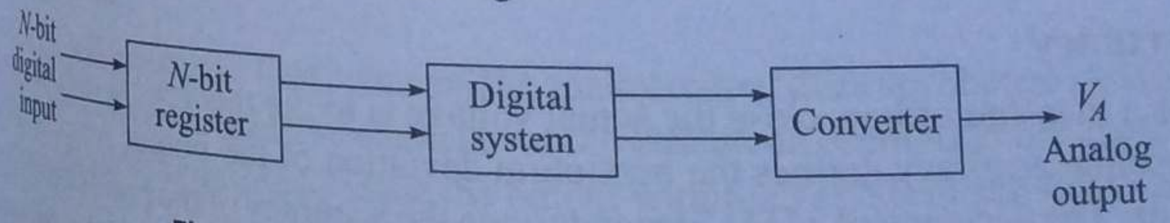


Fig 9.14 Complete block diagram of DAC

The N-bit register is used to store the digital information. The D flip-flop can be used as a basic element of a register. The number of flip-flops define the size of a register, one flip-flop stores 1-bit data. Digital data is loaded into a flip-flop with a clock pulse. The output of flip-flop is logic 1 or logic 0, values of logic 1 and logic 0 may not be same.

To provide the same level voltage for logic 1s and logic 0s, the level amplifiers are used. The level amplifier has two inputs; one is the reference voltage from an external voltage source and the other input is the output of the flip-flop. The amplifier works such that, when the input from a flip-flop is high, the output of the amplifier is V_{ref} and when the input from the flip-flop is low, the output is 0 V.

The schematic of a 4-bit DAC is shown in Fig. 9.15.

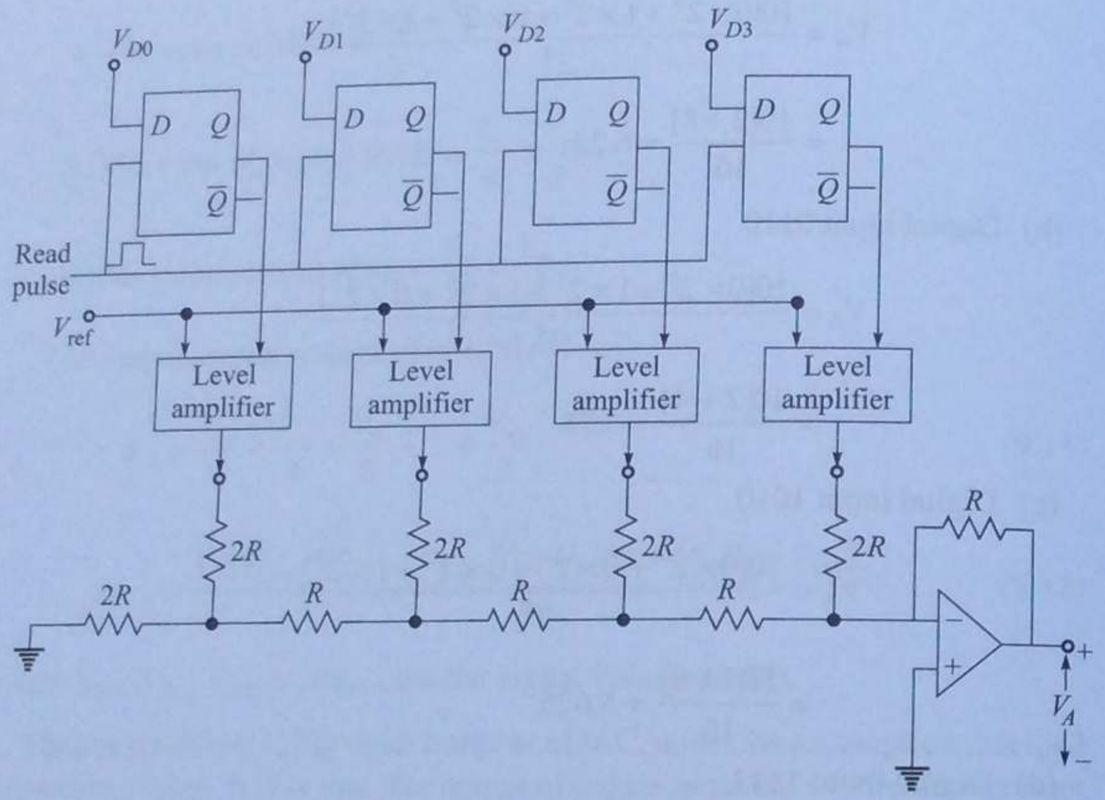


Fig. 9.15 Schematic of 4-bit DAC

9.2.3 Specifications of DAC

The specifications of a DAC which are generally specified by the designers are:

- Accuracy
- Resolution
- Linearity
- Settling time
- Temperature sensitivity

Accuracy

It is a measure of how close the actual voltage is to the theoretical output value. Absolute accuracy defines the maximum deviation of the output from the ideal value. The accuracy of a DAC depends upon the accuracy of the precision resistor used in the resistor divider or ladder network and the precision of the reference voltage. It is specified as a percentage of full-scale or maximum output voltage.

For example, suppose the theoretical output voltage is 10 V for a full-scale digital input and accuracy is ± 5 percent, it means that the output voltage of DAC for the same digital input lies between 9.5 V to 10.5 V.

The accuracy specifies the maximum error that can occur in a output voltage. For example, suppose the full-scale output voltage is 10 V and accuracy is ± 0.1 percent, then the maximum error will be 10 mV (i.e. 0.001 × 10 V).

Resolution

It defines the smallest possible change in the output analog voltage due to the change in digital input. The resolution is always equal to the weight of LSB, and it is also known as the *step size*. It is a function of the number of bits in the digital input. The resolution of an n-bit DAC using divider is $V_m/2^n$ and using ladder network, it is $V_m/2^n$ in volts.

In a 4-bit DAC using a ladder, if the full-scale voltage 16 V, then the resolution is $16/2^4 = 1$ V. Thus, the output voltage changes in a step of 1 V. If it is desired to produce 5.3 V using this converter, the actual output voltage would be 5 V. Similarly, if it is desired to produce 9.6 V, the actual output voltage would be 10 V.

It is more useful to express the resolution as a percentage. The percentage resolution is given by,

$$\text{Percentage resolution} = \frac{\text{step size}}{\text{full-scale}} \times 100 \tag{9.13}$$

The percentage resolution can be calculated as,

$$\text{Percentage resolution} = \frac{1}{\text{number of steps}} \times 100 \tag{9.14}$$

For an n-bit input,

$$\text{Percentage resolution} = \frac{1}{2^{n-1}} \times 100 \tag{9.15}$$

To get a good resolution, the number of input bits (n) of DAC should be maximum.

Linearity

In a DAC, the relation between digital input and analog output should be linear. That is, an equal increment in the numerical significance of the digital input should result in equal increments in the analog output voltage. Due to the errors in resistor values, the input-output relationship is not linear. The linearity error for a digital input is the difference between the expected voltage and the voltage obtained at the output of DAC.

Settling time

When digital inputs of a DAC change, the analog output does not change instantly. Due to the active and passive elements of the circuit, an oscillation occurs at the output. The time required to settle the output within ± ½ LSB of the final value,

after the change in the digital inputs, is referred to as the *settling time*, as shown in Fig. 9.16. It limits the frequency of digital input.

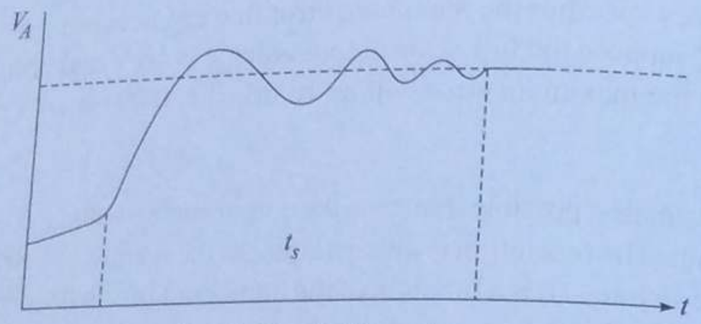


Fig. 9.16 Settling time

Temperature sensitivity

The components used in a circuit of DAC such as resistors, reference voltage source, and op-amp are sensitive to temperature. Due to the change in temperature, the characteristics of an op-amp, the values of resistor and reference voltage may change. Therefore, the analog output voltage for any fixed digital input varies with temperature. This change in values with the temperature is known as the *temperature sensitivity*. It is specified in terms of ± ppm/°C.

9.3 BASIC PRINCIPLE OF ADC

An ADC does the inverse function of a DAC. In a DAC, the possible number of digital inputs is fixed. For example, in a 4-bit DAC, there are 16 possible inputs. But in case of an ADC, the input analog voltage can have any value in a range and the digital output can have only 2^N discrete values for an N-bit ADC. The ADC process includes sampling of input analog signal and then, each sample is converted into its binary equivalent. The block diagram of an ADC is shown in Fig. 9.17.

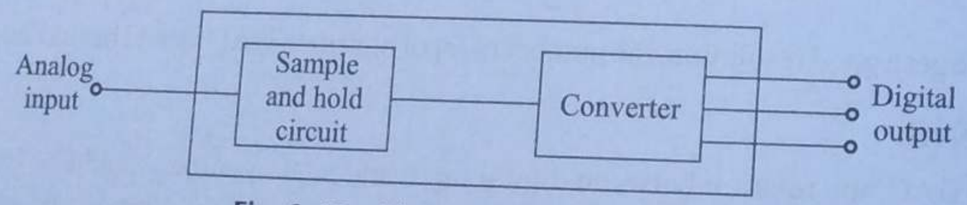


Fig. 9.17 Block diagram of ADC

Sample-and-hold circuit

The sample-and-hold circuit captures the sample after a fixed delay. The fixed delay $T = 1/f_s$ where f_s is a sampling frequency. As per the sampling theorem, the sampling frequency should be greater than or equal to twice the band limited frequency of the signal. The result of the sampling process is a series of sampling instants and the amplitude of the signal at that instant of time, i.e. output of the sample circuit is a discrete time signal of different amplitudes, as shown in Fig. 9.18. A discrete signal is one that is defined at discrete points of time only. The amplitude of a discrete signal is a continuous range of values, i.e. a discrete signal is the signal that is discrete in time but continuous in amplitude.

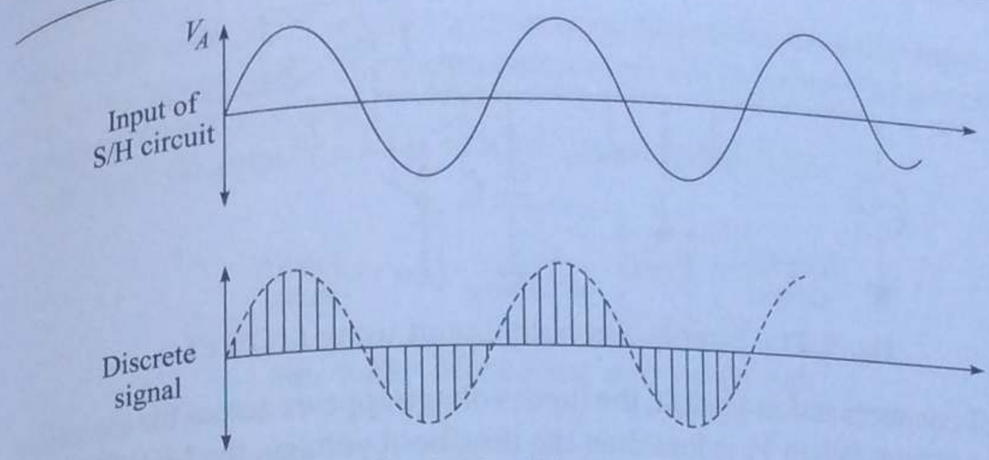


Fig. 9.18 Discrete signal

The discrete signal is converted to binary with the help of a converter. The discrete signal, as shown in Fig. 9.18, is present for a small instant and hence, the converter cannot convert it into binary. There is a need to hold the sample at least for the duration of conversion time. The sample-and-hold circuit captures the sample and holds it, as shown in Fig. 9.19.

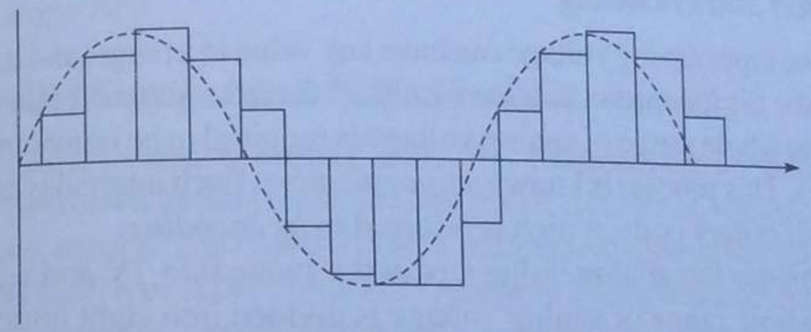


Fig 9.19 Hold discrete signal

A simple sample-and-hold circuit is shown in Fig. 9.20. In this circuit, the voltage across the capacitor follows the input signal voltage V_i when the switch S is closed. The capacitor holds the instantaneous value of the signal voltage attained just before the switch is opened. Thus for every T_s, the switch is closed for a short duration and then opened. The dc voltage across the capacitor gives the value of the signal at the instant when the switch is opened. We can say that the sample is captured after a fixed delay T_s. This dc voltage represents a sample of the signal and is converted to digital signal using ADC circuit during the hold period.

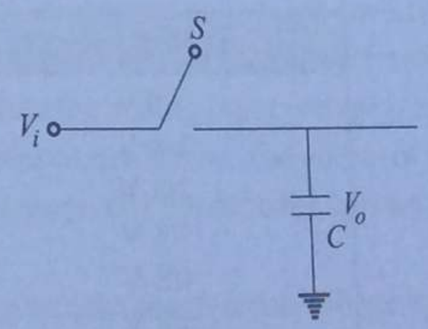


Fig. 9.20 Sample-and-hold circuit

Figure 9.21 shows a sample-and-hold circuit using MOSFET. In this circuit, an enhancement mode MOSFET is used as a switch, which is controlled by a control voltage V_s.

The op-amp is used to avoid the discharge of capacitor due to the loading effect of ADC. During the positive value of V_s (higher than the threshold voltage), the

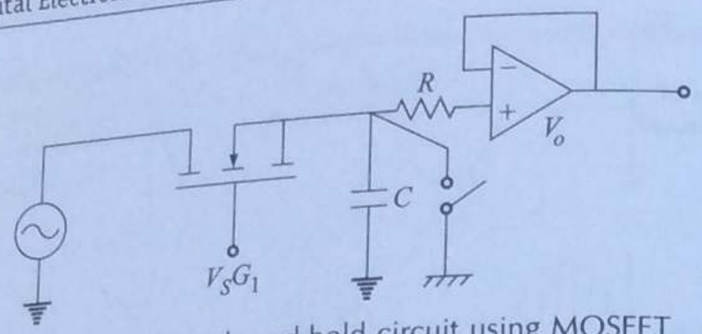


Fig. 9.21 Sample-and-hold circuit using MOSFET

MOSFET conducts and as a result, the input voltage appears across the capacitor C and at the output. When V_s is less than the threshold voltage, the MOSFET is cut off (switch open) and the voltage across C is retained because the input resistance of the op-amp is very high.

The accuracy of the circuit depends upon the holding of the charge in the capacitor, therefore a capacitor with very low leakage must be used. A capacitor with polycarbonate, polyethylene is preferred. Most of the other capacitors do not retain the stored charge for a long duration due to the polarization phenomenon.

Quantization and encoding

In an ADC, the input analog voltage can have any value in a range, and it is a function of time, but the digital output can have only 2^N discrete values for an N-bit ADC. Therefore, the whole range of analog voltage is required to be represented suitably in 2^N intervals. This process is known as *quantization*. Each interval is then assigned a unique N-bit binary code, which is referred to as *encoding*.

Let us consider the analog voltage is in the range 0 to 7V and a 3-bit digital output. The whole range of analog voltage is divided into eight intervals (2^3) of step size $S = 1/8$. Each interval is assigned a 3-bit binary value. The interval of the analog voltage and their corresponding digital values assigned are given in Table 9.2. These levels are known as the *quantization level*.

Table 9.2 Analog voltage and their corresponding digital values

Analog voltage	*Digital value*
0 V	000
1/8 V	001
2/8 V	010
3/8 V	011
4/8 V	100
5/8 V	101
6/8 V	110
7/8 V	111

During the process of conversion, the amplitude of a discrete signal is first converted to its nearest value of quantization level and then, it is encoded. The process of converting the amplitude of a discrete signal to its nearest value of quantization level is referred as *quantization*.

The process of quantization introduces error. This error is referred to as the *quantization error*. The maximum quantization error for any analog voltage is

$\pm 1/2 \times V_m/2^n = \pm V_m/2^{n+1}$ where V_m is the maximum value of analog input and n is the number bits of ADC. The quantization error will be reduced by increasing the quantization level (i.e. 2^n).

The block diagram of a 3-bit ADC is shown in Fig. 9.22.

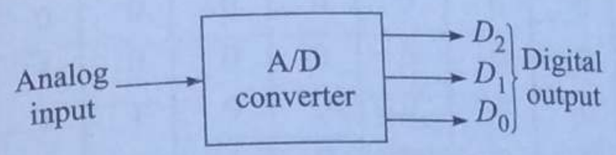


Fig. 9.22 Block diagram of 3-bit ADC

9.3.1 ADC Circuits

Different methods have been developed to convert an analog signal into its equivalent digital signal. Commonly used ADCs are:

(a) Parallel comparator ADC
(b) Counter type ADC
(c) Continuous type ADC
(d) Successive approximation ADC
(e) Dual slope ADC

9.3.2 Parallel Comparator ADC (Flash Converter)

The n-bit parallel comparator ADC uses $2^n - 1$ comparators. Each comparator compares an unknown analog input voltage with its reference voltage and provides an appropriate output. The comparison is performed simultaneously, hence it is also known as flash type ADC. Figure 9.23 shows a 3-bit parallel comparator ADC.

In a circuit diagram, the op-amp is used in open loop. The open loop op-amp acts as a comparator; it compares the input voltage with the reference voltage (fixed voltage) and the output of op-amp is $+V_{sat}$ or $-V_{sat}$. We can say that the output of comparator is high or low (logic 1 or logic 0). The reference voltages are generated using a simple resistive network and a voltage source V_R and given to the inverting terminal of op-amp. The analog input signal is connected to the non-inverting terminal of op-amp. When the analog input voltage is greater than the reference voltage, the output of comparator is high and when the analog input voltage is less than the reference voltage, then the output of comparator is low. The output of the comparator (1 or 0) is loaded into the latch and decoded by the decoder, and we get the digital output at D_2, D_1, and D_0.

For example, V_R is the full-scale voltage, from which the reference voltage $V_R/8$, $2V_R/8$, $3V_R/8$, $4V_R/8$, $5V_R/8$, $6V_R/8$, $7V_R/8$ are generated using a simple resistive network. The input analog voltage V_A is compared simultaneously with the reference voltages by using comparators. The outputs of comparators are high or low, which is a function of the input analog voltage. A 7-bit output is obtained from the comparators, which is stored in latches. The decoder circuit converts this 7-bit digital signal into a 3-bit output.

The comparator output and the digital output of a 3-bit parallel comparator ADC is given in Table 9.3. Assume $V_R = 1$ V.

Table 9.3 Comparator output and digital output of parallel comparator ADC

Analog input	*O/P comparators*							*O/P of decoder*		
V_A	C_7	C_6	C_5	C_4	C_3	C_2	C_1	D_3	D_2	D_1
$0 < V_A < 1/8$	0	0	0	0	0	0	0	0	0	0
$1/8 < V_A < 2/8$	0	0	0	0	0	0	1	0	0	1
$2/8 < V_A < 3/8$	0	0	0	0	0	1	1	0	1	0
$3/8 < V_A < 4/8$	0	0	0	0	1	1	1	0	1	1
$4/8 < V_A < 5/8$	0	0	0	1	1	1	1	1	0	0
$5/8 < V_A < 6/8$	0	0	1	1	1	1	1	1	0	1
$6/8 < V_A < 7/8$	0	1	1	1	1	1	1	1	1	0
$7/8 < V_A < V_m$	1	1	1	1	1	1	1	1	1	1

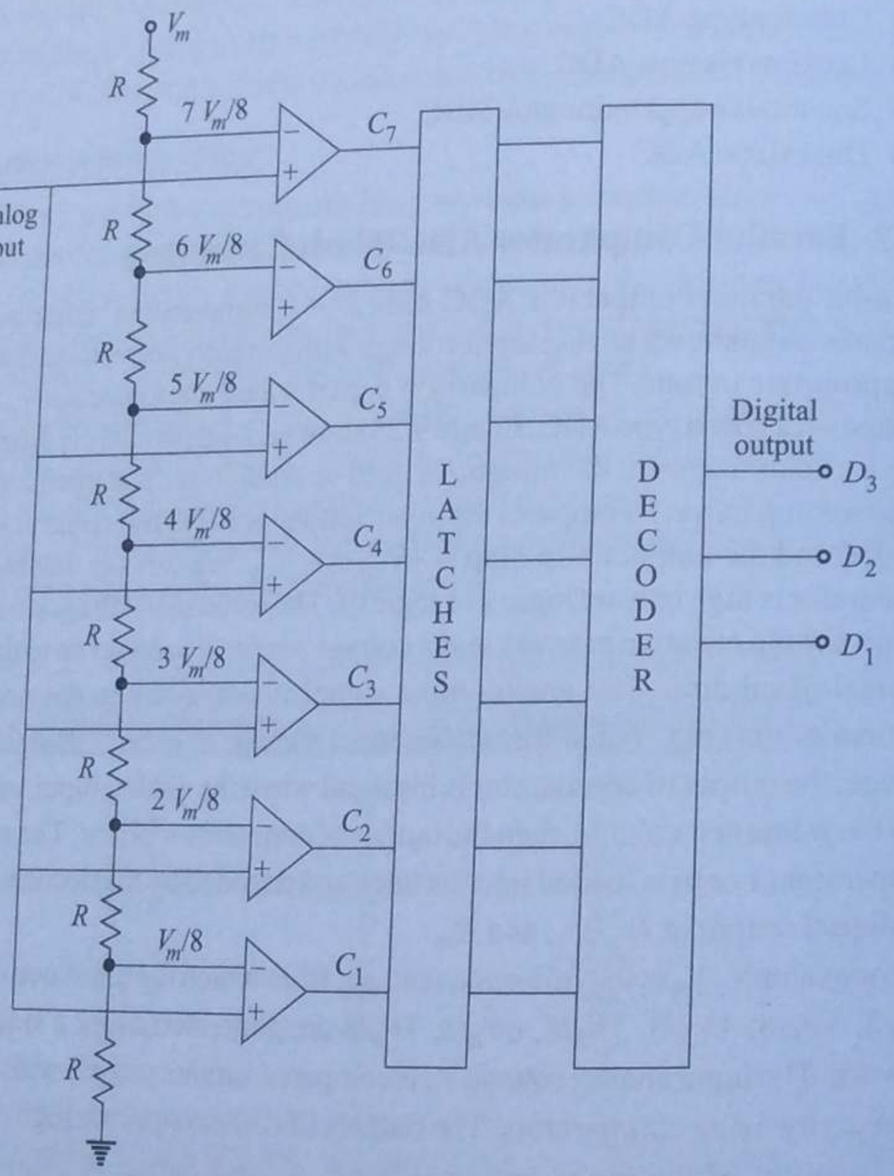


Fig. 9.23 3-bit parallel comparator ADC

The principle of parallel comparator ADC is the simplest in concept and its typical conversion time is 100 ns. But it requires a large number of comparators, which increases the cost and size of the circuit.

9.3.3 Counter Type ADC

It is a high-resolution ADC that uses a comparator with variable reference voltage. The variable reference voltage can be obtained by a sequence or binary counter and a DAC. The block diagram of a 4-bit counter type ADC is shown in Fig. 9.24.

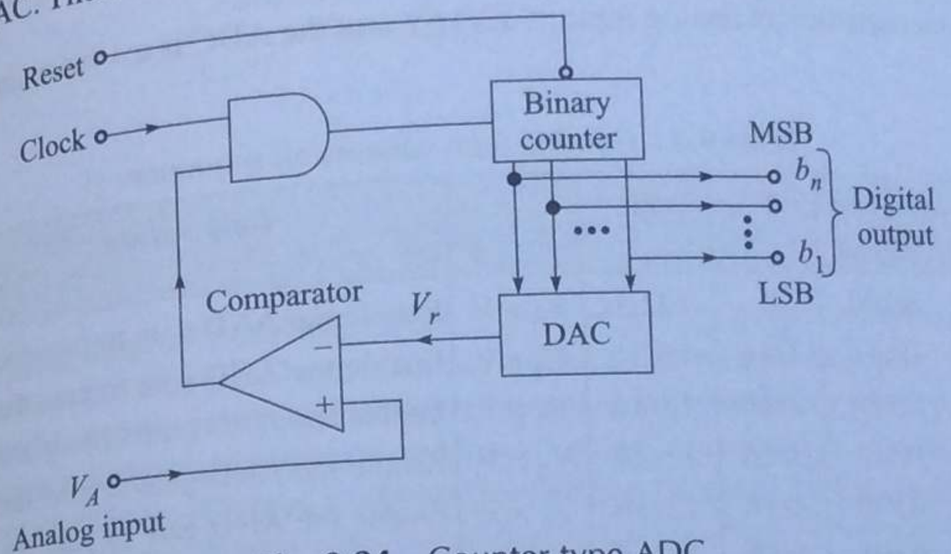


Fig. 9.24 Counter type ADC

The block diagram consists of a DAC, a comparator, an AND gate, and a binary counter. The DAC converts the digital data into analog signal, which is given as one of the input to the comparator. The op-amp is used in open loop. It acts as a comparator and compares the analog input, which has to be converted to digital with the analog output of DAC. The AND gate provides the clock to the binary counter whenever the output of comparator is high.

When the reset signal goes low, the binary counter will be set to 0, and the output of DAC is zero ($V_r = 0$). When the reset signal goes high, the clock pulse is given to the binary counter through the AND gate; it is counted by the binary counter. The DAC converts the digital output to an analog voltage and connects it to the input of comparator. The output of comparator enables the AND gate to pass the clock. The output of DAC increases with time and the analog output voltage waveform is a rising staircase, as shown in Fig. 9.25.

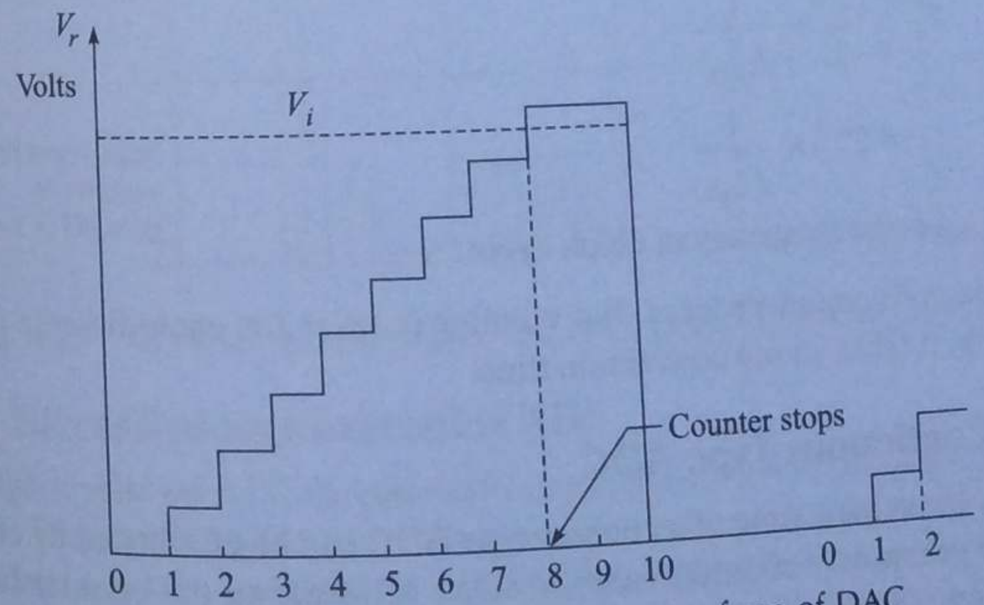


Fig. 9.25 Analog output voltage waveform of DAC

The comparator compares this analog voltage with the analog input voltage V_A. When $V_A > V_r$, the output of comparator is high, enable the AND gate to pass the clock. When $V_A \leq V_r$, the output of comparator is low, disable the AND gate to pass the clock. The binary counter stops counting, and the digital output of DAC represents the analog input voltage.

Let us consider an analog input is 13/16V and the ADC is a 4-bit counter type ADC.

Table 9.4 Counter type conversion sequence

Clock	*Digital signal*	V_r	V_A		*Description*
1	0000	0	13/16	$V_A > V_r$	Enable the AND gate to pass the clock
2	0001	1/16	13/16	$V_A > V_r$	Enable the AND gate to pass the clock
3	0010	2/16	13/16	$V_A > V_r$	Enable the AND gate to pass the clock
4	0011	3/16	13/16	$V_A > V_r$	Enable the AND gate to pass the clock
5	0100	4/16	13/16	$V_A > V_r$	Enable the AND gate to pass the clock
6	0101	5/16	13/16	$V_A > V_r$	Enable the AND gate to pass the clock
7	0110	6/16	13/16	$V_A > V_r$	Enable the AND gate to pass the clock
8	0111	7/16	13/16	$V_A > V_r$	Enable the AND gate to pass the clock
9	1000	8/16	13/16	$V_A > V_r$	Enable the AND gate to pass the clock
10	1001	9/16	13/16	$V_A > V_r$	Enable the AND gate to pass the clock
11	1010	10/16	13/16	$V_A > V_r$	Enable the AND gate to pass the clock
12	1011	11/16	13/16	$V_A > V_r$	Enable the AND gate to pass the clock
13	1100	12/16	13/16	$V_A > V_r$	Enable the AND gate to pass the clock
14	1101	13/16	13/16	$V_A < V_r$	Disable the AND gate to pass the clock

In this ADC, the counter advances by one count for every clock pulse, therefore the clock speed decides the conversion speed. The conversion time is variable and it is proportional to the amplitude of the analog input voltage. The average conversion time of an n-bit ADC is,

$$T_C = \frac{2^n}{2} \times \frac{1}{f_{clock}}$$

$$= 2^{n-1} \times \frac{1}{f_{clock}} \qquad (9.16)$$

where f_{clock} is the frequency of clock cycle.

In this conversion process, the counter is reset for each discrete sample. Therefore, it takes a long conversion time.

9.3.4 Continuous Type ADC

The long conversion time of a counter type ADC can be eliminated by counting from the previously counted value, instead of resetting the counter for each conversion. The block diagram a 4-bit of continuous type ADC is shown in Fig. 9.26.

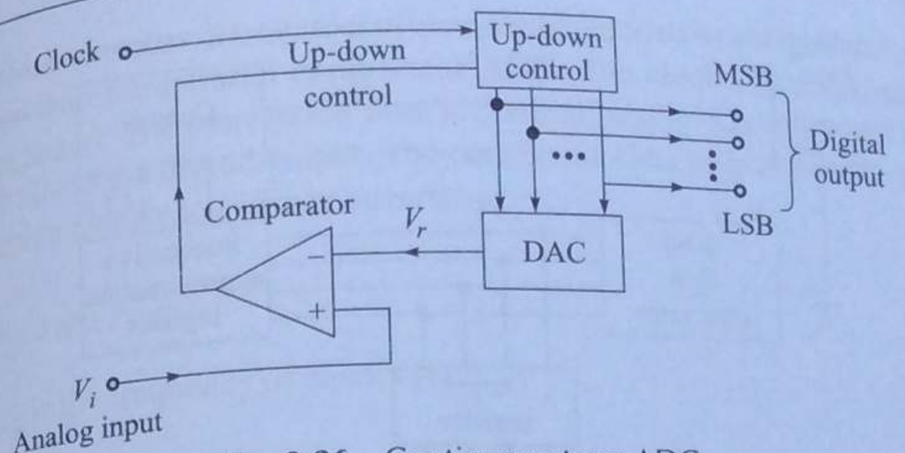


Fig. 9.26 Continuous type ADC

The block diagram consists of a DAC, a comparator, an up-down counter, and a control circuit. The DAC converts the digital data into analog signal, which is given to the inverting terminal of the comparator, and the non-inverting input of the comparator is connected to the unknown analog input voltage V_A. The op-amp is used in open loop. It acts as a comparator and compares the analog input that has to be converted into digital with the analog output of DAC. The control circuit controls the direction of up/down counter based on the result of comparator.

When $V_A > V_r$, the output of comparator is in high state. The counter is then made to count UP and the DAC output increases. The new input data is converted and compared with an unknown analog voltage. This process of counting-up continues, until V_r is less than V_A.

When $V_A < V_r$, the output of comparator is in low state. The counter is then made to count DOWN and the DAC output decreases. The new input data is converted and compared with an unknown analog voltage. This process of counting-down continues, until V_r is greater than V_A. Figure 9.27 shows the analog input signal and the input to up-down counter.

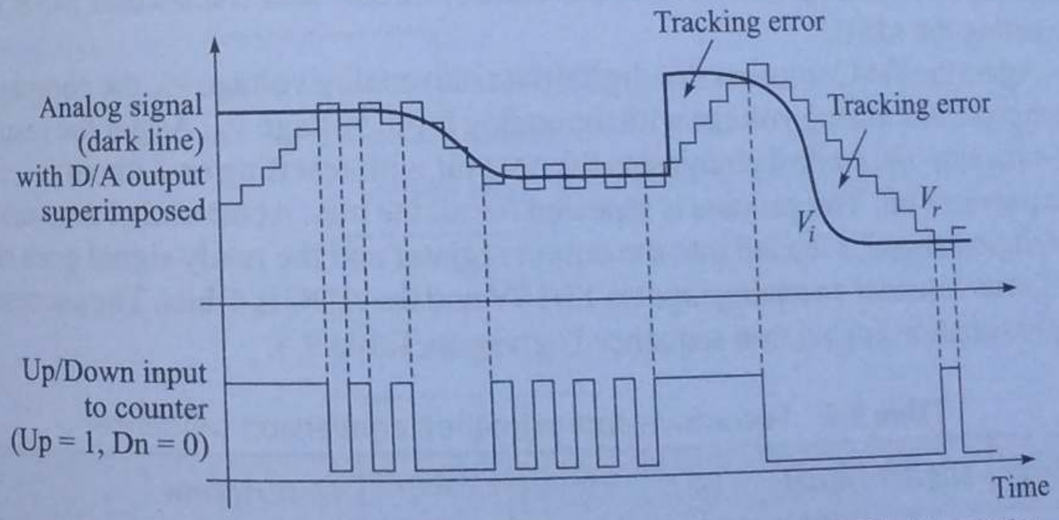


Fig. 9.27 The analog input signal and the input to UP/DOWN counter

9.3.5 Successive-approximation ADC

It is a high-resolution ADC that uses only one comparator with a variable reference voltage. The variable reference voltage can be obtained by a sequence or binary counter and a DAC. The basic principle of this ADC is that the unknown analog input voltage is approximate against an n-bit digital value by trying one bit at time, beginning with the MSB. The block diagram of a 4-bit successive-approximation ADC is shown in Fig. 9.28.

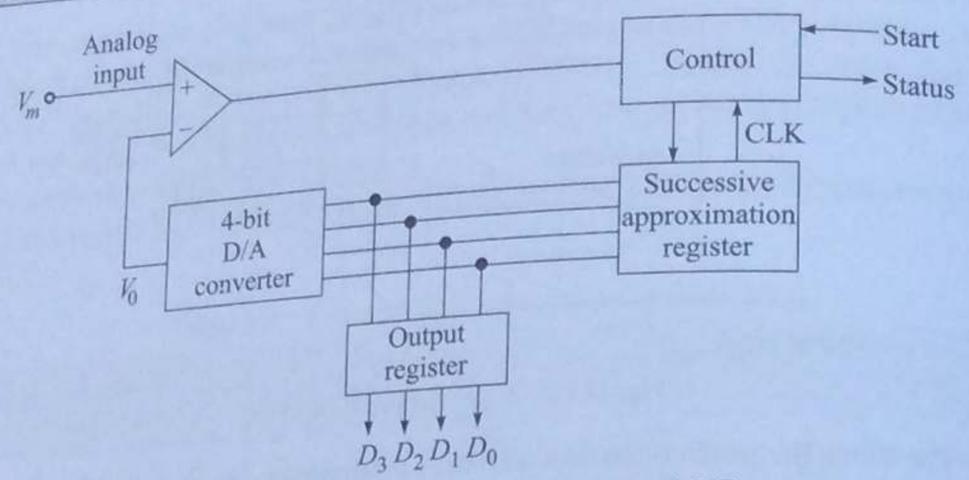


Fig. 9.28 Successive-approximation ADC

The block diagram consists of a DAC, a comparator, a successive-approximation register, a control circuit, and an output register. The DAC converts the digital data into analog signal, which is given as one of the input to the comparator. The op-amp is used in open loop. It acts as a comparator and compares the analog input that has to be converted into digital with the analog output of ADC. The result of the comparator may be $+V_{sat}$ or $-V_{sat}$, i.e. high or low (logic 1 or 0) which controls the control circuit. The control circuit changes the data of successive-approximation register as per the output signal of the comparator, as discussed below.

When the start signal is provided by a user, the MSB of successive-approximation register is set and other bits are reset with a clock pulse. The DAC converts this digital data into analog voltage V_0, the comparator compares this analog voltage with the analog input voltage V_A. When $V_A > V_0$, the output of comparator is high, the control circuit sets the second MSB without resetting the MSB. When $V_A < V_0$, the output of comparator is low, the control circuit sets the second MSB with resetting the MSB.

Again the DAC converts this digital data into analog voltage V_0, the comparator compares this analog voltage with the analog input voltage V_A. As per the result of comparator, the control circuit sets the next bit with resetting or without resetting the previous bit. This process is repeated for all the bits. At the end of conversion, the digital signal is loaded into the output register and the ready signal goes high.

Let us consider an analog input is 13/16V and the ADC is 4-bits. The successive approximation conversion sequence is given in Table 9.5.

Table 9.5 Successive-approximation conversion sequence

Clock	*Digital signal*	V_0	V_A		*Description*
1	1000	8/16	13/16	$V_A > V_0$	Set the MSB
2	1100	12/16	13/16	$V_A > V_0$	Set the second bit without resetting the MSB
3	1110	14/16	13/16	$V_A < V_0$	Set the next bit without resetting the previous bit
4	1101	13/16	13/16	$V_A = V_0$	Set the next bit with resetting the previous bit

Digital output is 1101.

For an N-bit converter, the number of clock pulse required would be N and hence, it is slower than the parallel comparator ADC. The clock speed decides the conversion speed. The conversion time is constant and it is proportional to the number of bits in the digital output. The conversion time of an N-bit successive approximation analog-to-digital converter is

$$T_C = N \times \frac{1}{f_{clock}} \tag{9.17}$$

where f_{clock} is the frequency of clock cycle.

9.3.6 Dual Slope ADC

In a dual slope ADC, the integrator generates two different ramps—one with an unknown analog input voltage V_A as the input, and another with a known reference voltage V_{ref} as the input. Hence, it is known as the dual slope ADC. The basic principle of a dual slope ADC is that, an analog signal is converted into a proportional time period, which is then measured using a digital counter. The block diagram of a typical dual slope ADC is shown in Fig. 9.29. It includes an integrator, comparator, counter, control logic, and a reference voltage.

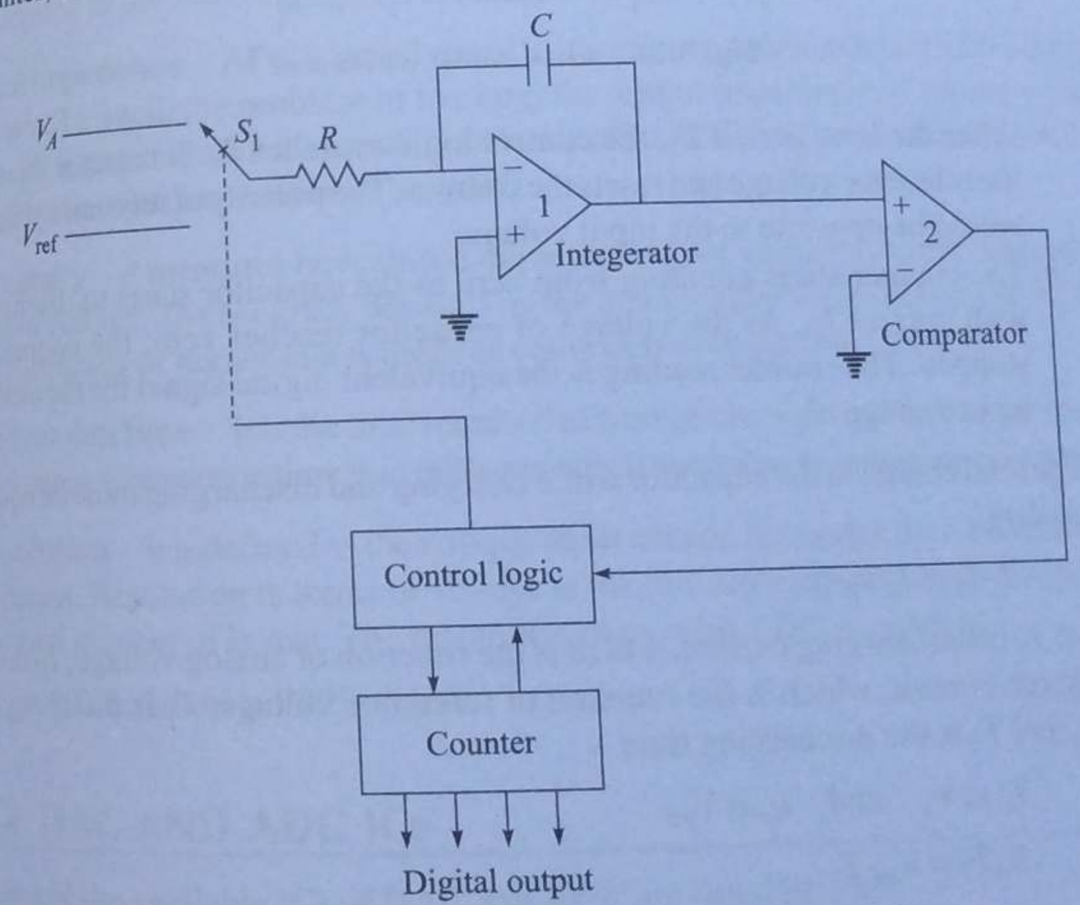


Fig. 9.29 Block diagram of a dual slope ADC

Op-amp 1 with resistor R and capacitor C acts as an integrator and when the input of integrator is connected to the analog input, the capacitor is charged linearly. The voltage across the capacitor depends on the amplitude of the analog input voltage and duration T.

Op-amp 2 is used in open loop. It acts as a comparator and compares the output of integrator with the reference input. The reference voltage is zero and the

comparator acts as a zero crossing detector. The output of the comparator is high or low depending on the voltage across the capacitor. When the voltage across the capacitor reaches zero, the output of the comparator is $-V_{sat}$, which is given to the control logic and the control logic stops the counting of counter.

The conversion process follows three steps.

- The switch S_1 connects the analog voltage V_A as the input to the integrator for a fixed period T_1. During this period, the capacitor of the integrator is charged up to a certain voltage with charging current I_C, which is the function of analog input voltage, as shown in Fig. 9.30.

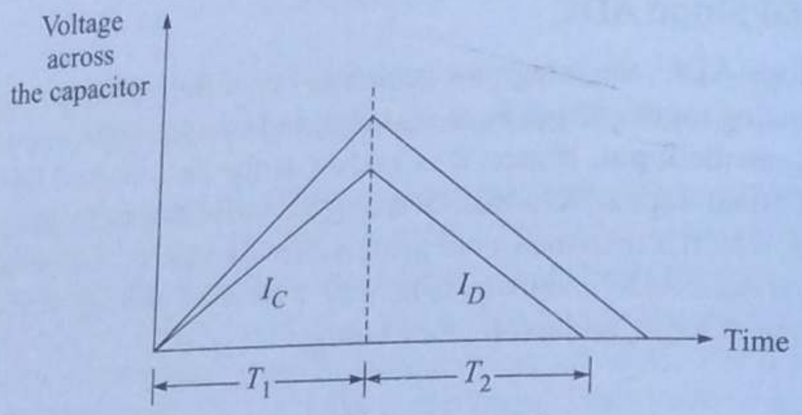


Fig. 9.30 Dual ramp waveform

- After the time period T_1, the control logic switches the integrator input to the reference voltage and resets the counter. The polarity of reference voltage would be opposite to the input voltage.
- The counter starts counting from zero as the capacitor starts to discharge with current I_D. As the voltage of capacitor reaches zero, the counter is stopped. The counter reading is the equivalent digital signal for the analog input voltage V_A.

The total charge in the capacitor while charging and discharging must be equal. Therefore,

$$I_C T_1 = I_D T_2$$

where I_C is the charging current, which is the function of analog voltage; I_D is the discharge current, which is the function of reference voltage; T_1 is the charging time; and T_2 is the discharging time.

$$I_C \alpha V_A \quad \text{and} \quad I_D \alpha V_{ref}$$

$$V_A T_1 = V_{ref} T_2$$

$$T_2 = \frac{T_1}{V_{ref}} \times V_A \tag{9.17}$$

The period T_1 and the reference voltage V_{ref} are constant.

$$\Rightarrow \quad \left(\frac{T_1}{V_{ref}}\right) = K$$

Using this in Eq. (9.17), we get

$$T_2 = KV_A \tag{9.18}$$

Thus, time period T_2 is proportional to analog voltage V_A and the counter measures the period T_2. Hence, the content of counter is the equivalent digital signal for the analog input V_A.

The accuracy of this ADC is high as compared to a successive-approximation ADC, but its speed is slow.

9.3.7 Specifications for ADC

The following specifications are usually specified by the manufactures of ADC.

(a) Range of input voltage
(b) Input impedance
(c) Accuracy
(d) Conversion time
(e) Resolution
(f) Differential linearity

Range of input voltages The input signal given to the ADC is an analog signal. The amplitude of a signal is the function of time. The range of the input voltage is defined by the maximum and minimum amplitude of an analog signal, which can be applied to the ADC for reliable operation.

Input impedance ADC is mainly used to interface the analog circuit with a digital system. To avoid the problem of loading, the output impedance of analog circuit should be matched with the input impedance of an ADC and hence, it is required to specify the input impedance of an ADC.

Accuracy It measures how closed the actual output voltage is to the theoretical output voltage. The accuracy depends on the quantization level (the size of ADC). The accuracy of higher bits is more as compared to lower bits.

Conversion time It is the time required to convert the analog data into its digital equivalent. Conversion time is in milliseconds. It should be as minimum as possible.

Resolution It is defined as the voltage input change necessary for a 1-bit change in output. Resolution in terms of voltage is the full-scale input voltage divided by the total number of levels. The resolution of an n-bit ADC is given as,

$$\text{Resolution} = V/(2^n - 1) \tag{9.19}$$

9.4 DAC AND ADC ICs

Some of the available ICs of DAC and ADC are listed below.

IC number	Description
AD DAC 80	12-bit DAC
AD 7522	10-bit DAC
AD 558	8-bit DAC
ADC 0809	8-bit ADC
AD ADC 80	12-bit ADC
ADC-7109	12-bit binary ADC

9.5 ADC 0809 (8-BIT ADC)

Introduction

The ADC 0809 data acquisition component is a monolithic CMOS device with an 8-bit ADC, an 8-channel multiplexer, and a microprocessor-compatible control logic. The 8-bit ADC uses successive approximation as the conversion technique. The converter features a high-impedance chopper-stabilized comparator, a 256*R* voltage divider with analog switch tree and a successive-approximation register. The 8-channel multiplexer can directly access any one of the 8-single-ended analog signals.

The device eliminates the need for external zero and full-scale adjustments. Easy interfacing to microprocessors is provided by the latched and decoded multiplexer address inputs and latched TIL TRI-STATE outputs. Incorporating the most desirable aspects of several A/D conversion techniques has optimized the design of ADC 0809. It offers high-speed, high-accuracy, minimal temperature dependence, excellent long-term accuracy and repeatability, and consumes minimal power. These features make this device ideally suited from applications from process and machine control to consumer and automotive applications.

Features

1. Easy interface to all microprocessors
2. Operates ratiometrically or with 5 V_{DC} or analog span adjusted voltage reference
3. No zero or full-scale adjustment required
4. 8-channel multiplexer with address logic
5. 0 V to 5 V input range with single 5 V power supply
6. Outputs meet TTL voltage level specifications
7. Standard hermetic or molded 28-pin DIP package
8. 28-pin molded IC carrier package
9. Resolution 8 bits
10. Total unadjusted error ±1/2 LSB and ±1 LSB
11. Single supply 5 V_{DC}
12. Low power 15 mW
13. Conversion time 100 μs

Functional block diagram of ADC 0809

Figure 9.31 shows the block diagram of ADC 0809.

Multiplexer

The device contains an 8-channel single-ended analog signal multiplexer. A particular input channel is selected by using the address decoder. Table 9.6 shows the input states for the address lines to select any channel. The address is latched into the decoder on low-to-high transition of the address latch enable signal.

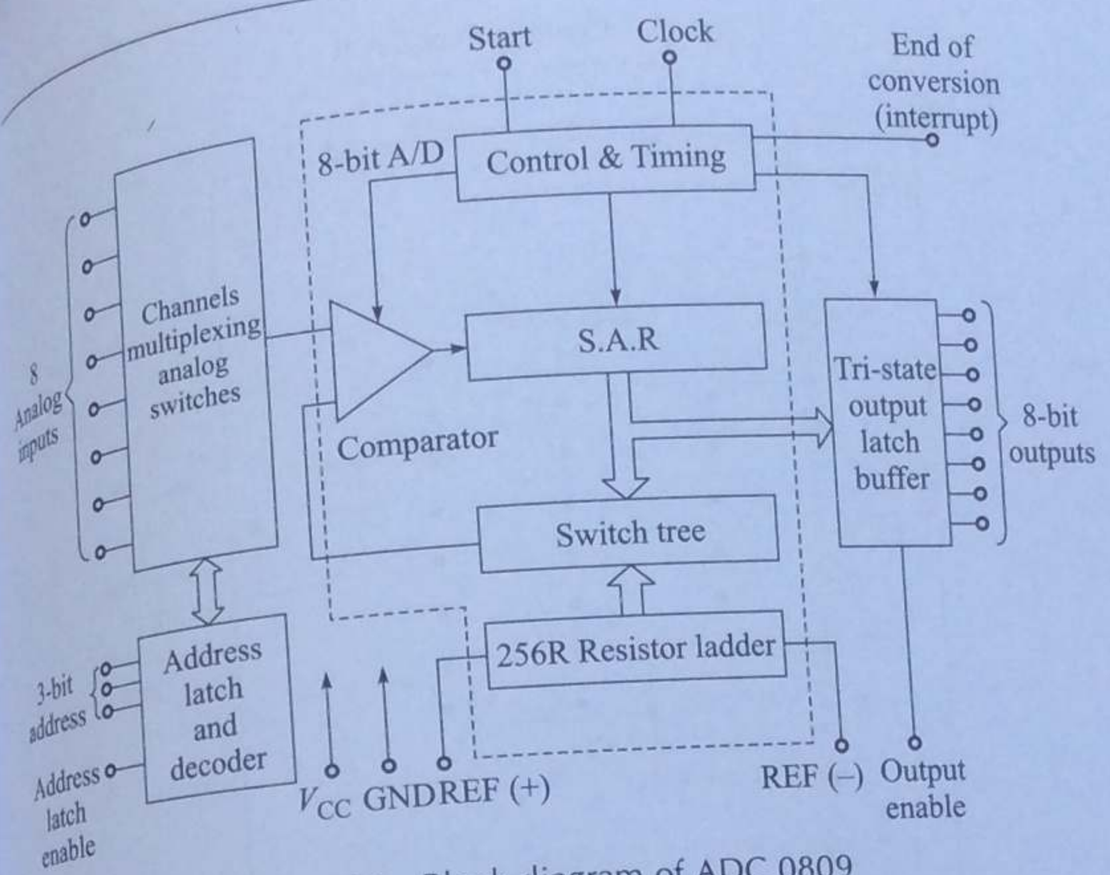


Fig. 9.31 Block diagram of ADC 0809

Table 9.6 Input states for the address lines to select channel

Selected analog channel	*Address line*		
	C	*B*	*A*
IN0	0	0	0
IN1	0	0	1
IN2	0	1	0
IN3	0	1	1
IN4	1	0	0
IN5	1	0	1
IN6	1	1	0
IN7	1	1	1

Converter characteristics

The heart of this single IC data acquisition system is its 8-bit ADC. The converter is designed to provide fast, accurate, and repeatable conversions over a wide range of temperatures. The converter is partitioned into three major sections, the 256*R* ladder network, the successive-approximation register, and the comparator.

The 256*R* ladder network approach shown in Fig. 9.32 was chosen over the conventional *R*/2*R* ladder network because of its inherent monotonicity, which guarantees no missing digital codes. Monotonicity is particularly important in closed loop feedback control systems. A non-monotonic relationship can cause oscillations that will be catastrophic for the system. Additionally, the 256*R* network does not cause load variations on the reference voltage.

The bottom resistor and the top resistor of the ladder network in Fig. 9.32 are not the same value as the remainder of the network. The difference in these resistors causes the output characteristics to be symmetrical with the zero and full-scale points of the transfer curve. The first output transition occurs when the analog signal has

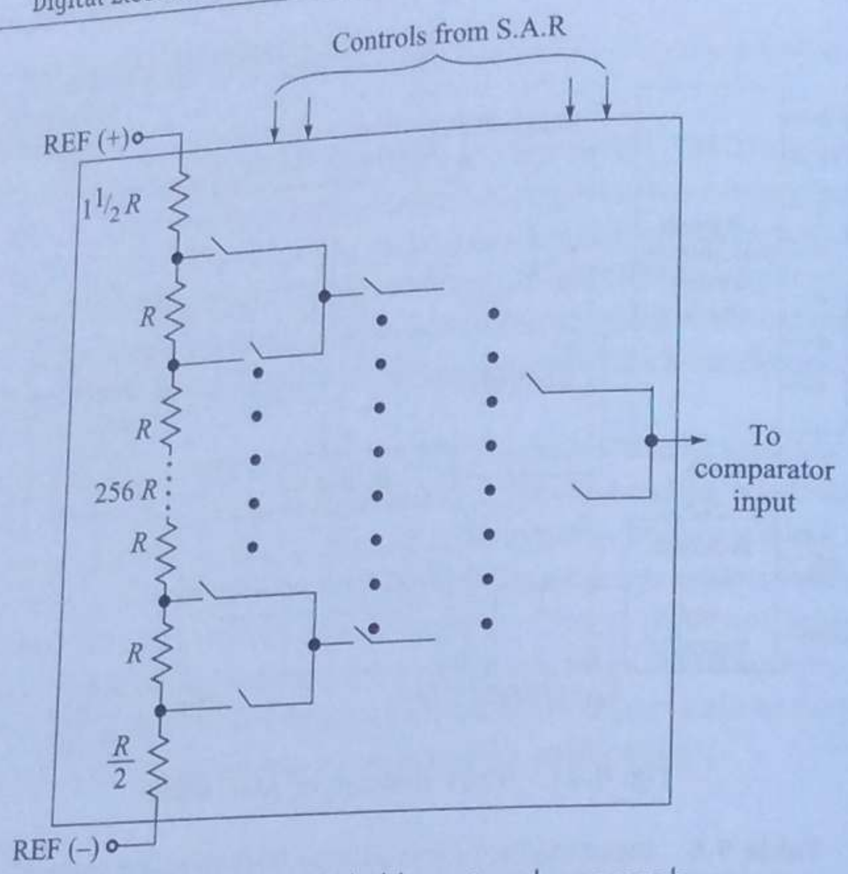


Fig. 9.32 256*R* ladder network approach

reached $+\frac{1}{2}$ LSB and the succeeding output transition occurs every 1 LSB up to full-scale.

The successive approximation register (SAR) performs eight iterations to approximate the input voltage. For any type SAR converter, *n*-iterations are required for an *n*-bit converter. Figure 9.33 shows a typical example of a 3-bit converter. In ADC 0809, the approximation technique is extended to 8-bits using the 256*R* network.

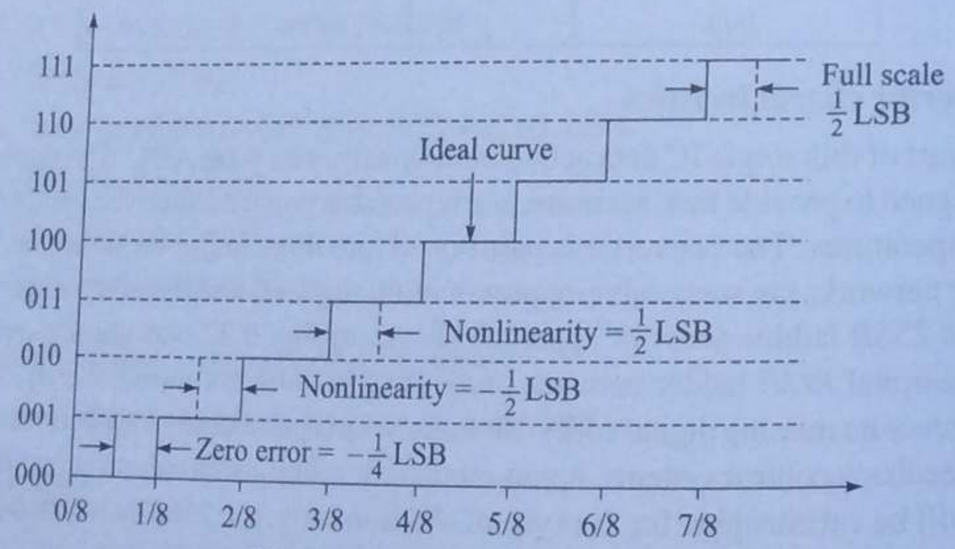


Fig. 9.33 3-bit transfer curve

The ADC successive approximation register (SAR) of ADC is reset on the positive edge of the start conversion (SC) pulse. The conversion process starts at the falling edge of the start conversion pulse. A conversion process will be interrupted

by receipt of a new start conversion pulse. Continuous conversion may be accomplished by tying the end-of-conversion (EOC) output of the SC input. If used in this mode, an external start conversion pulse should be applied after power up. EOC will go low between 0 to 8 clock pulses after the rising edge of start conversion.

The most important section of an ADC is its comparator. This section is responsible for the ultimate accuracy of the entire converter.

The chopper-stabilized comparator converts the DC input signal into an AC signal. This signal is then fed through a high-gain AC amplifier and has the DC level restored. This technique limits the drift component of the amplifier, since the drift is a DC component, which is not passed by the AC amplifier. This makes the entire ADC extremely insensitive to temperature, long-term drift, and input offset errors.

The pin configuration of ADC 0809 is given in Fig. 9.34.

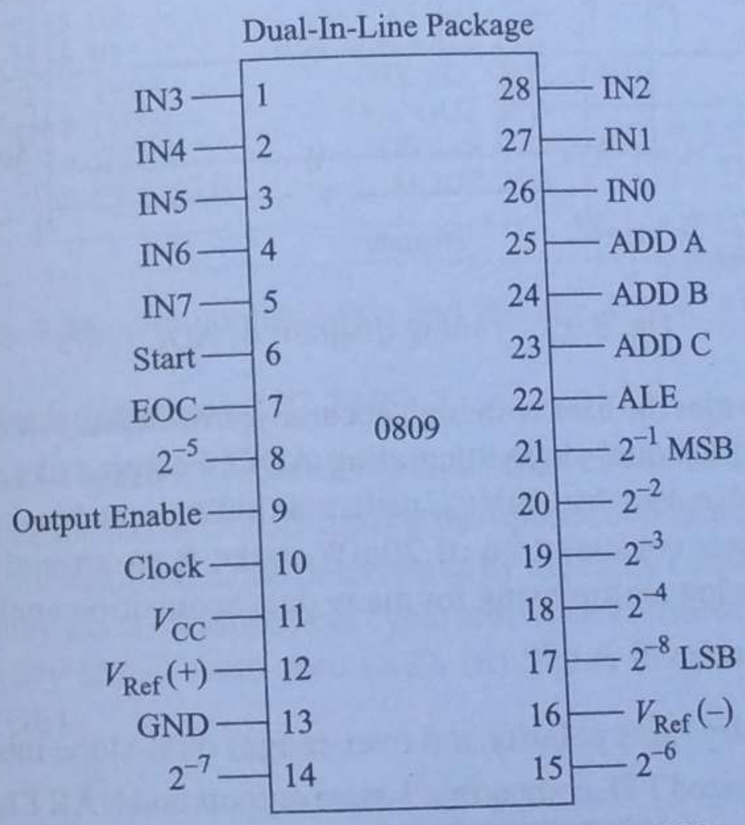


Fig. 9.34 Pin configuration of ADC 0809

The timing diagram is given in Fig. 9.35.

9.6 ADC-7109 (12-BIT BINARY ADC)

Introduction

The ICL7109 is a high performance, low power integrating 12-bit ADC, designed to easily interface with microprocessors.

The output data (12-bits), polarity, and over-range may be directly accessed under the control of two byte enable inputs and an IC select input for a simple parallel bus interface. A UART handshake mode is provided to allow the IC 7109 to work with industry-standard UARTs in providing serial data transmission, ideal for remote data logging applications. The RUN/$\overline{\text{HOLD}}$ input and STATUS output allow monitoring and control of conversion timing.

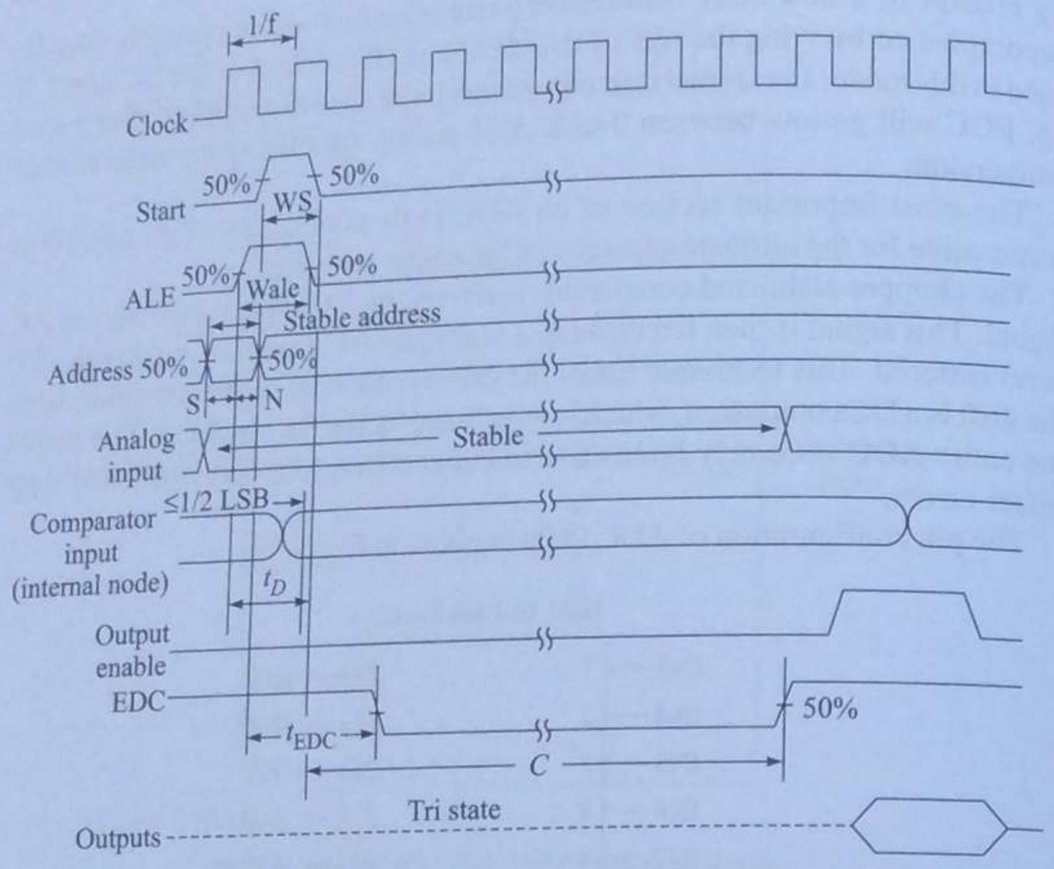


Fig. 9.35 Timing diagram of ADC 0809

IC 7109 provides the user with high accuracy, low noise, low drift, versatility, and economy of the dual-slope integrating ADC. Features like true differential input and reference, less than 1μV/°C drift, maximum input bias current of 10 pA, and typical power consumption of 20mW make it an attractive per-channel alternative to analog multiplexing for many data acquisition applications.

Features

1. 12-bit binary (plus polarity and over-range) dual-slope integrating ADC.
2. Byte-organized TTL compatible 3-state outputs and UART handshake mode for simple, parallel, or serial interfacing to microprocessor systems.
3. RUN/$\overline{\text{HOLD}}$ input and STATUS output can be used to monitor and control conversion timing.
4. True differential input and differential reference.
5. Low noise—typically 15 μV p-p.
6. 1 pA typical input current.
7. Operates at up to 30 conversions per second.
8. On-IC oscillator operates with inexpensive 3.58 MHz TV crystal giving 7.5 conversions per second for 60 Hz rejections. May also be operated as RC oscillator for other clock frequencies.
9. Fabricated using MAX-CMOS technology combining analog and digital functions on a single low-power LSI CMOS IC.
10. All inputs are fully protected against static discharge, no special handling precautions necessary.

Figure 9.36 shows the pin configuration and test circuit of IC 7109.

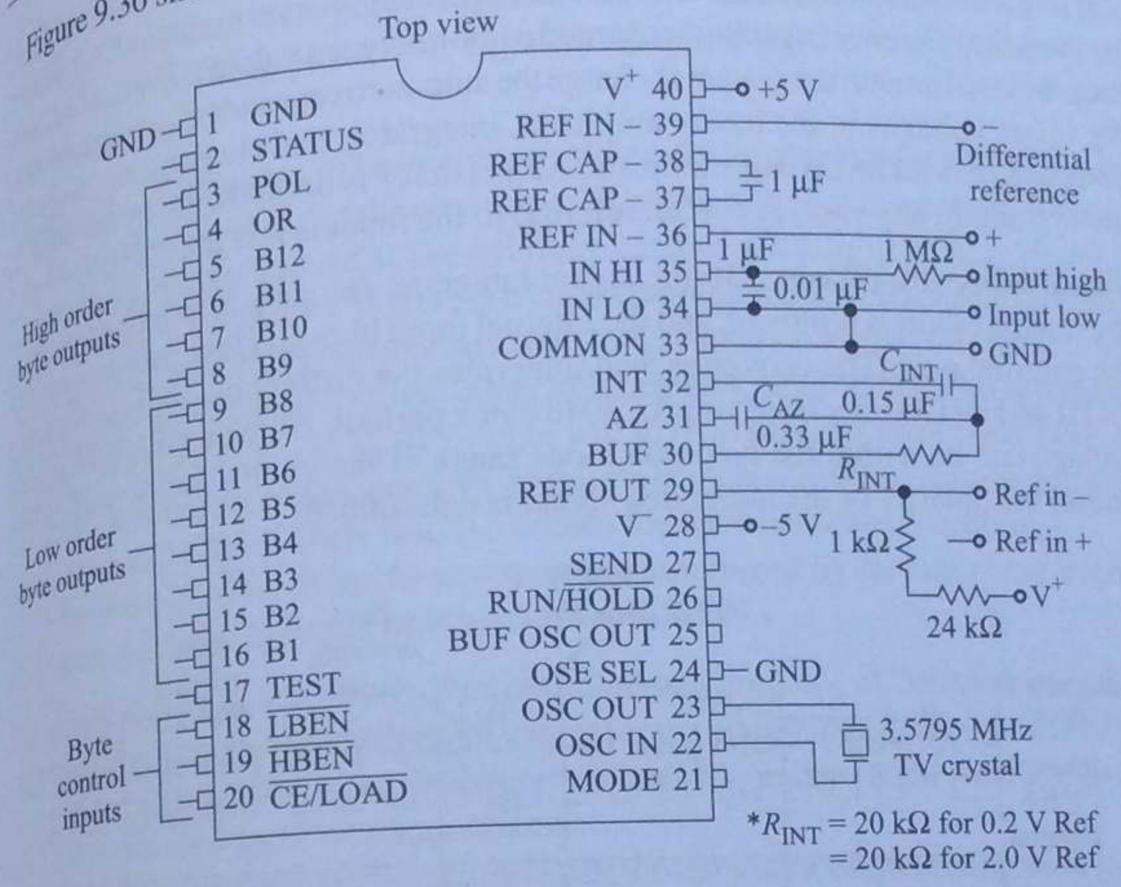


Fig. 9.36 Pin configuration and test circuit of IC 7109

Functional block diagram of IC 7109

Analog section Figure 9.37 shows the equivalent circuit of the analog section of IC 7109. When the RUN/$\overline{\text{HOLD}}$ input is left open or connected to V^+, the circuit will perform conversions at a rate determined by the clock frequency (8192 clock periods per cycle). Each measurement cycle is divided into three phases, as shown in Fig. 9.38. They are (i) Auto-zero (AZ), (ii) Signal integrate (INT), and (iii) De-integrate (DE).

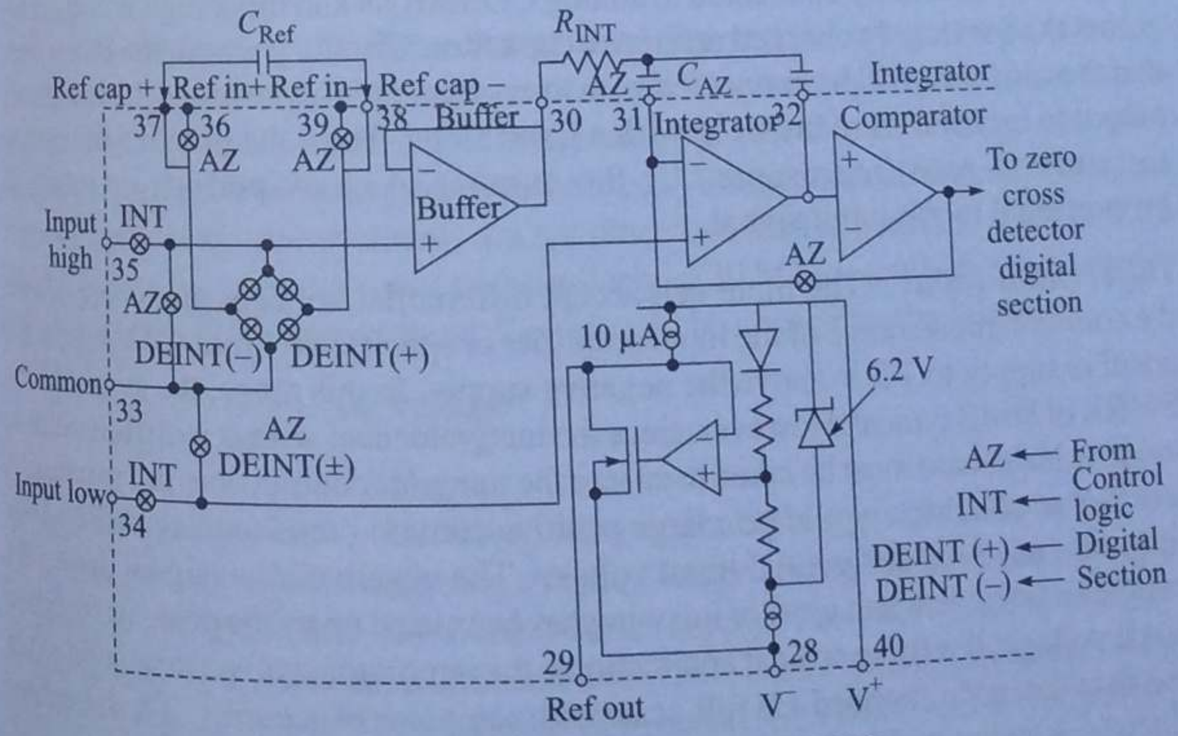


Fig. 9.37 Analog section of IC 7109

Auto-zero phases During auto-zero, three things happen. First, the input high and low are disconnected from their pins and internally shorted to analog COMMON. Second, the reference capacitor is charged to the reference voltage. Third, a feedback loop is closed around the system to charge the auto-zero capacitor C_{AZ} to compensate for offset voltages in the buffer amplifier, integrator, and comparator. Since the comparator is included in the loop, the AZ accuracy is limited only by the noise of the system. In any case, the offset referred to the input is less than 10 μV.

Signal integrate phase During signal integrate, the auto-zero loop is opened, the internal short is removed, and the internal input high and low are connected to the external pins. The converter then integrates the differential voltage between IN HI and IN LO for a fixed time of 2048 clock periods. Note that this differential voltage can be within the common mode range of the inputs. At the end of this phase, the polarity of the integrated signal is determined.

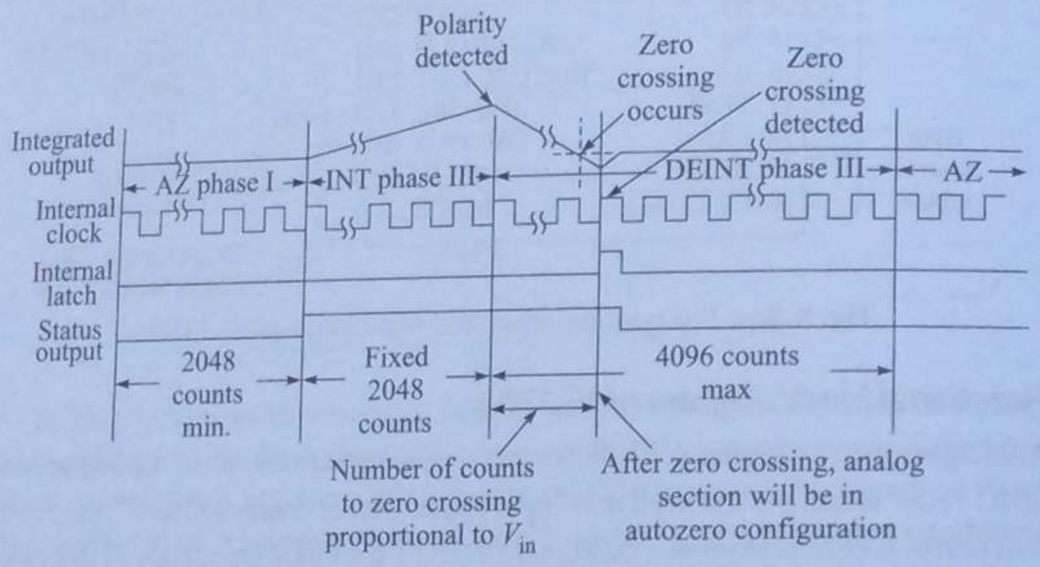


Fig. 9.38 Conversion timing RUN/$\overline{\text{HOLD}}$ pin high

De-integrate phase The final phase is de-integrated, or reference integrated. Input low is internally connected to analog COMMON and input high is connected across the previously charged reference capacitor. Circuitry within the IC ensures that the capacitor will be connected with the correct polarity to cause the integrator output to return to zero crossing with a fixed slope. Thus, the time for the output to return to zero (represented by the number of clock periods counted) is proportional to the input signal.

Differential input The input can accept differential voltages anywhere within the common mode range of the input amplifier or specifically from 1.0 V below the positive supply to 1.5 V above the negative supply. In this range, the system has a CMRR of 86 dB typical. However, since the integrator also swings with the common mode voltage, care must be taken to ensure the integrator output does not saturate. A worst-case condition would be a large positive common mode voltage with a near full-scale negative differential input voltage. The negative input signal drives the integrator positive when most of its swing has been used up by the positive common mode voltage. For these critical applications, the integrator swing can be reduced to less than the recommended 4 V full-scale with some loss of accuracy. The integrator output can swing within 0.3 V of either supply without loss of linearity.

The IC7109 has however been optimized for operation with analog COMMON near digital ground. With power supplies of + 5 V and –5 V, it allows a 4 V full-scale integrator swing positive or negative, maximizing the performance of the analog section.

Differential reference The reference voltage can be generated anywhere within the power supply voltage of the converter. The main source of common mode error is a roll-over voltage caused by the reference capacitor, losing or gaining charge to stray capacity on its nodes. If there is a large common mode voltage, the reference capacitor can gain charge when called up to de-integrate a positive signal, but it loses charge when called up to de-integrate a negative input signal. This difference in reference for (+) or (–) input voltage will give a roll-over error. However, by selecting the reference capacitor large enough in comparison to the stray capacitance, this error can be held at less than 0.5 counts for the worst-case condition.

The roll-over error from these sources is minimized by having the reference common mode voltage near or at analog COMMON.

Component value selection For optimum performance of the analog section, care must be taken in the selection of values for the integrator capacitor and resistor, auto-zero capacitor, reference voltage, and conversion rate. These values must be chosen to suit the particular application.

The most important consideration is that the integrator output swing be as large as possible. For example, with ± 5 V supplies and COMMON connected to GND, the nominal integrator output swing at full scale is ± 4 V. Since the integrator output can go to 0.3 V from either supply without significantly affecting linearity, a 4 V integrator output swing allows 0.7 V for variations in output swing due to component value and oscillator tolerances. With ±5 V supplies and a common mode range of ±1 V required, the component values should be selected to provide ±3 V integrator output swing. Noise and roll-over errors will be slightly worse than in the ±4 V case. For larger common mode voltage ranges, the integrator output swing must be reduced further. This will increase both noise and roll-over errors. To improve the performance, supplies of ±6 V may be used.

Integrating resistor Both the buffer amplifier and the integrator have a class A output stage with 100 μA of quiescent current. They supply 20 μA of drive current with negligible non-linearity. The integrating resistor should be large enough to remain in this very linear region over the input voltage range, but small enough so that undue leakage requirements are not placed on the PC board. For 4.096 V full scale, 200 kΩ is near optimum and similarly, a 20 kΩ for a 409.6 mV scale. For other values of full-scale voltage, R_{INT} should be chosen by the relation,

$$R_{INT} = \frac{\text{Full-scale valtage}}{20\ \mu A} \tag{9.20}$$

Integrating capacitor The integrating capacitor C_{INT} should be selected to provide the maximum integrator output voltage swing without saturating the integrator approximately 0.3 V from either supply. For the ICL7109 with ±5 V supplies and analog common connected to GND, a ±3.5 to ±4 V integrator output swing is nominal. For $7^{1/2}$ conversions per second (61.72 kHz clock frequency) as provided by the crystal oscillator, the nominal values for C_{INT} and C_{AZ} are 0.15 mF

and 0.33 mF, respectively. If different clock frequencies are used, these values should be changed to maintain the integrator output voltage swing. In general, the value of C_{INT} is given by,

$$C_{INT} = \frac{(2048 \times \text{clock period})(20\ \mu\text{A})}{\text{integrator output voltage swing}} \qquad (9.21)$$

An additional requirement of the integrating capacitor is that it has low dielectric absorption to prevent roll-over errors. While other types of capacitors are adequate for this application, polypropylene capacitors give undetectable errors at reasonable cost up to 85°C. For the military temperature range, Teflon capacitors are recommended.

Auto-zero capacitor The size of auto-zero capacitor has some influence on the noise of the system—a big capacitor produces less noise. However, it cannot be increased without limits since it, in parallel with the integrating capacitor, forms an R-C time constant that determines the speed of recovery from overloads and more important, the error that exists at the end of an auto-zero cycle. For 409.6 mV full-scale where noise is very important and the integrating resistor small, a value of C_{AZ} twice C_{INT} is optimum. Similarly for 4.096V full-scale where recovery is more important than noise, a value of C_{AZ} equal to half of C_{INT} is recommended. For optimal rejection of stray pickup, the outer foil of C_{AZ} should be connected to the R-C summing junction and the inner foil to pin 31. Similarly the outer foil of C_{INT} should be connected to pin 32 and the inner foil to the R-C summing junction. Teflon or equivalent capacitors are recommended above 85°C for their low leakage characteristics.

Reference capacitor A 1 μF capacitor gives good results in most applications. However, where a large common mode voltage exists and a 409.6 μV scale is used, a larger value is required to prevent roll-over error. Generally 10 μF will hold the roll-over error to 0.5 count in this instance. Again, Teflon or equivalent capacitors should be used for temperatures above 85°C for their low leakage characteristics.

Reference voltage The analog input required to generate a full-scale output of 4096 counts is $V_{IN} = 2V_{REF}$. Thus for a normalized scale, a reference of 2.048 V should be used for a 4.096 V full-scale, and 204.8 mV should be used for a 0.4096 V full-scale. However, in many applications where the A/D is sensing the output of a transducer, there will exist a scale factor other than unity between the absolute output voltage to be measured and a desired digital output. For instance, in a weighing system, the designer might like to have a full-scale reading when the voltage from the transducer is 0.682 V. Instead of dividing the input down to 409.6 mV, the input voltage should be measured directly and a reference voltage of 0.341 V should be used. Suitable values for integrating resistor and capacitor are 34 K and 0.15 mF. This avoids a divider on the input. Another advantage of this system is when a zero reading is desired for a non-zero input. Temperature and weight measurements with an offset are the examples. The offset may be introduced by connecting the voltage output of the transducer between common and analog high, and the offset voltage between common and analog low, observing polarities carefully. However, in processor-based systems using the ICL7109, it may be more efficient to perform this type of scaling subtraction digitally using software.

Digital section The digital section includes the clock oscillator and a scaling circuit, a 12-bit binary counter with output latches and TTL compatible 3-state output driver's polarity, over-range and control logic, and UART handshake logic, as shown in Fig. 9.39.

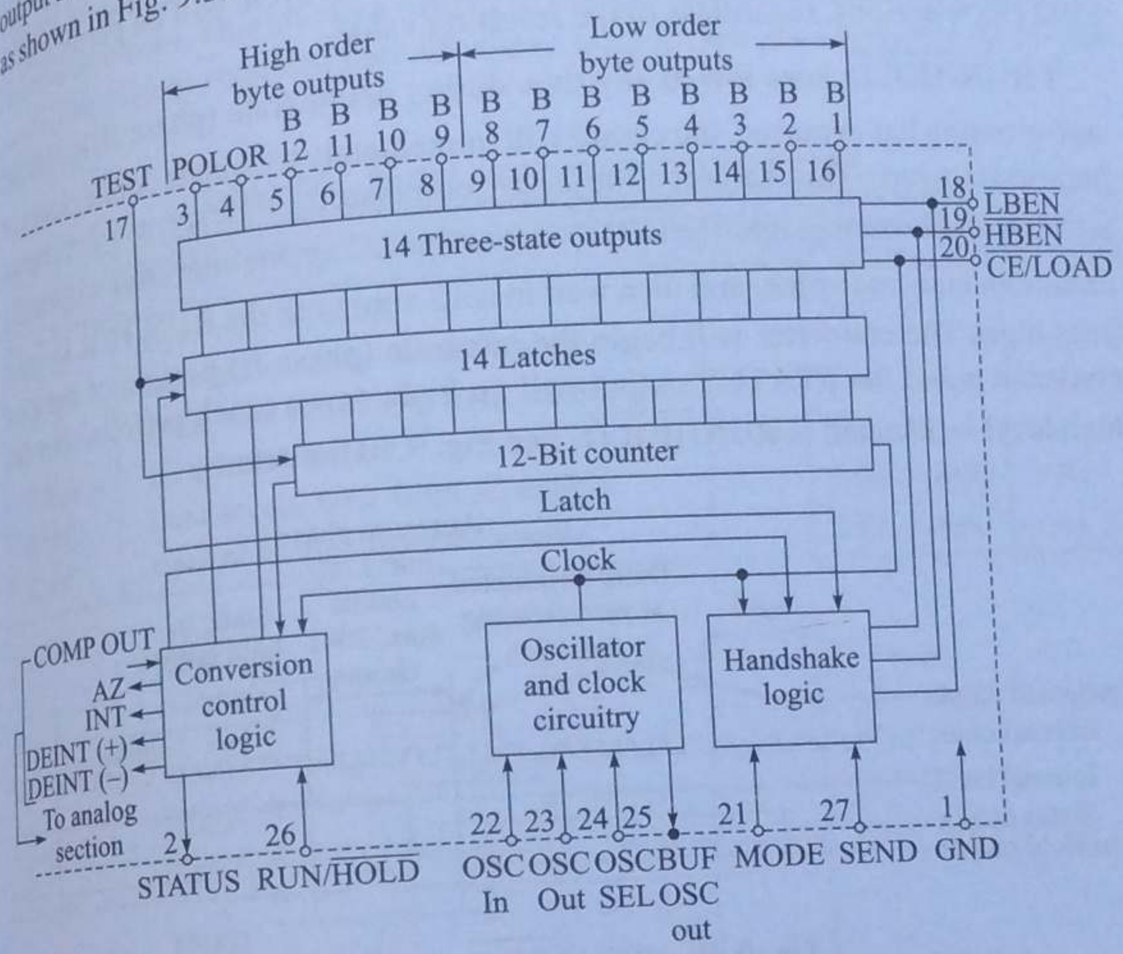


Fig. 9.39 Digital section of IC 7109

Throughout this description, logic levels will be referred to as *low* or *high*. For minimum power consumption, all inputs should swing from GND (low) to V^+ (high). Inputs driven from TTL gates should have 3.5 kΩ pull-up resistors added for maximum noise immunity.

MODE input The MODE input is used to control the output mode of the converter. When the MODE pin is low or left open (this input is provided with a pull-down resistor to ensure a low level when the pin is left open), the converter is in its *direct* output mode, where the output data is directly accessible under the control of the IC and byte enables inputs. When the MODE input is pulsed high, the converter enters the UART handshake mode and outputs the data in two bytes, then returns to *direct* mode. When the MODE input is left high, the converter will output data in the handshake mode at the end of every conversion cycle.

STATUS output During a conversion cycle, the STATUS output goes high at the beginning of signal integrate (phase II), and goes low one-half clock period after new data from the conversion has been stored in the output latches. See Fig. 9.36 for details of this timing. This signal may be used as a *data valid* flag (data never changes while STATUS is low) to drive interrupts, or for monitoring the status of the converter.

RUN/$\overline{\text{HOLD}}$ input When the RUN/$\overline{\text{HOLD}}$ input is high or left open, the circuit will continuously perform conversion cycles, updating the output latches after zero crossing during the deintegrate (phase III) portion of the conversion cycle (see Fig. 9.36). In this mode of operation, the conversion cycle will be performed in 8192 clock periods, regardless of the resulting value.

If RUN/$\overline{\text{HOLD}}$ goes low at any time during deintegrate (phase III) after the zero crossing has occurred, the circuit will immediately terminate deintegrate and jump to auto-zero. This feature can be used to eliminate the time spent in deintegrate after the zero crossing. If RUN/$\overline{\text{HOLD}}$ stays or goes low, the converter will ensure minimum auto-zero time, and then wait in auto-zero until the RUN/$\overline{\text{HOLD}}$ input goes high. The converter will begin the integrate (phase II) portion of the next conversion and the STATUS output will go high, seven clock periods after the high level is detected at RUN/$\overline{\text{HOLD}}$. See Fig. 9.40 for details.

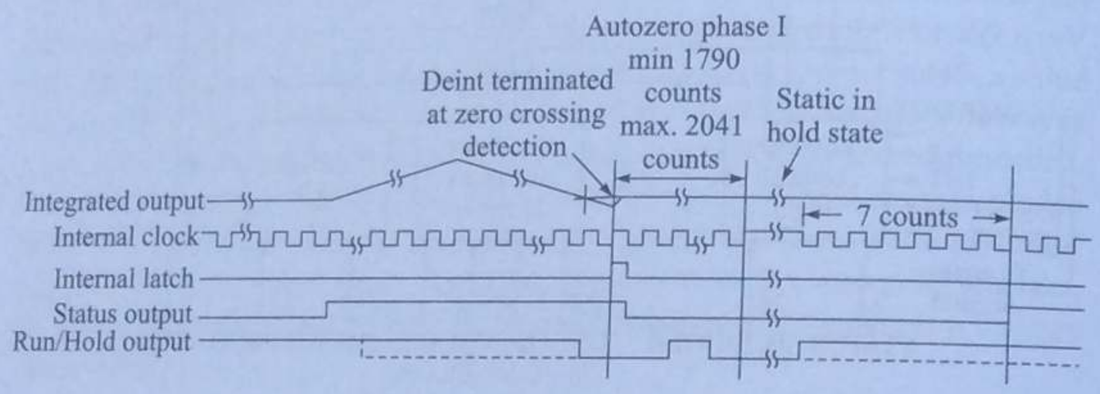


Fig. 9.40 RUN/$\overline{\text{HOLD}}$ operation

Using the RUN/$\overline{\text{HOLD}}$ input, in this manner, allows an easy *convert on demand* interface to be used. The converter may be held at idle in auto-zero with RUN/$\overline{\text{HOLD}}$ low. When RUN/$\overline{\text{HOLD}}$ goes high, the conversion is started, and when the STATUS output goes low, the new data is valid (or transferred to the UART, see Handshake Mode). RUN/$\overline{\text{HOLD}}$ may now go low terminating deintegrate and ensuring a minimum auto-zero time before stopping to wait for the next conversion.

Alternately, RUN/$\overline{\text{HOLD}}$ can be used to minimize the conversion time by ensuring that it goes low during deintegrate, after zero crossing and goes high after the hold point is reached. The required activity on the RUN/$\overline{\text{HOLD}}$ input can be provided by connecting it to the Buffered Oscillator Output. In this mode, the conversion time is dependent on the input value measured. Also refer to Intersil Application Bulletin A032 for a discussion of the effects this will have on auto-zero performance.

If the RUN/$\overline{\text{HOLD}}$ input goes low and stays low during auto-zero (phase I), the converter will simply stop at the end of auto-zero and wait for RUN/$\overline{\text{HOLD}}$ to go high. As above, integrate (phase II) begins seven clock periods after the high level is detected.

Direct mode When the MODE pin is left at a low level, the data outputs (bits 1 through 8 low-order byte, bits 9 through 12, polarity and over-range high-order

byte) are accessible under the control of the byte and IC enable terminals as inputs. These three inputs are all active low, and are provided with pull-up resistors to ensure an inactive high level when left open. When the IC enable input is low, taking a byte enable input low will allow the outputs of that byte to become active (three-started on). This allows a variety of parallel data accessing techniques to be used. The timing requirements for these outputs are shown in Fig. 9.41 and Table 9.6.

Table 9.6 Direct mode timing requirements

Symbol	*Desciption*	*Min*	*Typ*	*Max*	*UNITS*
tBEA	Byte enable width	350	220		ns
tDAB	Data access time from byte enable		210	350	ns
tDHB	Data hold time from byte enable		150	300	ns
tCEA	IC enable width	400	260		ns
tDAC	Data access time from IC enable		260	400	Ns
tDHC	Data hold time from IC enable		240	400	ns

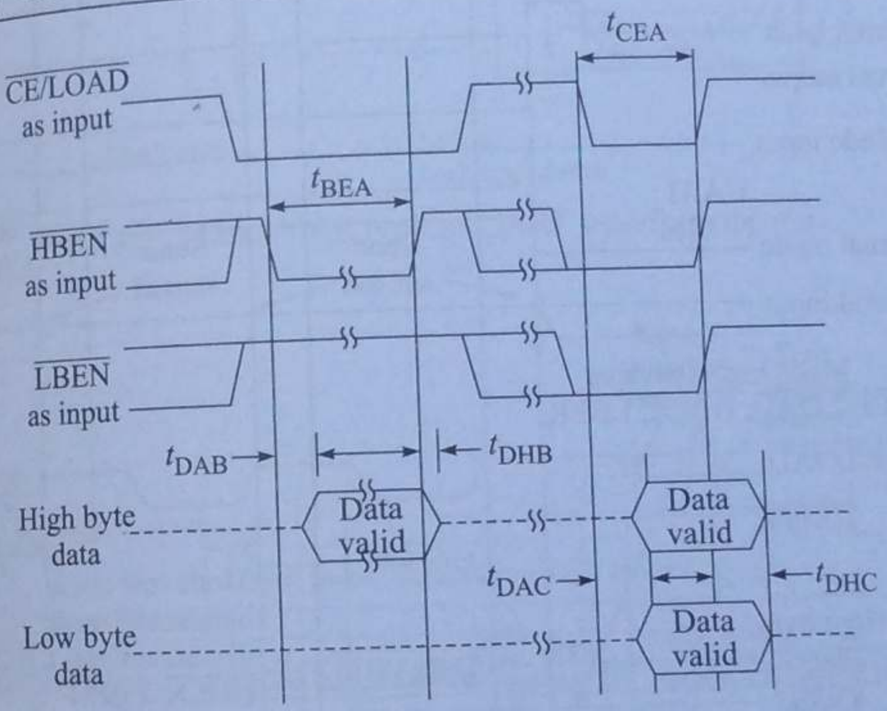


Fig. 9.41 Direct mode output timing

It should be noted that these control inputs are asynchronous with respect to the converter clock—the data may be accessed any time. Thus, it is possible to access the data while it is being updated, which can lead to scrambled data. Synchronizing the access of data with the conversion cycle by monitoring the STATUS output will prevent this. Data is never updated while STATUS is low.

Handshake mode The handshake output mode is provided as an alternative means of interfacing the IC 7109 to digital systems, where the ADC becomes active in controlling the flow of data instead of passively responding to IC and byte enable inputs. This mode is specifically designed to allow a direct interface between the IC 7109 and industry standard UARTs (such as the intersil CMOS UARTs, IM6402/3) with no external logic required. When triggered into the handshake mode, the IC 7109 provides all the control and flag signals necessary to sequence

the two bytes of data into the UART and initiate their transmission in serial form. This greatly eases the task and reduces the cost of designing remote data acquisition stations using serial data transmission to minimize the number of lines to the central controlling processor.

Entry into the handshake mode is controlled by the MODE pin. When the MODE terminal is held high, IC 7109 will enter the handshake mode after new data has been stored in the output latches at the end of every conversion performed (see Figs. 9.42 and 9.43). The MODE terminal may also be used to trigger entry into the handshake mode on demand. At any time during the conversion cycle, the low to high transition of a short pulse at the MODE input will cause immediate entry into the handshake mode.

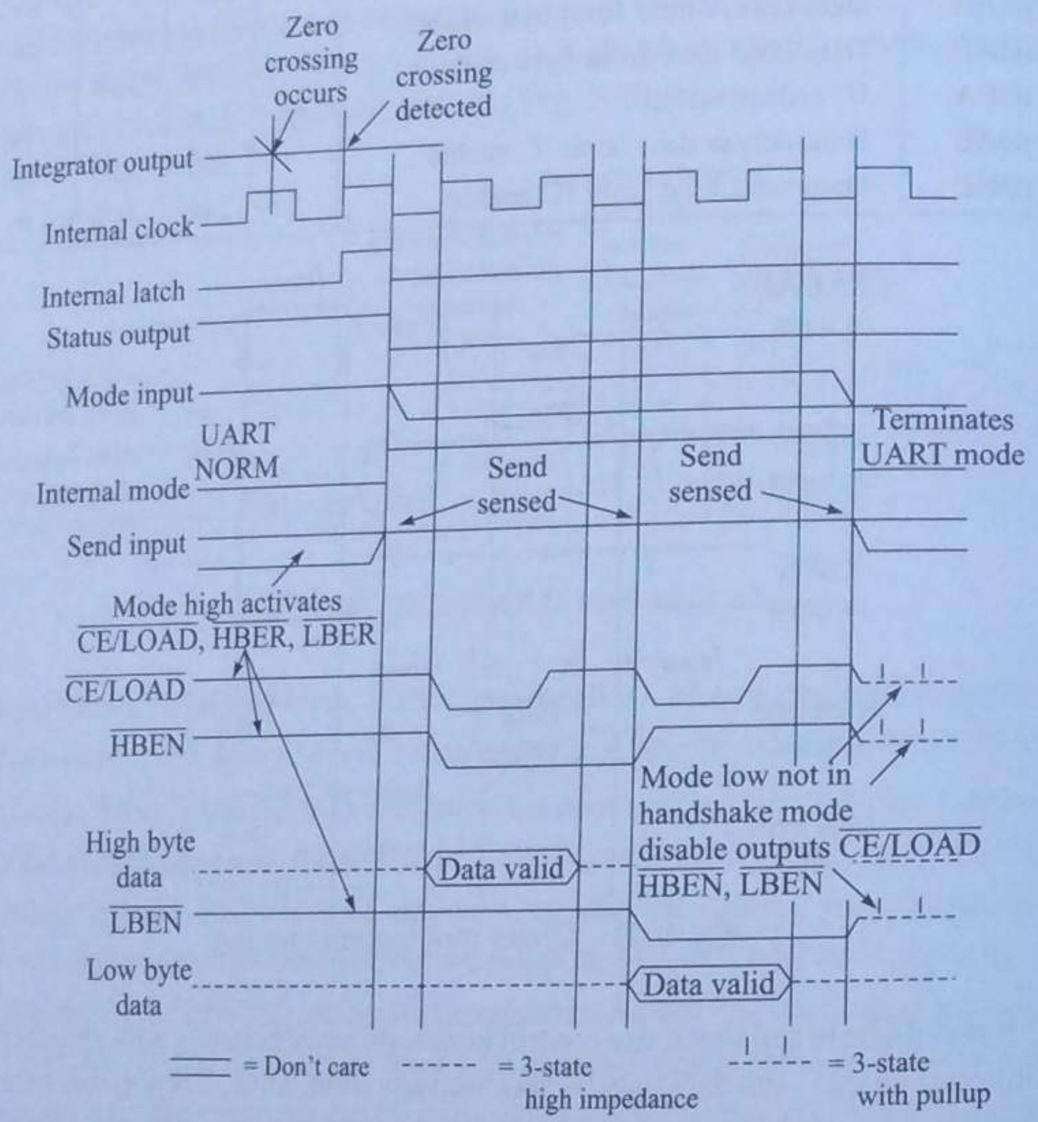


Fig. 9.42 Handshake with SEND held positive

If this pulse occurs while a new data is being stored, the entry into handshake mode is delayed until the data is stable. While the converter is in the handshake mode, the MODE input is ignored, and although conversions will still be performed, data updating will be inhibited (see Fig. 9.44) until the converter completes the output cycle and clears the handshake mode.

When the converter enters the handshake mode, or when the MODE input is high, the IC and byte enable terminals become TTL compatible outputs, which provide the control signals for the output cycle (see Figs. 9.43, 9.44, and 9.45).

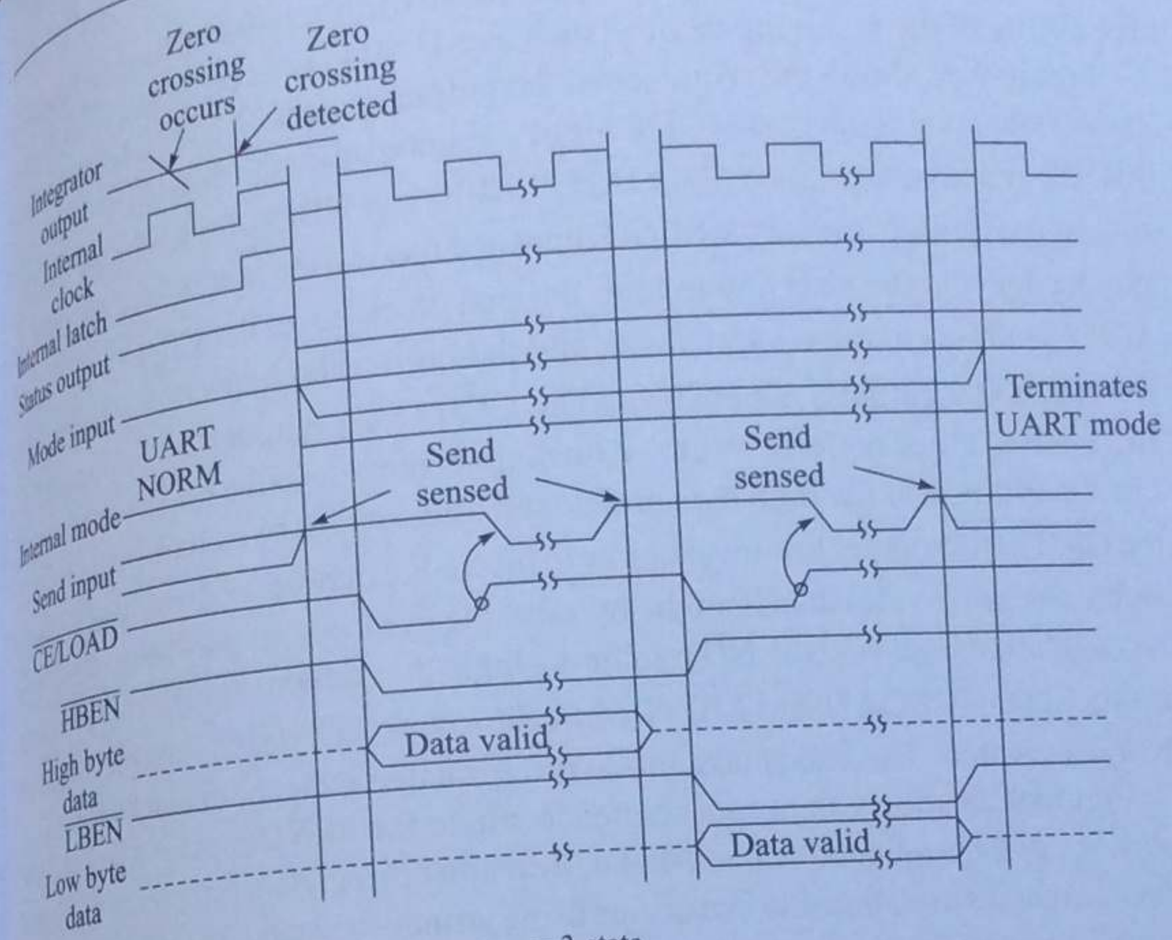


Fig 9.43 Handshake typical UART interface timing

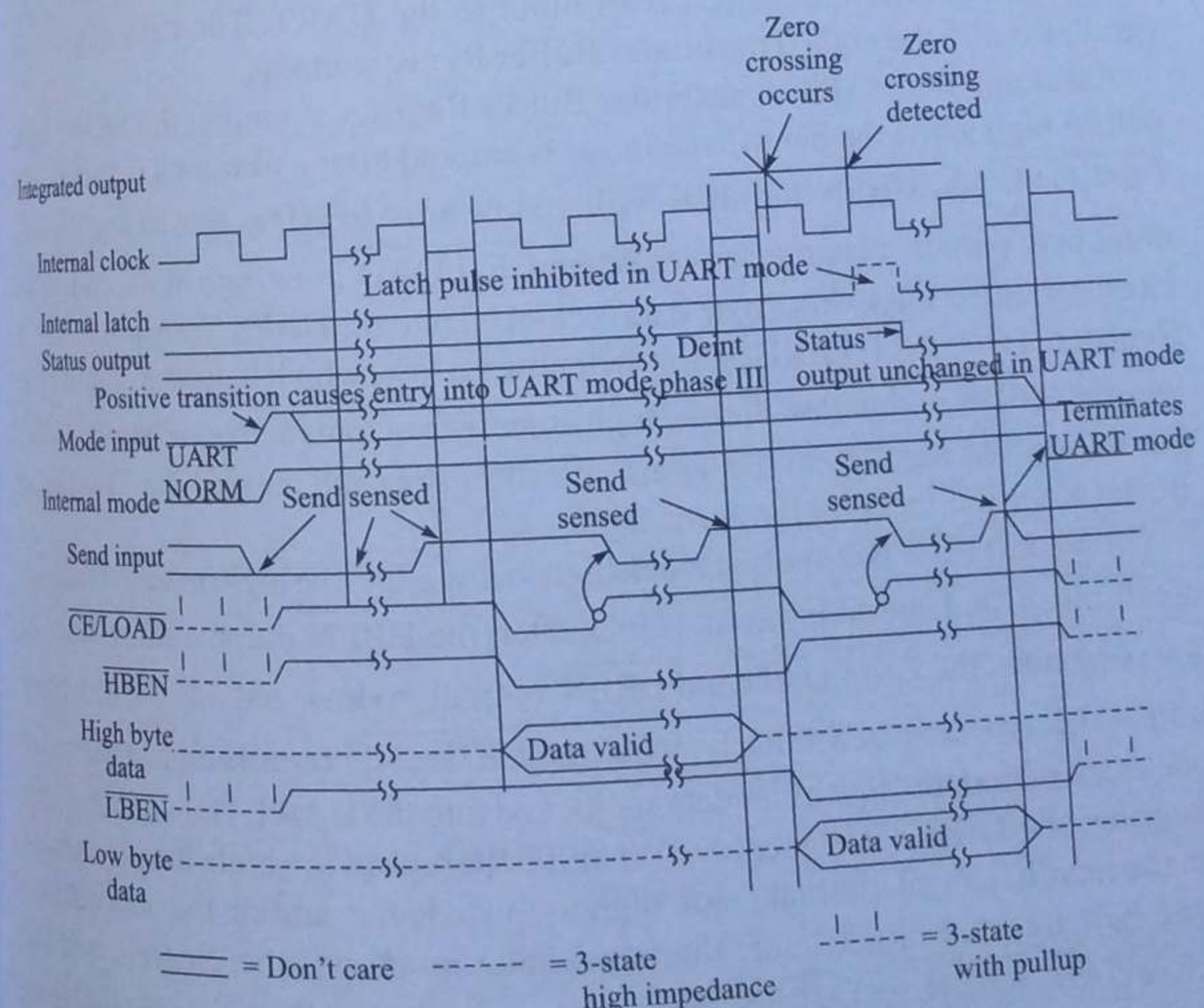


Fig. 9.44 Handshake triggered by mode

In handshake mode, the SEND input is used by the converter as an indication of the ability of the receiving device (such as a UART) to accept data.

Figure 9.42 shows the sequence of the output cycle with SEND held high. The handshake mode (internal MODE high) is entered after the data latch pulse (since MODE remains high, the $\overline{\text{CE/LOAD}}$, $\overline{\text{LBEN}}$ and $\overline{\text{HBEN}}$ terminals are active as outputs). The high level at the SEND input is sensed on the same high to low internal clock edge. On the next low to high internal clock edge, the $\overline{\text{CE/LOAD}}$ and the $\overline{\text{HBEN}}$ outputs assume a low level, and the high-order bytes (bits 9 through 12, POL, and OR outputs) are enabled. The $\overline{\text{CE/LOAD}}$ output remains low for one full internal clock period only, the data outputs remain active for one-half internal clock periods, and the high byte enable remains low for two clock periods. Thus, the $\overline{\text{CE/LOAD}}$ output low level or low to high edge may be used as a synchronizing signal to ensure valid data, and the byte enable as an output may be used as a byte identification flag. With SEND remaining high, the converter completes the output cycle using $\overline{\text{CE/LOAD}}$ and $\overline{\text{LBEN}}$ while the low order byte outputs (bits 1 through 8) are activated. The handshake mode is terminated when both bytes are sent.

Figure 9.38 shows an output sequence where the SEND input is used to delay portions of the sequence or handshake to ensure correct data transfer. This timing diagram shows a relationship that occurs using an industry standard IM6402/3 CMOS UART to interface to serial data channels. In this interface, the SEND input to the ICL7109 is driven by the $\overline{\text{TBRE}}$ (Transmitter Buffer Register Empty) output of the UART, and the $\overline{\text{CE/LOAD}}$ terminal of the ICL7109 drives the TBRL (Transmitter Buffer Register Load) input to the UART. The data outputs are paralleled into the eight Transmitter Buffer Register inputs.

Assuming the UART Transmitter Buffer Register is empty, the SEND input will be high when the handshake mode is entered after a new data is stored. The $\overline{\text{CE/LOAD}}$ and $\overline{\text{HBEN}}$ terminals will go low after SEND is sensed and the high order byte outputs become active. When $\overline{\text{CE/LOAD}}$ goes high at the end of one clock period, the high order byte data is clocked into the UART Transmitter Buffer Register. The UART TBRE output will now go low, which halts the output cycle with the $\overline{\text{HBEN}}$ output low, and the high order byte outputs active. When the UART has transferred the data to the Transmitter Register and cleared the Transmitter Buffer Register, the TBRE returns high.

On the next ICL7109 internal clock high to low edge, the high order byte outputs are disabled, and one-half internal clock later, the $\overline{\text{HBEN}}$ output returns high. At the same time, the $\overline{\text{CE/LOAD}}$ and $\overline{\text{LBEN}}$ outputs go low, and the low order byte outputs become active. Similarly, when the $\overline{\text{CE/LOAD}}$ returns high at the end of one clock period, the low order data is clocked into the UART Transmitter Buffer Register, and TBRE again goes low. When TBRE returns to a high, it will be sensed on the next ICL7109 internal clock high to low edge, disabling the data outputs. One-half internal clock later, the handshake mode will be cleared, and the $\overline{\text{CE/LOAD}}$, $\overline{\text{LBEN}}$ and $\overline{\text{HBEN}}$ terminals return high and stay active (as long as MODE stays high).

With the MODE input remaining high as in these examples, the converter will output the results of every conversion except those completed during a handshake operation. By triggering the converter into handshake mode with a low to high edge on the MODE input, handshake output sequences may be performed on demand. Figure 9.42 shows a handshake output sequence triggered by such an edge. In addition, the SEND input is shown as being low when the converter enters handshake mode. In this case, the whole output sequence is controlled by the SEND input and the sequence for the first (high order) byte is similar to the sequence for the second byte. This diagram also shows the output sequence taking longer than a conversion cycle. Note that the converter still makes conversions, with the STATUS output and RUN/HOLD input functioning normally. The only difference is that new data will not be latched when in handshake mode and is therefore lost.

Oscillator IC 7109 is provided with a versatile three-terminal oscillator to generate the internal clock. The oscillator may be overdriven, or may be operated as an RC or crystal oscillator. The OSCILLATOR SELECT input changes the internal configuration of the oscillator to optimize it for RC or crystal operation. When the OSCILLATOR SELECT input is high or left open, the input is provided with a pull-up resistor, the oscillator is configured for RC operation, and the internal clock will be of the same frequency and phase as the signal at the BUFFERED OSCILLATOR OUTPUT. The resistor and capacitor should be connected as in Fig. 9.45. The circuit will oscillate at a frequency given by $f = 0.45/RC$. A 100 kΩ resistor is recommended for useful ranges of frequency. For optimum 60 Hz line rejection, the capacitor value should be chosen such that 2048 clock period is close to an integral multiple of the 60 Hz period (but not less then 50 pF).

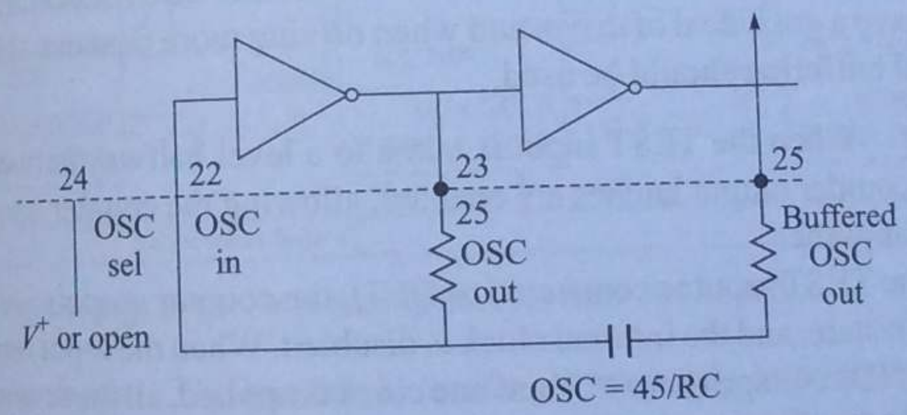


Fig. 9.45 Oscillate at a frequency given by $f = 0.45/RC$

When the OSCILLATOR SELECT input is low, a feedback device and output and input capacitors are added to the oscillator. In this configuration, as shown in Fig. 9.46, the oscillator will operate with crystals in the 1 to 50 MHz range with no external components. Using an inexpensive 3.58 MHz TV crystal, this division ratio provides an integration time given by,

$$T = (2048 \text{ clock periods}) \times \left(\frac{58}{3.58 \text{ MHz}}\right) = 33.18 \text{ ms} \qquad (9.22)$$

This time is very close to two 60 Hz periods or 33.33 ms. The error is less than one percent, which will give better than 40 dB 60 Hz rejection. The converter will operate reliably at conversion rates of up to 30 per second, which corresponds to a clock frequency of 245.8 kHz.

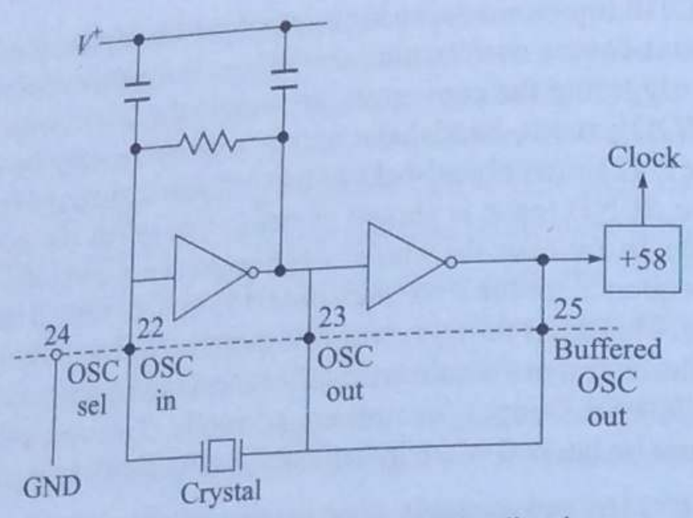


Fig. 9.46 Oscillator with feedback

If at any time, the oscillator is to be overdriven, the over-driving signal should be applied at the OSCILLATOR INPUT, and the OSCILLATOR OUTPUT should be left open. The internal clock will be of the same frequency, duty cycle and phase as the input signal when the OSCILLATOR SELECT is left open. When the OSCILLATOR SELECT is at GND, the clock will be a factor of 58 below the input frequency.

When using the ICL 7109 with the IM6403 UART, it is possible to use one 3.58 MHz crystal for both devices. The BUFFERED OSCILLATOR OUTPUT of the ICL7109 may be used to drive the OSCILLATOR INPUT of the UART, saving the need for a second crystal. However, the BUFFERED OSCILLATOR OUTPUT does not have a great deal of drive, and when driving more than one slave device, an external buffering should be used.

Test input When the TEST input is taken to a level halfway between V^+ and GND, the counter output latches are enabled, allowing the counter contents to be examined anytime.

When the TEST input is connected to GND, the counter outputs are all forced into the high state, and the internal clock is disabled. When the input returns to the 1/2 (V^+ – GND) voltage (or to V^+) and one clock is applied, all the counter outputs will be clocked to the low state. This allows easy testing of the counter and its outputs.

Interfacing

Direct mode Figure 9.47 shows some of the combinations of IC enable and byte enable control signals, which may be used when interfacing the ICL7109 with parallel data lines. The $\overline{\text{CE/LOAD}}$ input may be tied low, allowing either byte to be controlled by its own enable, as in Fig. 9.47(a). Figure 9.47(b) shows a configuration where the two byte enables are connected together. In this configuration, the $\overline{\text{CE/LOAD}}$ serves as an IC enable, and the $\overline{\text{HBEN}}$ and $\overline{\text{LBEN}}$ may be connected to GND or serve as a second IC enable. The 14 data outputs will all be enabled simultaneously. Figure 9.47(c) shows the $\overline{\text{HBEN}}$ and $\overline{\text{LBEN}}$ as flag inputs, and $\overline{\text{CE/LOAD}}$ as a master enable, which could be the READ strobe available from most microprocessors.

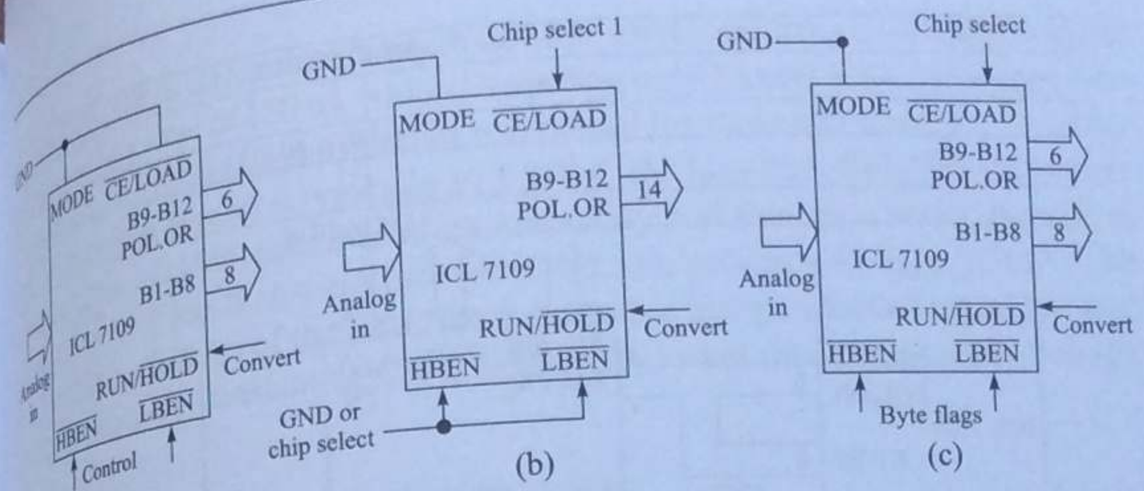


Fig. 9.47 Direct IC mode and byte enable combinations

Figure 9.48 shows an approach of interfacing several ICL7109 to a bus, ganging the $\overline{\text{HBEN}}$ and $\overline{\text{LBEN}}$ signals to several converts together, and using the $\overline{\text{CE/LOAD}}$ inputs (perhaps decoded from an address) to select the desired converter.

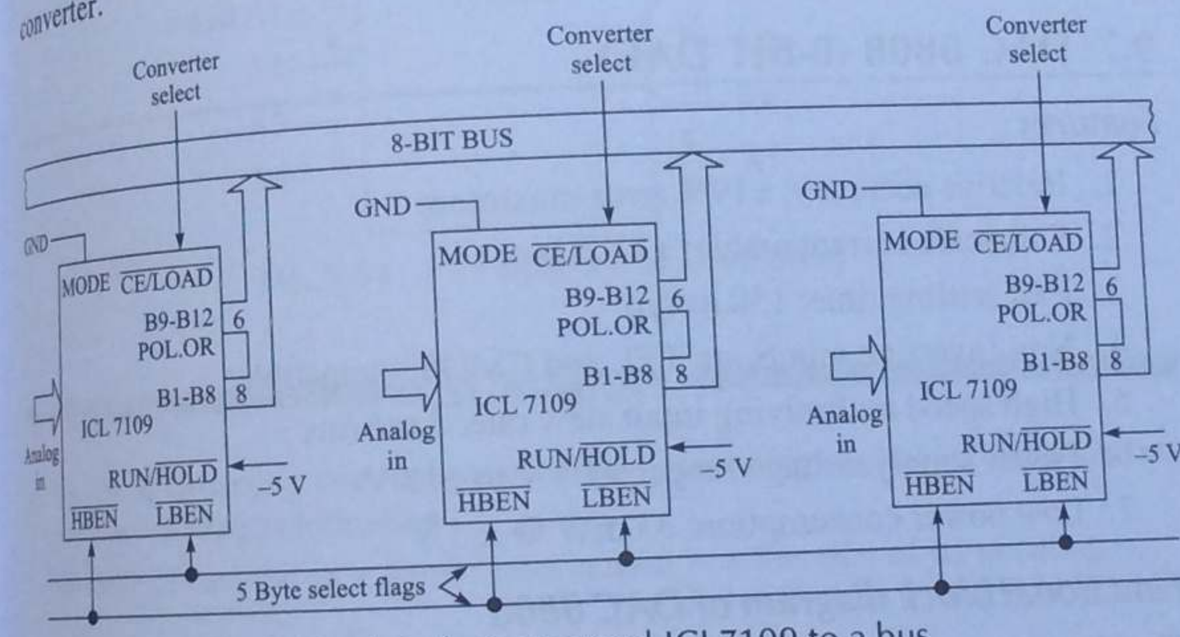


Fig. 9.48 Interfacing several ICL7109 to a bus

Handshake mode The handshake mode allows ready interface with a wide variety of external devices. For instance, external latches may be clocked by using the edge of $\overline{\text{CE / LOAD}}$ and the byte enables may be used as byte identification flags or as load enables.

Figure 9.49 shows a handshake interface to intel microprocessors, again using an 8255 PPI. The handshake operation with the 8255 is controlled by inverting its Input Buffer Full (IBF) flag to drive the SEND input to the ICL 7109, and using the $\overline{\text{CE/LOAD}}$ to drive the 8255 strobe. The internal control register of the PPI should be set in MODE 1 for the port used. If the 7109 is in handshake mode and the 8255 IBF flag is low, the next word will be strobe into the port. The strobe will cause IBF to go high (SEND goes low), which will keep the enabled byte outputs active. The PPI will generate an interrupt which when executed will result in the data being read. When the byte is read, the IBF will be reset low, which causes the ICL 7109 to sequence into the next byte. This figure shows the MODE input to the ICL 7109 connected to a control line on the PPI.

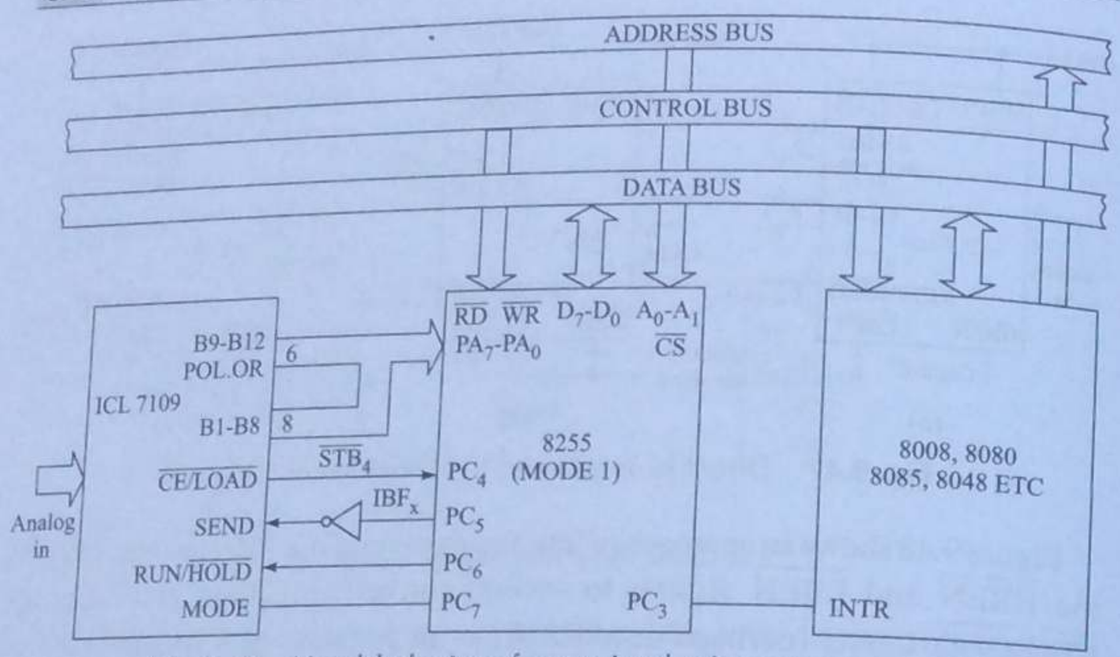


Fig. 9.49 Handshake interface to Intel microprocessors

9.7 DAC 0808 (8-BIT DAC)

Features

1. Relative accuracy: ±19% error maximum
2. Full-scale current match: ±1 LB typ.
3. Fast settling time: 150 ns typ.
4. Non-inverting inputs are TTL and CMOS compatible
5. High speed multiplying input slew rate: 8 mA/ms
6. Power supply voltage range: ±4.5 V to ±18 V
7. Low power consumption: 33 mW @ ± 5 V

Functional block diagram of DAC 0808

The block diagram of DAC 0808 is shown in Fig. 9.50 and its pin configuration is shown in Fig. 9.51.

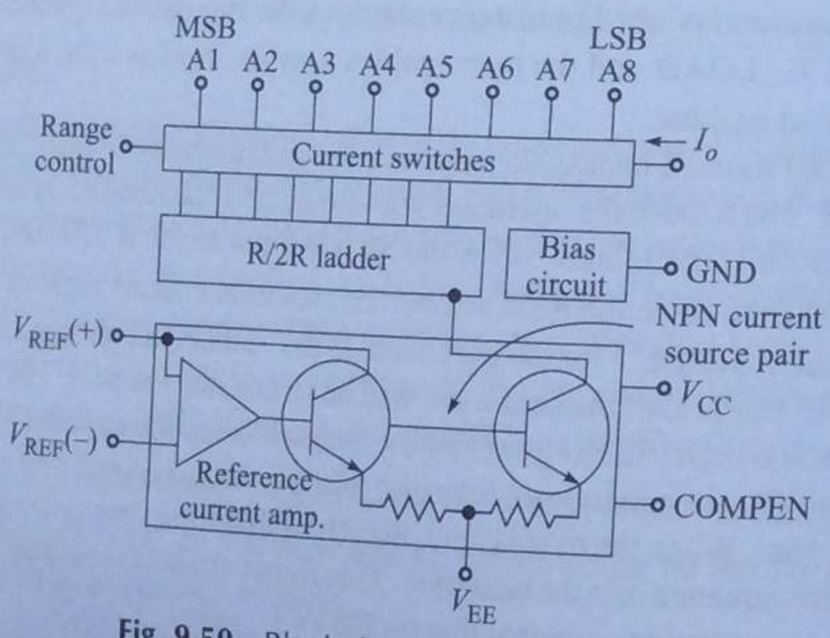


Fig. 9.50 Block diagram of DAC 0808

The DAC 0808 series is an 8-bit monolithic DAC featuring a full-scale output current settling time of 150 ns, while dissipating only 33 mW with ± 5 V supplies. No reference current (I_{REF}) trimming is required for most applications, since the full-scale output current is typically ±1 LSB of 255 I_{REF}/256. Relative accuracies of better than ±0.19% assure 8-bit monotonicity and linearity, while zero level output current of less than 4 mA provides 8-bit zero accuracy for $I_{REF} \geq 2$ mA. The power supply currents of the DAC 0808 series are independent of bit codes, and exhibit essentially constant device characteristics over the entire supply voltage range.

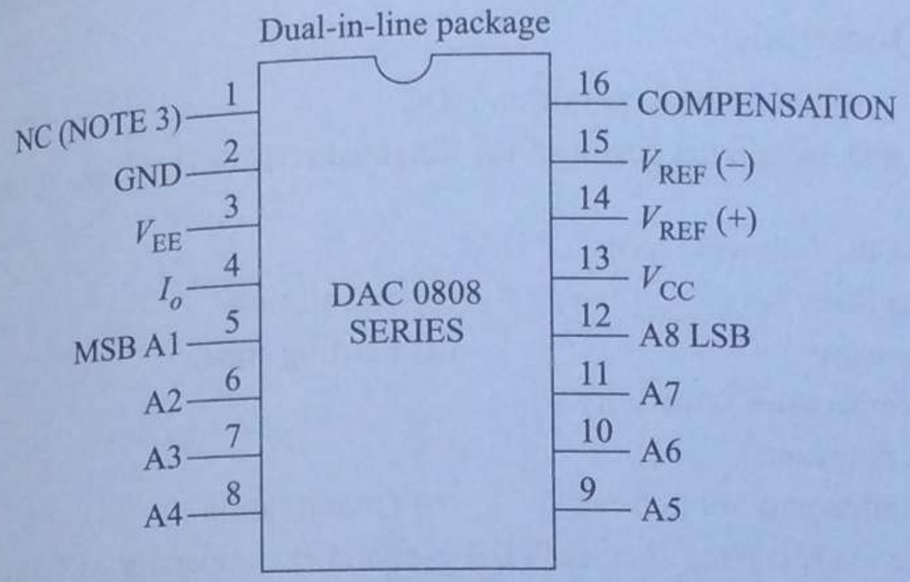


Fig. 9.51 PIN diagram of DAC 0808

SUMMARY

- The digital-to-analog conversion involves translating digital information into equivalent analog information.
- The value of each bit in a digital information is a function of its position.

1. The value of LSB is $\frac{V_m}{2^n}$
2. The value of second LSB is $\frac{2^1 \cdot V_m}{2^n}$
3. The value of *N*th LSB is $\frac{2^{N-1} \cdot V_m}{2^n}$

where n is the number of bits of a digital signal and V_m is the maximum amplitude. The output analog voltage V_A is

$$V_A = \frac{\left[V_{D0}2^0 + V_{D1}2^1 + V_{D2}2^2 \ldots V_{D(n-1)}2^{n-1}\right]}{2^n}$$

where V_{D0}, V_{D1}, V_{D2}, ..., $V_{D(n-1)}$ are the digital inputs of n-bit DAC.

- The circuits of DAC are two types: (a) resistor divider DAC, and (b) R/2R ladder network DAC.

- The characteristics of a DAC which are generally specified by the designers are (a) Accuracy, (b) Resolution, (c) Linearity, (d) Settling time, and (e) Temperature sensitivity.
- In case of an ADC, the input analog voltage can have any value in a range and the digital output can have only 2^N discrete values for an N-bit ADC.
- Commonly used ADCs are: (a) Parallel comparator ADC, (b) Successive approximation ADC, and (c) Dual slope ADC.

EXERCISE

Review Questions

1. Explain the basic concepts of an ADC.
2. Draw and explain the circuit of (a) weighted resistor DAC, and (b) R-$2R$ ladder DAC.
3. Define the following terms of DAC.
   (a) Accuracy (b) Resolution
   (c) Linearity (d) Settling time
   (e) Temperature sensitivity
4. Write notes on
   (a) Sample-and-hold circuit (b) Quantization
5. Draw and explain the circuit of (a) 4-bit parallel comparator ADC (b) Successive approximation ADC, and (c) Dual slope ADC.
6. Calculate the values of LSB, MSB, and full-scale output for an 8-bit DAC for 0–12 V range.
7. Find the output voltage for a 4-bit ladder, having the digital inputs
   (a) 1010 (b) 0110 (c) 1001 (d) Full-scale output
   Assume logic 0 = 0 V and logic 1 = 12 V.

Multiple-Choice Questions

1. Among the following, the slowest ADC is
   (a) parallel-comparator (i.e. Flash) type
   (b) successive-approximation type
   (c) integrating type
   (d) counting type
2. The noumber of comparisons carried out in a 4-bit Flash type ADC is
   (a) 16 (b) 15 (c) 4 (d) 4
3. A 10-bit converter is used to digitize an analog signal in the 0 to 5V range. The maximum peak-to-peak ripple voltage that can be allowed in the dc supply voltage is
   (a) nearly 100 mV (b) nearly 50 mV
   (c) nearly 25 mV (d) nearly 5 mV
4. The number of comparators in a parallel conversion type 8-bit ADC is
   (a) 8 (b) 16 (c) 255 (d) 256

5. An analog voltage in the range of 0 to 8 V is divided in eight equal intervals for conversion to 3-bit digital output. The maximum quantization error is
   (a) 0 V (b) 0.5 V (c) 1 V (d) 2 V
6. The resolution of a DAC is approximately 0.4% of its full-scale range. It is
   (a) 8-bit converter (b) 10-bit converter
   (c) 12-bit converter (d) 16-bit converter
7. A 12-bit ADC is operating with a 1-microsec clock period and the total conversion time is seen to be 14 microsec. The ADC must be of
   (a) Parallel-comparator type (b) Counting type
   (c) Integrating type (d) Successive-approximation type
8. Dual slope integration type ADC provides
   (a) Higher speed as compared to all other types of ADCs
   (b) Very good accuracy without putting extreme requirement on component stability
   (c) Poor rejections of supply hum
   (d) Better resolution compared to all other types of ADCs for the same number of bits
9. In a 4-bit weighted resistor DAC, the resistor value corresponding to LSB is 32 kΩ. The resistor value corresponding to MSB will be
   (a) 32 K (b) 16 K (c) 8 K (d) 4 K
10. The resolution of a 12-bit DAC using a binary ladder with 10 V as the full-scale output will be
    (a) 2.44 mV (b) 3.50 mV (c) 4.32 mV (d) 5.12 mV
11. A DAC uses a ladder of 10 V full-scale output. The number of bits required of its input for a resolution of 5 mV will be
    (a) 7 (b) 8 (c) 15 (d) 16
12. For successive approximation with N output bits, the number of clocks required is
    (a) $N + 2$ or N (b) $2N + 1$ (c) 2^N (d) $2^N - 1$
13. Which of the following is a D/A conversion technique?
    (a) Successive approximation (b) Weighted resistor
    (c) Dual slope technique (d) Single slope technique

Answers

1. (d)	2. (b)	3. (d)	4. (c)
5. (b)	6. (a)	7. (d)	8. (b)
9. (d)	10. (a)	11. (d)	12. (a)
13. (b)			

Chapter

10

SEMICONDUCTOR MEMORY DEVICES

Chapter Outline

- Study of memory organization
- Classification of semiconductor memory
- Characteristics of memory devices

10.1 INTRODUCTION

Memory is an important part of computers and other microprocessor-based systems. The data to be processed by a program as well as the code of the program are stored in memory. Hence, these systems require memory facilities to store permanent as well as temporary data and code. The speed of a system is not only a function of microprocessor speed but also a function of how efficient the memories are. Hence, high-speed memories are used in systems.

Traditionally, punch cards and magnetic tapes were used to store data. These devices have low speed, less storage density, and bulky circuits. At present, digital technology has progressed remarkably from vacuum-tube to integrated circuits and from magnetic tape to semiconducting memories. Semiconducting memory, on the other hand, is very compact, can be accessed at very high speeds, and it has high storage density.

Memory is classified into two types, *main memory* and *secondary memory*. The main memory is further classified as *read only memory* and *read-write memory*. This chapter includes the study of memory organization and operations, and the characteristics and classifications of memories.

10.2 MEMORY ORGANIZATION

Memory is a storage device. Semiconductor memories store the data in terms of 0 and 1. It is also referred as a *memory cell*. A group of cells forms the register and a set of registers forms a memory. Each register in a memory is a storage location, which can be recognized by a specific address during the read and write operations. The register is a group of flip-flops. The size of the register is nothing but the number of flip-flops or cells contained in the register, which define the size of the words. Most memory ICs have eight cells in each register and the number of registers is 2^n, where n is an integer.

Memories are always defined as $M \times N$, where M is the number of locations (register) and N is the size of registers. Each location has a unique address and it is in binary. For M memory locations, n address lines are required to enable one location where $M = 2^n$. These n address lines are decoded by the decoder and one of the memory locations is selected. The basic block diagram of $M \times N$ memory is shown in Fig.10.1.

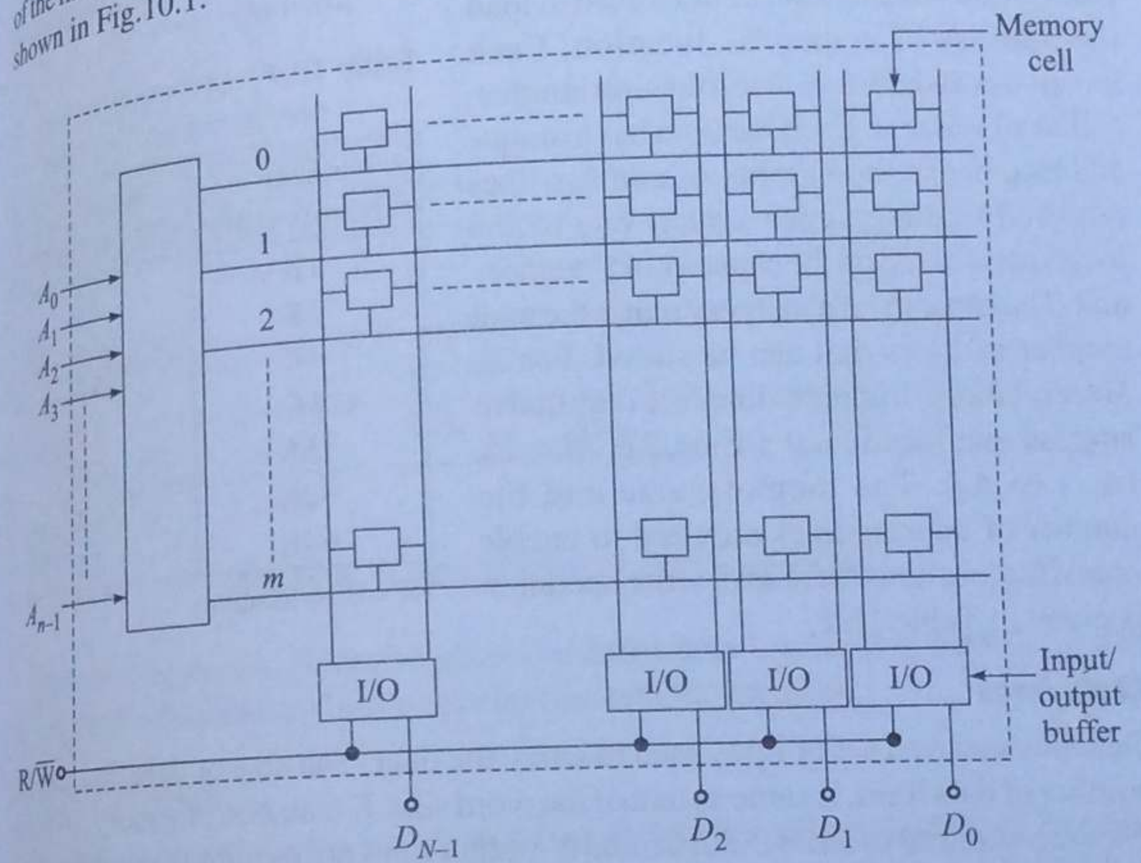


Fig. 10.1 Block diagram of M×N memory

The memory has $M \times N$ memory cells, A_{n-1} to A_0 are the address lines, D_{N-1} to D_0 are the data lines, and $\overline{RD}$, $\overline{WR}$ are the control signals. The data lines are used to read or write the N-bits of data simultaneously. For example, a 1K×8 memory IC has 1024×8 = 8192 basic cells, 10 address lines A_9 to A_0, and 8 data lines D_7 to D_0.

A memory IC has $\overline{RD}$ and $\overline{WR}$ control signals. The $\overline{RD}$ control signal enables the output buffer and data from the selected memory location load onto output data lines. The $\overline{WR}$ control signal enables the input buffer and data from the input data lines load onto selected memory locations. Most of the memory ICs have a common data bus for input and output, as shown in Fig. 10.1. When this control signal goes low, the input buffer is enabled and the data from the data lines load onto selected memory locations. When the control signal goes high, the output buffer is enabled and the data from the selected memory location load onto data lines.

10.3 FUNCTIONAL DIAGRAM OF MEMORY

A memory unit is a collection of storage cells with associated circuits needed to transfer data in and out of the device. The data is stored in memory in terms of groups of binary bits called *words*. In a computer system, the microprocessor reads

or writes the data from or to the memory. For read/write operation, we require address bus, data bus, and control signals. Figure 10.2 shows the functional diagram of a memory.

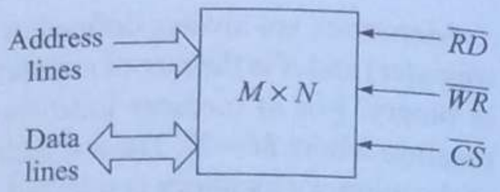


Fig. 10.2 Functional diagram of memory

Address lines

Address bus means a set of lines used to load the address of a specific location. Each location is assigned an identification number, called an *address*. Each location has a unique address, hence the number of address lines required to address (or select) one of the locations is based on the capacity of a memory unit. The capacity of a memory unit is the total number of bytes that can be stored. For an $M \times N$ memory, n address lines are required to address one location at a time, i.e. $M = 2^n$, (A_{n-1} to A_0). The memory size and the number of address lines required to enable one of the locations for read or write operation is given in Table 10.1.

Table 10.1 Memory size and number of address lines

Number of locations(M)	Number of address lines(n)
1K	10
2K	11
4K	12
8K	13
16K	14
32K	15
64K	16

1K =1024 locations

Data lines

Data bus means a set of lines used to carry the data. The size of data bus, i.e. the number of data lines, is same as that of the word size. For an $M \times N$ memory, N-input lines are required to write the data, and N-output lines are required to read the data. Most of the memory ICs have a bidirectional single data bus, which is capable to load/write as well as read the data to or from the memory cells. Most of the ICs have eight data lines, which is collectively referred to as the data bus.

Chip select signal ($\overline{CS}$)

It is an active low input signal. When this signal is connected to ground, the chip is selected (enable) and read/write operation is allowed, otherwise the chip is disabled.

Read and write signals ($\overline{RD}$ and $\overline{WR}$)

These are active low input signals. When $\overline{RD}$ goes low, the output buffer is enabled and the data of selected location gets loaded on the data bus. When $\overline{WR}$ goes low, the data on the data bus is loaded onto the selected location. Some memory ICs have single control input R/$\overline{W}$. It is low to write the data and high to read the data.

10.4 MEMORY OPERATIONS

Figure 10.3 shows the organization of a 16×8 memory chip. This memory organization has 16 locations, each of which stores 8 data bits. It has four address lines, eight data lines, an active low chip select signal, and active low R/W control signals.

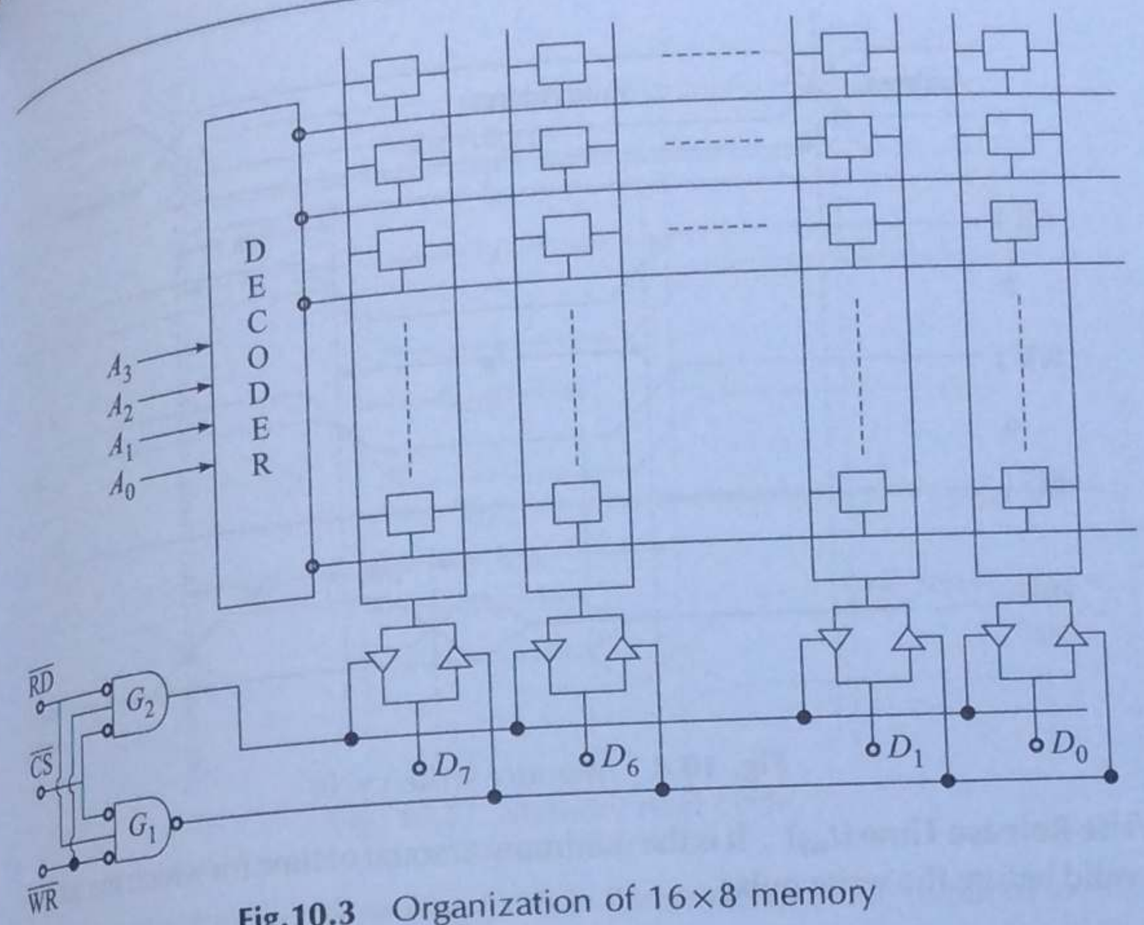


Fig.10.3 Organization of 16×8 memory

The external device can both read and write the data. Let us study how an external device performs read and write operations with the memory.

Write operation

The write operation is nothing but writing the data to a selected memory location. During the write operation, the address of a location is loaded on address lines A_0 to A_3, which is decoded by the decoder to enable the memory location and the data is loaded by the microprocessor over the data lines. The $\overline{CS}$ and $\overline{WR}$ signals go low, and $\overline{RD}$ signal is high. Since $\overline{WR}$ is low, $\overline{CS}$ is low and $\overline{RD}$ is high, the output of gate G_1 is high, input buffer is enabled and the data which is present on the data lines is loaded into the selected memory location. For write operation, the following sequence of operations has to be performed:

(a) The address of a desired memory location is loaded on address lines first.
(b) The chip select signal goes low.
(c) The word to be stored is loaded on data input lines.
(d) Write command signal is applied to the write control input terminal ($\overline{WR} = 0$, $\overline{RD} = 1$), and the data is written at that memory location.

The memory write cycle is shown in Fig. 10.4.

Write cycle time (t_{wc}) It is the minimum amount of time for which the valid address is present for writing the data in memory location. It is in the order of 200 ns.

Write pulse time (t_w) It is the minimum duration of the write pulse. It is in the order of 120 ns.

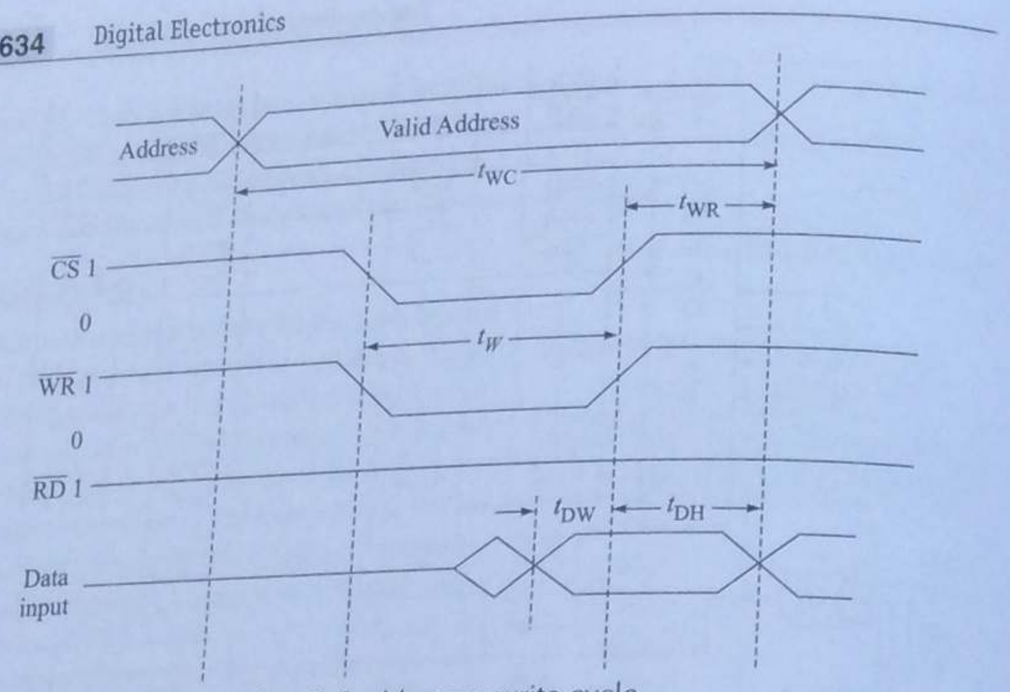


Fig. 10.4 Memory write cycle

Write Release Time (t_{wr}) It is the minimum amount of time for which the address is valid before the write pulse.

Data setup time (t_{DW}) It is the minimum amount of time for which the data is valid before the write pulse ends. It is in the order of 120 ns.

Data hold time (t_{DH}) It is the minimum amount of time for which the data is valid after the write pulse ends.

Read operation

The read operation is nothing but reading the data from a selected memory location. During the read operation, the address of a location is loaded on address lines A_0 to A_3, which is decoded by the decoder to enable the memory location. The $\overline{CS}$ and $\overline{RD}$ signals go low, and $\overline{WR}$ signal is high. Since $\overline{WR}$ signal is high, $\overline{CS}$ is low and $\overline{RD}$ is low, the output of gate G_2 is high, output buffer is enabled and the data from the selected memory location will be loaded on data lines. For read operation, the following sequence of operations has to be performed:

(a) The address of the desired memory location is loaded on address lines.
(b) The chip select signal goes low.
(c) Read command signal is applied to the read control input terminal ($\overline{WR} = 1$, $\overline{RD} = 0$), and the data is loaded on data lines.

The memory read cycle is shown in Fig. 10.5.

Read cycle time (t_{RC}) It is the minimum amount of time for which the valid address is present for reading the data from a memory location, as shown in Fig. 10.5. It is in the order of 200 ns.

Access time (t_A) It is the time required to access the data from a memory location. It is the maximum time from the start of valid address of the read cycle to the availability of valid data at the data outputs, as shown in Fig. 10.5. It is in the order of 200 ns.

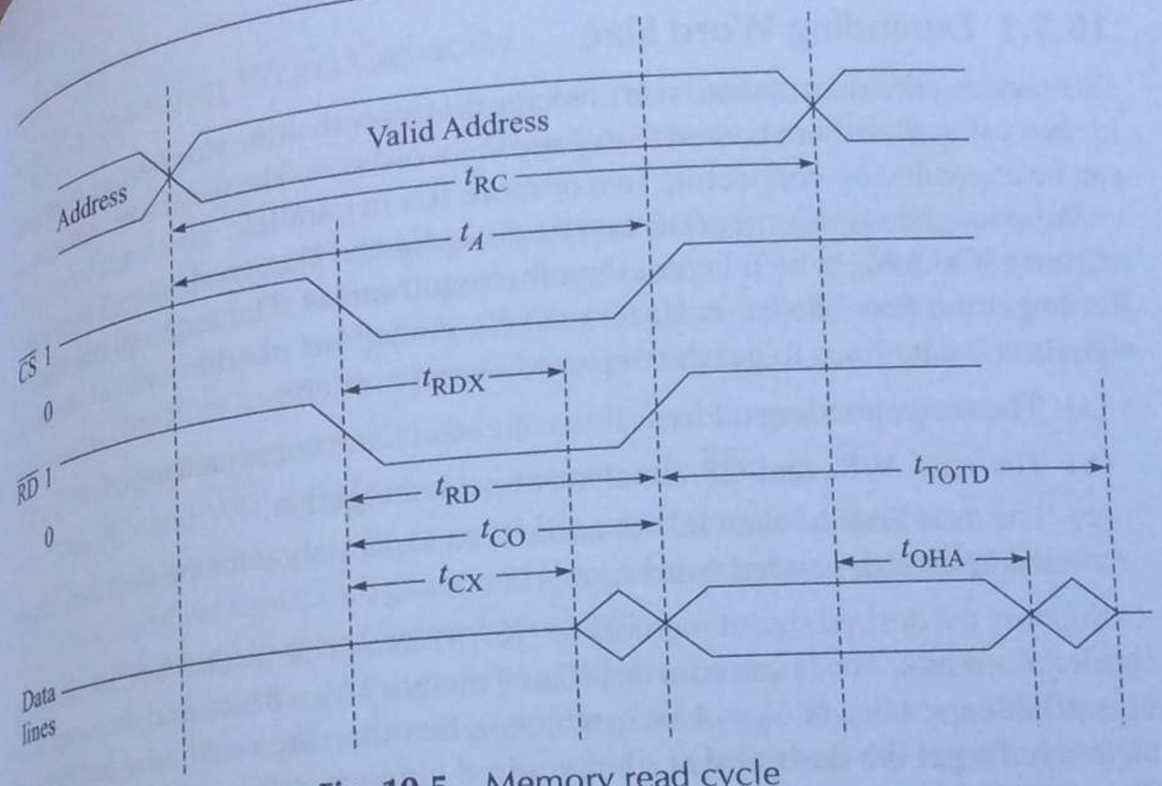


Fig. 10.5 Memory read cycle

Read to output active time (t_{RDX}) It is the minimum time required to enable the output buffer after the beginning of the read pulse. It is in the order of 20 ns.

Read to output valid time (t_{RD}) It is the maximum delay time between the start of the read pulse and the availability of valid data at the data outputs. It is in the order of 70 ns.

IC select to output valid time (t_{CO}) It is the maximum time delay between the start of IC select signal and the availability of valid data outputs. It is in the order of 70 ns.

IC select to output active time (t_{CX}) It is the maximum time delay between the start of IC select signal and the output buffer coming to active state. It is in the order of 20 ns.

Output tristate from read (t_{TOTD}) It is the maximum time delay between the end of read pulse and the output buffers going to tristate. It is in the order of 60 ns.

Data hold time (t_{OHA}) It is the minimum time for which the valid data is available at the data output after the address ends. It is in the order of 50 ns.

10.5 EXPANDING MEMORY SIZE

The required memory capacity is based on the application and the type of microprocessor. For example, 8085 can access 64K × 8, 8086 can access 1M × 16, and so on. In many applications, a single available memory IC cannot satisfy the required memory capacity. Word size or the number of locations or both may not be sufficient. In such applications, by connecting several ICs in series and/or parallel, we can expand the size of memory.

10.5.1 Expanding Word Size

When the requirement of word size of memory is more than the word size of available memory ICs, then there is need to expand the word size. The word size of memory can be expanded by connecting two or more ICs in parallel.

Suppose the requirement of word size is N_R and the word size of available memory ICs is N_A, which is less than the requirement. The required number of ICs to get the desired size is N_R / N_A. The connections of address lines, control signals and data lines to get the expanded word size are:

(a) The corresponding address lines of each IC are connected together.

(b) The $\overline{RD}$, $\overline{WR}$, and $\overline{CS}$ are also connected together.

(c) The data lines of each IC are connected separately with the data bus to get the desired expanded word size.

Suppose the desired size of memory is 1K×8 bits and the available IC of memory size is 1K×4 bits. The required word size of memory $N_R = 8$ bits, and the word size of available memory is $N_A = 4$ bits, which is less than the required word size of memory. To get the desired size, the required number of ICs is $N_R/N_A = 8/4 = 2$. Figure 10.6 shows a 1K×8 memory using two 1K×4 ICs.

A 1K×4 memory has 10 address lines (M=1K=1024=2^{10}), A_9 to A_0, and four data lines, D_3 to D_0.

(a) The corresponding address lines of ICs are connected together, i.e. A_9–A_0 of IC_1 and A_9–A_0 of IC_2 are connected together.

(b) The $\overline{RD}$, $\overline{WR}$, and $\overline{CS}$ are also connected together.

(c) The data lines of IC_1 are connected to D_3 to D_0 data lines of a system bus and the data lines of IC_2 are connected to D_7 to D_4 data lines of a system bus to get the desired expanded size of words.

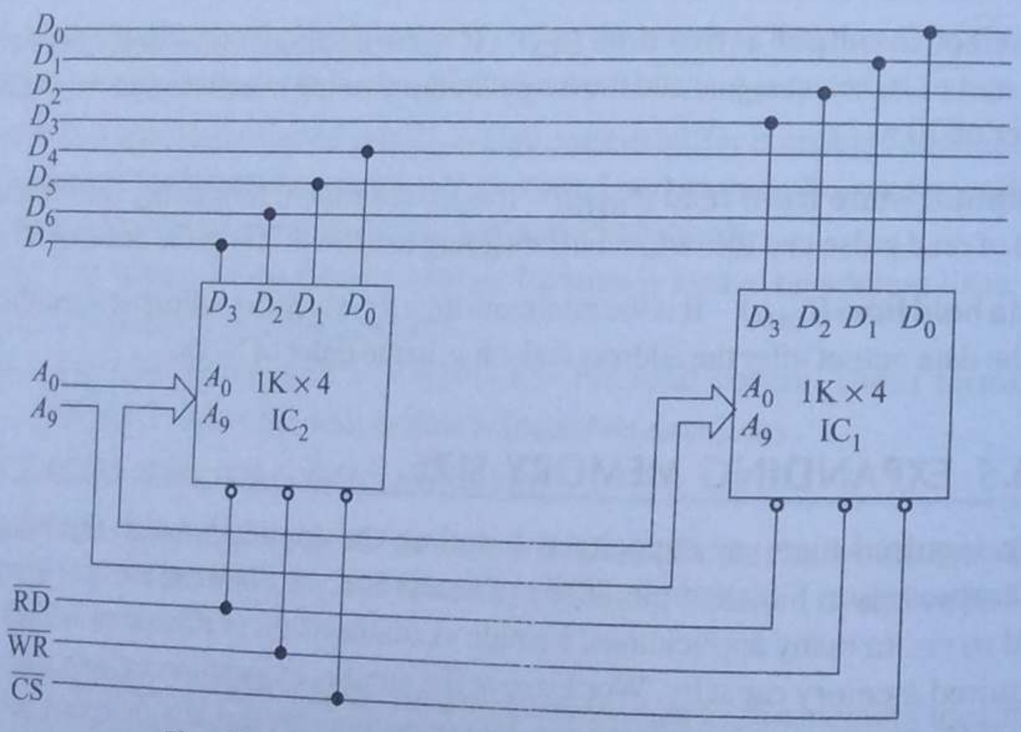


Fig. 10.6 1K×8 memory using two 1K×4 ICs

10.5.2 Expanding Word Capacity

The word capacity of memory is nothing but the number of locations (set of registers). Connecting two or more memory ICs in series can increase the word capacity of the memory.

Suppose the requirement of word capacity is M_R and the word capacity of available memory ICs is M_A, which is less than the requirement. The required number of ICs to get the desired size is M_R / M_A. The connections of address lines, control signals, and data lines to get the expanded capacity are:

(a) The corresponding address lines of each IC are connected together.

(b) The $\overline{RD}$ and $\overline{WR}$ signals are also connected together.

(c) The corresponding data lines of all ICs are connected together.

(d) The chip select signals are generated using a decoder, and separate chip select signals are used to select individual chips.

Suppose the desired size of memory is 2K×8 bits and the available memory size is 1K×8 bits. The required word capacity of memory, M_R= 2K, and the word capacity of available memory, M_A = 1K, which is less than the required word capacity. To get the desired size of memory, the required number of ICs is: M_R/M_A = 2K/1K = 2. Figure 10.7 shows a 2K×8 memory using two 1K×8 ICs.

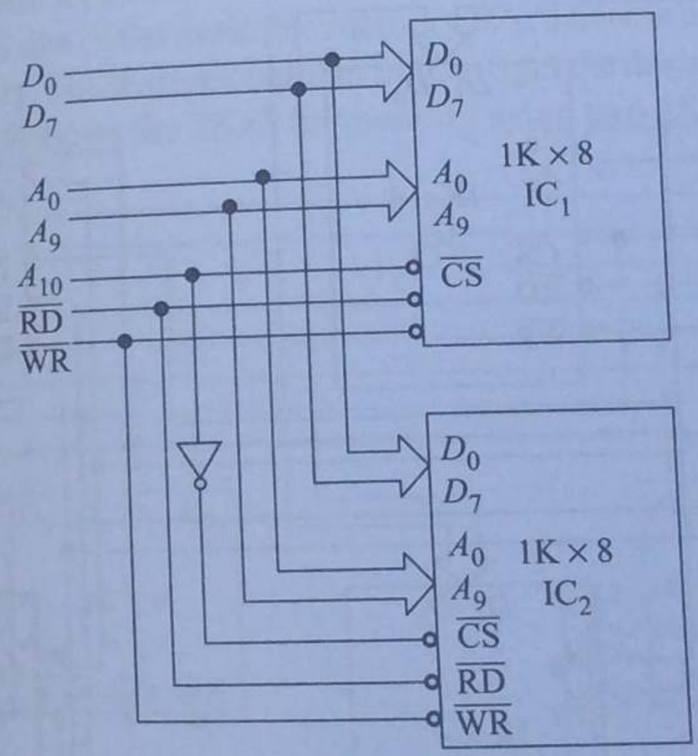


Fig. 10.7 2K×8 memory using two 1K×8 ICs

A 1K×8 memory IC has 10 (1024 = 2^n, n =10) address lines, A_0– A_9, and eight data lines, D_7 to D_0.

(a) The corresponding eight data lines, D_7–D_0 of IC_1 and D_7–D_0 of IC_2, are connected to get the common data lines D_7–D_0.

(b) The $\overline{RD}$, $\overline{WR}$ of IC_1 and IC_2 are connected to get the common $\overline{RD}$ and $\overline{WR}$ control signals.

(c) The corresponding ten address lines, A_0–A_9 of IC_1 and A_0–A_9 of IC_2, are connected together to get A_0–A_9.

(d) Address line A_{10} is directly connected to the chip select signal of IC_1, after the inverter is connected to the chip select signal of IC_2.

When address line A_{10} goes low, IC_1 is enabled and IC_2 is disabled, the address range of IC_1 is 000,00000,0000 to 011,1111,1111. When A_{10} is 1, IC_1 is disabled and IC_2 is enabled, the address range of IC_2 is 100,0000,0000 to 111,1111,1111 and the address range of 2K×8 bits memory is 000,0000,0000 to 111,1111,1111.

10.5.3 Expanding Word Size and Word Capacity

The word size of memory is increased by connecting two or more ICs in parallel and the word capacity is increased by connecting two or more ICs in series.

Suppose the desired memory size is 2K×8 bits and the available memory chip is 1K×4 bits. To construct a 2K×8 bit memory, two ICs of 1K×4 size are connected in parallel or two ICs of 1K×8 size are connected in series. Hence, a 2K×8 memory is obtained by using four 1K×4 memory chips, as shown in Fig.10.8.

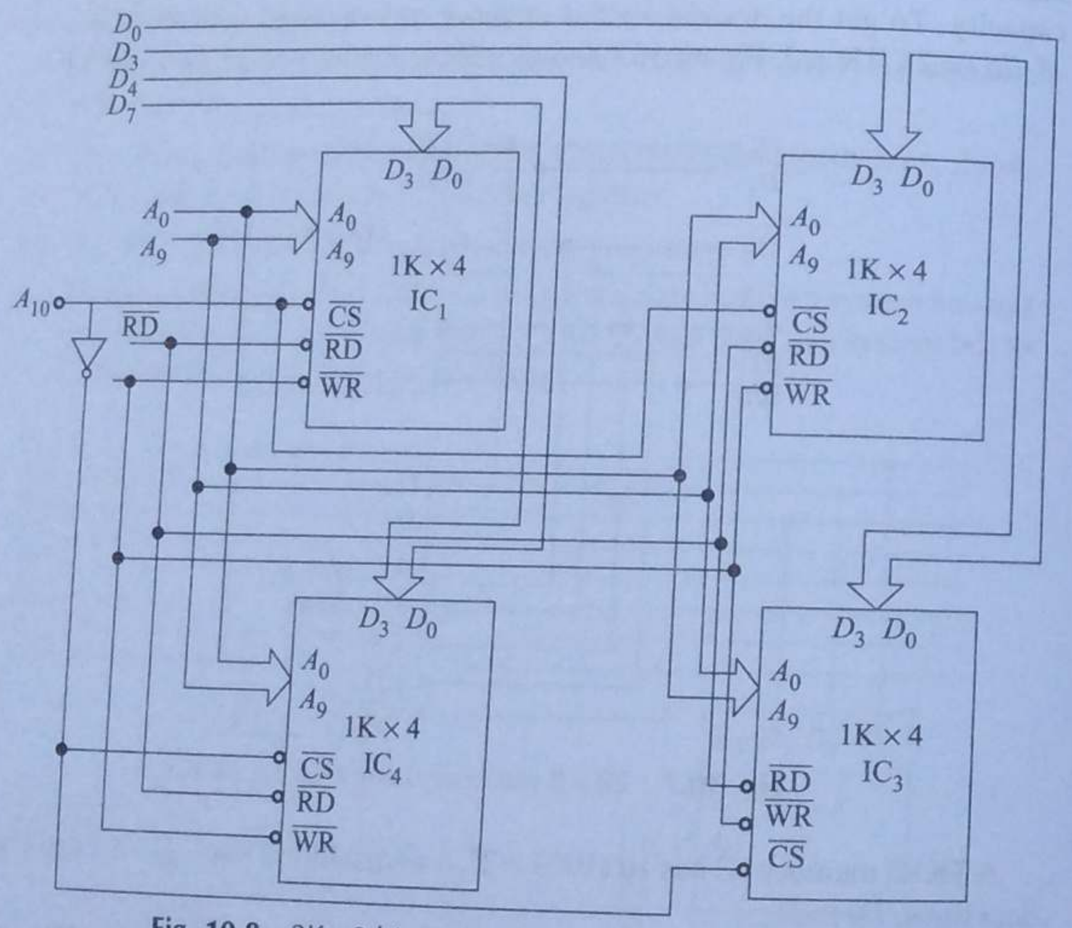


Fig. 10.8 2K×8 bit memory using four 1K×4 ICs

A 1K×4 bit memory IC has 10 address lines (M=1, K=1024=2^{10}), A_9 to A_0, and four data lines, D_3 to D_0.

(a) The four data lines of IC_1 and IC_3 are connected to D_3–D_0 data lines of the system bus and the four data lines of IC_2 and IC_4 are connected to D_7–D_4 data lines of the system bus.

(b) $\overline{RD}$, $\overline{WR}$ signals of IC_1, IC_2, IC_3, and IC_4 are connected to get the common $\overline{RD}$ and $\overline{WR}$ control signals.

(c) The corresponding ten address lines (A_9–A_0) of IC_1, IC_2, IC_3, and IC_4 are connected together to get the common address lines (A_9–A_0).

(d) Address line A_{10} is directly connected to the chip select signal of IC_1 and IC_2, after the inverter is connected to the chip select signal of IC_3 and IC_4.

When the address line A_{10} goes low, IC_1 and IC_2 are enabled, IC_3 and IC_4 are disabled, D_3 to D_0 data lines of IC_1 act as D_3 to D_0, and D_3 to D_0 data lines of IC_2 act as D_7 to D_4 data lines.

When the address line A_{10} goes high, IC_1 and IC_2 are disabled, IC_3 and IC_4 are enabled, D_3 to D_0 data lines of IC_3 act as D_3 to D_0, and D_3 to D_0 data lines of IC_4 act as D_7 to D_4 data lines.

Example 10.1 Design a 2K×8 bit memory chip using 2K×4 chips.

Solution

The desired memory size is 2K×8 bits and the available memory size 2K×4 bits. The required word capacity of memory is M_R = 2K and the word size of memory is N_R=8 bits. The word capacity of available memory ICs is M_A = 2K and the word size of available memory ICs is N_A = 4 bits. The required word size of memory is more than the word size of the available memory ICs. There is need to expand the word size of memory. The required number of ICs to get the desired size is N_R / N_A= 8/4 = 2. Figure 10.9 shows the 2K×8 bit memory using two 2K×4 chips.

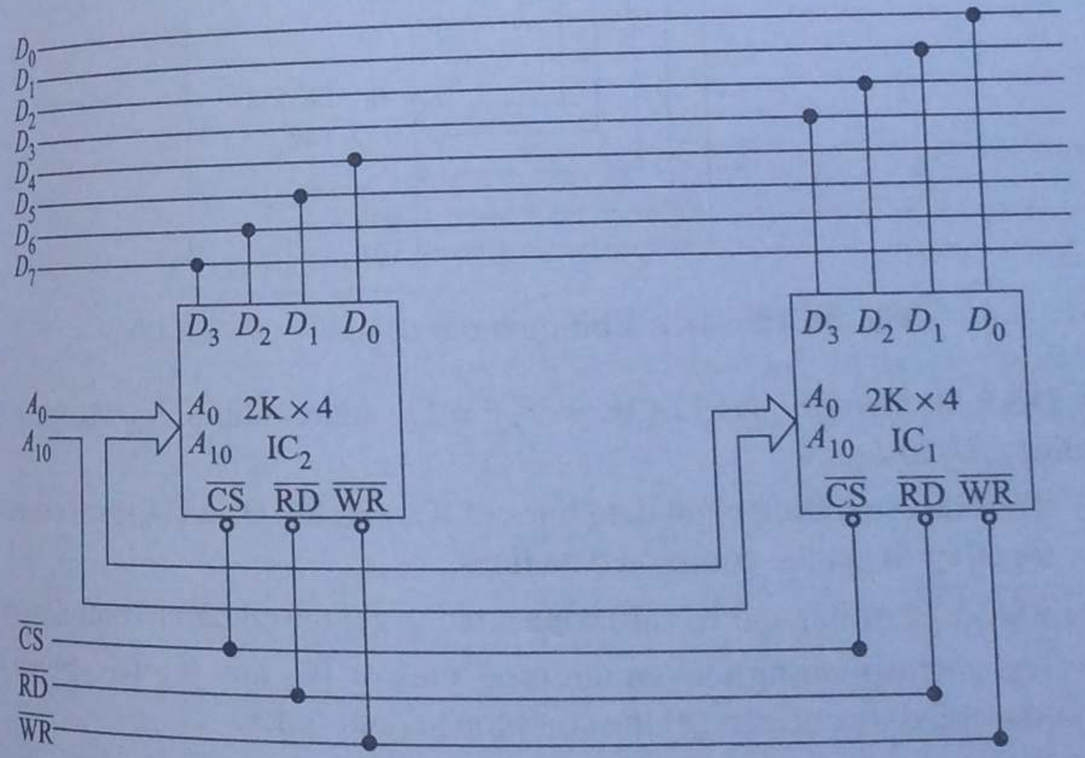


Fig. 10.9 2K×8 bit memory using two 2K×4 chips

A 2K×4 bit memory has 11 address lines (M = 2K = 2^{11}), A_{10} to A_0, and four data lines, D_3 to D_0.

(a) The corresponding address lines of the two ICs are connected together, i.e. A_{10}–A_0 of IC_1 and A_{10}–A_0 of IC_2 are connected together.

(b) The $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$ are connected together.
(c) The data lines of IC_1 and IC_2 are connected to D_3 to D_0 and D_7 to D_4 data lines of a system bus, respectively, to get the desired expanded word size.

Example 10.2 Design a 4K×8 memory chip using 2K×8 chips.

Solution
The desired size of memory is 4K×8 bits and the available memory size 2K×8 bits. The required word capacity of memory is $M_R = 4K$ and the word size of memory is $N_R = 8$ bits. The word capacity of available memory ICs is $M_A = 2K$ and the word size of available memory ICs is $N_A = 8$ bits. The required word capacity of memory is more than the word capacity of available memory ICs. There is need to expand the word capacity of memory. The required number of ICs to get the desired size is $M_R / M_A = 8/4 = 2$. Figure 10.10 shows the 4K×8 bits memory using 2K×8 bits.

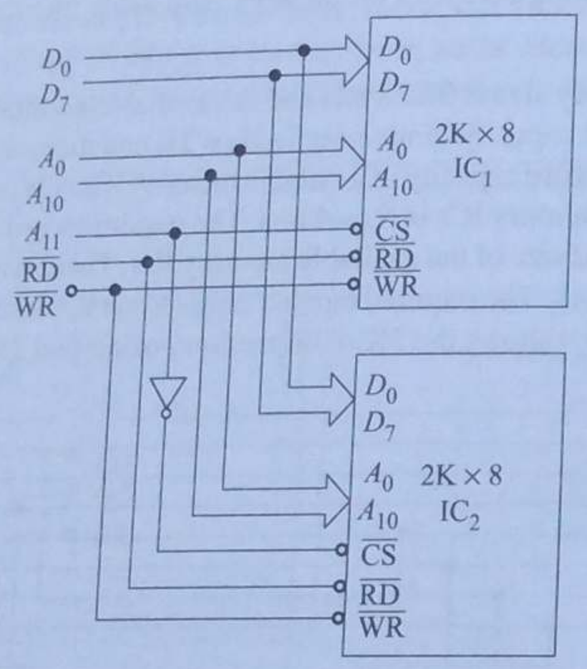


Fig. 10.10 4K×8 bit memory using two 2K×8 Chips

A 2K×8 bit memory has 11 ($2K = 2^n$, $n = 11$) address lines, A_{10}–A_0, and eight data lines, D_7 to D_0.

(a) The corresponding eight data lines of IC_1 and IC_2 (D_7 to D_0) are connected together to get the common data lines.
(b) $\overline{RD}$, $\overline{WR}$ of IC_1 and IC_2 are connected to get the common control signals.
(c) The corresponding eleven address lines of IC_1 and IC_2 (A_{10} to A_0) are connected together to get the common address lines.
(d) Address line A_{11} is directly connected to $\overline{CS}$ of IC_1, after the inverter is connected to $\overline{CS}$ of IC_2.

When address line A_{11} goes low, IC_1 is enabled and IC_2 is disabled, the address range of IC_1 is 0000,00000,0000 to 0111,1111,1111. When address line A_{11} goes high, IC_1 is disabled and IC_2 is enabled, the address range of IC_2 is 1000,0000,0000 to 1111,1111,1111, and the address range of 2K×8 bit memory is 0000,0000,0000 to 1111,1111,1111.

10.6 CHARACTERISTICS OF MEMORY DEVICES

All modern computers and microprocessor systems manufacturing has become possible due to the development of inexpensive and reliable VLSI semiconductor memory chips. Let us study the comparative behaviours of various semiconductor memory devices and the characteristics of memory devices such as storage capacity, units of transfer, access modes, performance of storage, and physical characteristics.

Storage capacity

Storage capacity is a representation of the size of memory. The capacity of memory can be expressed in terms of the number of words or bytes. For 1K×8 bit memory, the capacity is 1024×8 = 8192 bytes.

Units of transfer

It is defined as the number of bits read in a single read cycle or written in a single write cycle. For main memory and internal memory, it is equal to the word length of the processor and the size of data bus of the processor. For external memory, it is quite large and is referred to as the *block of data*.

Access modes

The external device (microprocessor) performs read or write operation with the memory locations. There are three different techniques used to access (read or write) the memory locations:

(a) Random access
(b) Sequential access
(c) Direct access

Random access In random access mode, any memory location can be accessed in any order in the same amount of time. Semiconductor memories are the example of random access memories.

Sequential access In sequential access mode, memories can be accessed in a predefined sequence. For example, the songs stored on a cassette can be accessed one by one. The time required for accessing the word depends on where the word is located in the memory. The first song can be accessed immediately and the last song requires more time. Magnetic tapes are the example of sequential access memories.

Direct access Direct access mode is also referred as *semi-random mode*. In certain cases, the data is accessed neither randomly nor in sequence, but something in between. In this kind of access, a separate read/write head exists for a track and on a track, the information can be accessed serially. This semi-random mode of operation exists in magnetic disks.

Access time

It is the time from a valid read/write signal to the data available or written at the desired location. The access time depends on the physical characteristics and the accessing mode used for that device.

Performance of storage

In some of the memories, the data is lost over a period of time. There are several reasons for data loss, such as destructive readout, dynamic storage, and volatility and hardware failure.

The reading process destroys the stored data, which is called as *destructive readout*. In such memories, the data has to be written back on the same location. The reading process where the data is not destroyed on reading is referred to as *non-destructive readout*.

If the data is lost due to a power failure, such a memory is known as *volatile memory*. If there is no data loss on power failure, it is known as *nonvolatile memory*. RAM is volatile and magnetic tapes are non-volatile.

Cycle time

Cycle time is the minimum time elapsed between two successive read requests. For most semiconductor memories, cycle time is equal to the access time. But for destructive readout memories, cycle time is the addition of time required to read the data and the time required to write the data back in the same memory location, as the previous value has been destroyed by reading.

Data transfer rate

Data transfer rate is the amount of data that can be transferred in or out of the memory in a second. It is also known as the *bandwidth of memory*. It is measured in bits per second. The maximum number of bits that can be transferred in a second depends on how many bits can be transferred in or out of the memory simultaneously and the speed of memory.

Physical characteristics

According to physical characteristics, memories are classified into four categories: electronic, magnetic, mechanical, and optical. The physical size of a memory also depends on its storage density. Energy consumption is another key factor, which determines the power consumption of the computer and its cooling requirements. Higher the power consumption, costlier will be the equipment required for internal cooling of the computer.

Cost

It is an important factor and is generally expressed per bit. The cost of the memory system includes not only the cost of memory device but also the cost of accessing circuit and peripheral essentials for the operation of the memory.

10.7 CLASSIFICATION OF SEMICONDUCTOR MEMORIES

According to the principle of operations, memories are classified as

- Sequential memory
- Read and write memory
- Read only memory
- Content addressable memory

On the basis of physical characteristics, memories are classified as

- Erasable or non-erasable memory
- Volatile or non-volatile memory

On the basis of accessing modes, memories are classified as

- Sequential access memory
- Random access memory

On the basis of fabrication technology, memories are classified as

- Bipolar memory
- Unipolar memory

Sequential memory

In sequential memories, the data can be accessed in a predefined sequence. Therefore, the access time of a sequential memory varies depending on the location to be accessed. The first location requires less time and the last location requires more time. Magnetic audio/video cassettes are the examples of sequential memory. Shift register and charge-coupled devices are the examples of sequential semiconductor memory.

Shift registers consist of static and dynamic cells. In a static cell, the data is stored in terms of voltage. It cannot be lost as long as the power is ON. In a dynamic cell, the data is stored in terms of charge, which changes with time. The charge-coupled devices have high density and low cost. It is implemented using MOS technology.

Read-write memory

This type of memory supports both read and write operations. On the basis of physical characteristics, RWM or RAM is further classified as (i) static RAM and (ii) dynamic RAM.

Read only memory (ROM)

ROM allows the user to read the data, but write permission is denied. On the basis of the technique used for storing data, ROM is further classified as

- Masked ROM
- Programmable ROM
- Erasable Programmable ROM
- Electrically Erasable Programmable ROM

Content addressable memory

It is a random access memory where the locations are identified by their content.

Erasable or non-erasable memory Erasable memory, e.g. RAM, allows the user to erase old data and write new data. In non-erasable memory, e.g. ROM, data once written cannot be erased.

Volatile or non-volatile memory The memories are said to be volatile, where the content is lost when the power is switched off. Read-write memory is the example of volatile memories.

On the other hand, non-volatile memories retain their content even after the power is switched off. Read only memories are the example of non-volatile memories.

Sequential access memory In sequential access memories, the information can be accessed in a predefined sequence. Therefore, the access time of a sequential memory varies depending on the location to be accessed.

Random access memory In RAM, any memory location can be accessed in any order in the same amount of time. Semiconductor memories are the example of random access memories.

Bipolar or unipolar memory The transistor is a bipolar device and MOS is a unipolar device. TTL, ECL are the bipolar logic families and NMOS, CMOS, PMOS are the unipolar logic families. The memory is said to be bipolar, when it is fabricated using bipolar technology. RAM, ROM and PROM are the examples of bipolar logic memories. The dynamic RAM, EPROM and EEROM can be fabricated using unipolar devices only and such memories are known as unipolar memories.

10.8 READ AND WRITE MEMORY

Read and write memory is the main memory of a digital system. In read and write memory, data can be read or written from or into the memory with the same speed. These memories are also referred as Random Access Memories. The read and write memory is volatile and therefore, the data stored in memory will be lost, whenever the power is switched off.

RAM is further classified as (i) static RAM and (ii) dynamic RAM.

10.8.1 Static RAM

In static RAM, the content does not change with time and the data is stored in terms of voltage; the flip-flop is a basic cell of static RAM. The cells are arranged in rows and columns. The number of rows defines the number of locations and the number of cells in each row (number of columns) defines the size of words. The cells of RAM are constructed using bipolar or MOS transistors.

TTL RAM Cell

TTL RAM cells are implemented by bipolar transistors. A bipolar transistor cell is also known as bipolar memory cell. The circuit diagram of a TTL RAM cell is shown in Fig.10.11.

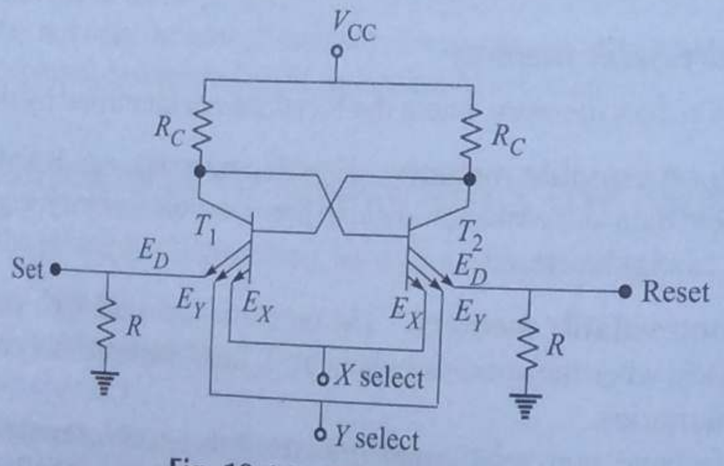


Fig. 10.11 TTL RAM cell

The transistors T_1 and T_2 are multi-emitter transistors. They are cross-coupled, i.e. the output of T_1 is connected to the input of T_2 and the output of T_2 is connected to the input of T_1. T_1 and T_2 act as cross-coupled inverts and hence, when T_1 is ON, T_2 is OFF, and vice versa. The X and Y are the select input lines to select the cell from the matrix. To select the cell, both the select lines must be held HIGH. If either X or Y select line is LOW, then the memory cell is disabled. The structure of a TTL memory cell with write/$\overline{\text{read}}$, data input and data output signal is shown in Fig. 10.12. The data stored in memory cell is logic 1, if T_2 is in cut-off and T_1 is in saturation. The data stored in memory cell is logic 0, if T_1 is in cut-off and T_2 is in saturation.

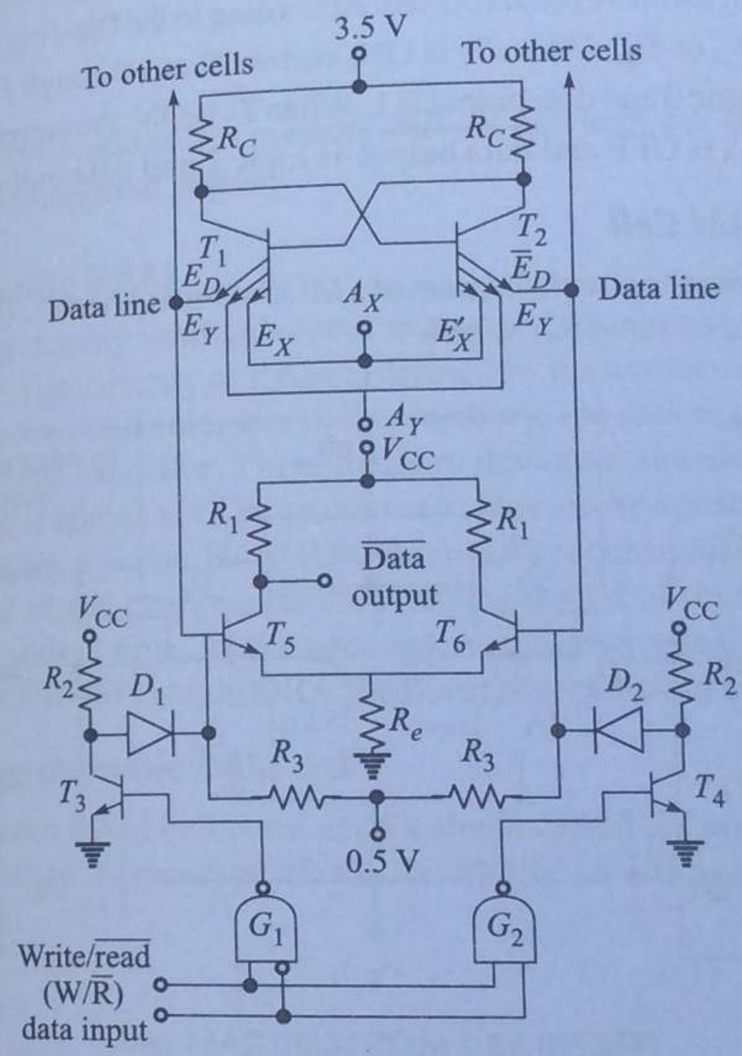


Fig. 10.12 TTL memory cell with write/$\overline{\text{read}}$

Write operation For write operation, the select inputs, A_x and A_y, are logic 1. The corresponding emitter junction of T_1 and T_2 are reverse biased. The write/$\overline{\text{read}}$ control input is logic 1 and the data to be written is loaded on input data line.

Assume that the data to be written is logic 1. Since W/$\overline{\text{R}}$ =1 and data input =1, the output of gate G_1 will be 1 and the output of G_2 will be 0, T_3 is ON and T_4 is OFF. The voltage at the collector of T_4 is high and diode D_2 is forward biased. The voltage at the collector of T_3 is low and diode D_1 is reverse biased. The voltage at the reset input becomes high and T_2 goes in cut-off and T_1 in saturation and logic 1 data will be stored.

Assume that the data to be written is logic 0. Since W/$\overline{R}$=1 and data input =1, the output of G_1 will be 0 and the output of G_2 will be 1, T_3 is OFF and T_4 is ON. The voltage at the collector of T_3 is high and diode D_1 is forward biased. The voltage at the collector of T_4 is low and diode D_2 is reverse biased. The voltage at the set input becomes high and T_1 goes in cut off and T_2 in saturation and logic 0 data will be stored.

Read operation For read operation, the select inputs, A_X and A_Y, are logic 1, the corresponding emitter junction of T_1 and T_2 are reverse biased. The write/$\overline{\text{read}}$ control input is logic 0. The output of G_1 and G_2 is 1, therefore T_3 and T_4 are ON; the diodes D_1 and D_2 are reversed biased. According to the flip-flop state, the current flows through E_D or $\overline{E}_D$. When T_1 is ON, current flows through E_D, T_5 is ON and $\overline{\text{data output}}$ is logic 0 and data output is 1. When T_2 is ON, the current flows through $\overline{E}_D$, T_6 is ON, T_5 is OFF and $\overline{\text{data output}}$ is logic 1 and data output is 0.

MOS static RAM Cell

Figure 10.13 shows the circuit diagram of a MOS static RAM cell. The enhancement MOSFET is used in static RAM cell.

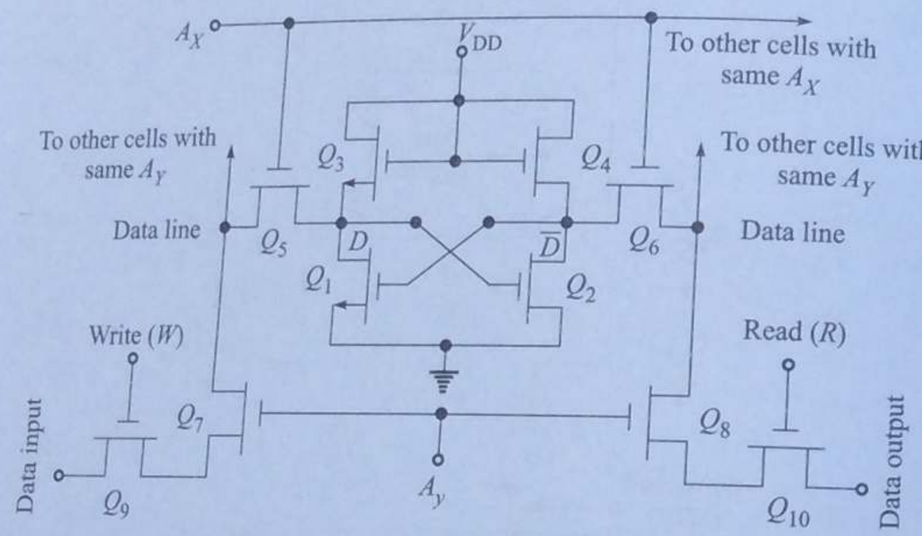


Fig. 10.13 MOS static RAM cell

MOSFET Q_1 and MOSFET Q_2 form the basic cross-coupled inverters. GATE and DRAIN terminals of Q_3 and Q_4 are short circuits. Q_3 and Q_4 act as load resistors. Q_5 and Q_6 act as switch. When select input A_X= logic1, Q_5 and Q_6 get ON and the cell is connected to Data line and $\overline{\text{data}}$ line.

Similarly, Q_6 and Q_7 act as switch. When select input A_Y= logic1, Q_7 and Q_8 get ON and the data is read or written. If Q_1 is ON and Q_2 is OFF, then the data stored in the cell is '0' and if Q_1 is OFF and Q_2 ON, then the data stored in the cell is 1.

Write operation For write operation, write signal is 1, select line A_X is 1 and A_Y is 1 and data is given to the DATA line. Since write signal is 1 and A_Y is 1, Q_9 and Q_7 get ON, the data is connected to the data line and since A_X is 1, the data line is connected to the drain terminal of Q_1 and the gate terminal of Q_2. When data input is 1, Q_1 is OFF and Q_2 is ON. When data input is 0, Q_1 is ON and Q_2 is OFF.

Read operation For read operation, read control signal is 1, A_X is 1 and A_Y is 1. Since read signal is 1, A_Y is 1 and A_X is 1. MOSFETs Q_{10}, Q_8 and Q_6 get ON, data out line is connected to the drain terminal of Q_2 and the data is read.

Advantages

1. The access time is less, i.e. static memory is fast.
2. Refreshing circuit is not required.
3. Cost is less.

Disadvantages

1. Static RAM contains less memory cell per unit area, i.e. density is low.
2. Power consumption is less.

10.8.2 Dynamic RAM

It is the highest density random access memory. Capacitor is the basic cell of dynamic RAM. The capacity of RAM is defined by the number of cells. The cells are arranged in rows and columns. In this memory, the data is stored in terms of electrical charge on capacitor. These memory devices make use of an integrated MOS capacitor; it allows very large memory arrays to be constructed on a small size chip. Therefore, dynamic RAM (DRAM) has more density. The MOS capacitor cannot hold the stored charge over an extended period of time and it has to be refreshed thousands of times within a second. It requires a refreshing circuit; hence the total cost of DRAM is high. DRAM offers reduced power consumption.

Four-transistor dynamic RAM cell

A simple-dynamic RAM cell consists of a single MOSFET and a capacitor. It is shown in Fig. 10.14. In this type of cell, the MOSFET acts as a switch.

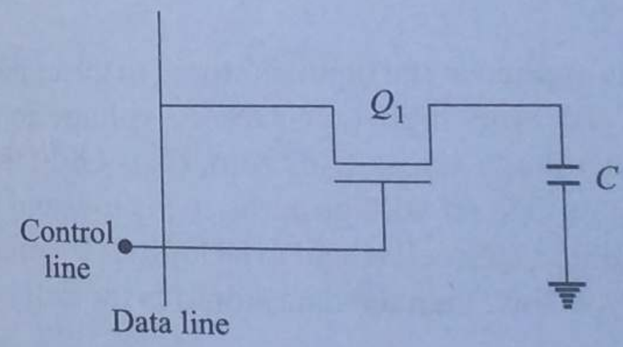


Fig. 10.14 Simple-dynamic RAM cell

When the sense line and control line are high, the MOSFET Q_1 conducts, data is stored in capacitor, i.e. the capacitor gets charged for logic 1. When the sense line and control line are low, Q_1 is OFF and the capacitor remains in the original state. Figure 10.15 shows a four-transistor dynamic RAM cell with read, write, and control signals and data input and data output lines with four transistors.

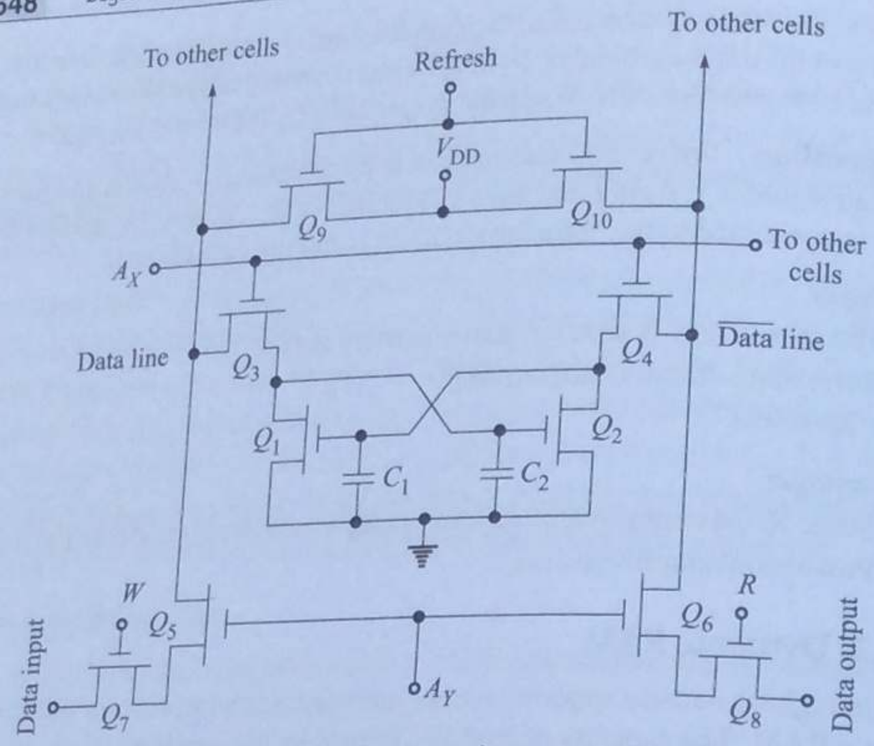


Fig. 10.15 Four-transistor dynamic RAM cell

Q_1 and Q_2 form the basic cross-coupled inverters. Q_3 and Q_4 act as a switch, which is controlled by A_X. Q_5 and Q_6 act as a switch, which is controlled by the input A_Y. Q_9 and Q_{10} act as a switch, which is controlled by the input refresh signal. Q_7 and Q_8 are controlled by the write signal and read signal, respectively. Q_7 and Q_8 are common for the entire cell, which are in the same column. The data is stored in capacitors C_1 and C_2.

Write operation For write operation, the select inputs, A_X and A_Y, are at logic 1, write signal is high and the data to be written is applied to the data input line. Since A_X is logic 1, A_Y is logic 1, and write signal is high, the MOSFETs Q_7, Q_5, Q_3, and Q_4 are ON, data input line is connected to the drain terminal of Q_1 and gate terminal of Q_2.

The data, which is applied at the input, is stored in the capacitor. If it is logic 1, the voltage across C_2 becomes high, Q_2 is ON, the voltage across C_1 is 0 and Q_1 is OFF. If data is 0, the voltage across C_2 is zero, Q_2 is OFF, the voltage across C_1 becomes high and Q_1 is ON. (If voltage across C_1 is low and voltage across C_2 is high, then the data stored in the cell is said to be logic 1. If voltage across C_1 is high and voltage across C_2 is low, then the data stored in the cell is said to be logic 0).

Read operation For read operation, A_X is logic 1 and A_Y is logic 1, read signal is high and the data will be available on $\overline{\text{data}}$ out line. Since A_X is logic 1, A_Y is logic 1 and read signal is high, MOSFETs Q_5, Q_6, Q_8 and Q_4 are ON. $\overline{\text{data}}$ out line is connected to the drain terminal of Q_2. If the data stored in the cell is logic 1, Q_2 will be ON and voltage at drain terminal will be low, i.e. $\overline{\text{data}}$ out will be 0 which means data out will be 1. If the data stored in the cell is logic 0, Q_2 will be OFF and voltage at drain terminal will be high, i.e. $\overline{\text{data}}$ out will be 1 which means data out will be 0.

Refresh operation Under normal conditions, A_X is logic 0, A_Y is logic 0, MOSFETs Q_5, Q_6, Q_7, and Q_8 are OFF. We have assumed that the charge in capacitor cannot be lost. But because of leakage current, the capacitors are discharged and the stored data is lost. Hence, there is a need to use the refresh cycle after a fixed interval of time.

For refresh operation, A_X is logic 1 and refresh signal is high, the MOSFETs Q_3, Q_4, Q_9 and Q_{10} are ON and the refresh cycle starts.

Assume initially, voltage across C_1 is high and C_2 is low, Q_1 is ON and Q_2 is OFF. During the refresh interval, V_{DD} is applied to C_1 through Q_{10} and Q_4, and to C_2 through Q_9 and Q_3. Since Q_1 is ON and Q_2 is OFF, the current flows through C_1, and C_1 is charged.

Assume initially, voltage across C_1 is low and C_2 is high, Q_1 is OFF and Q_2 is ON. During the refresh interval, V_{DD} is applied to C_2 through Q_9 and Q_3, and to C_2 through Q_{10} and Q_4. Since Q_1 is OFF and Q_2 is ON, the current flows through C_2, and C_2 is charged.

Three-transistor dynamic memory cell

The circuit of a three-transistor dynamic memory cell is divided into two parts: *dynamic memory cell* and *refresh circuit*. The memory cell consists of MOSFETs Q_1, Q_2 and Q_3. The refresh circuit consists of MOSFETs Q_9, Q_{10} and Q_{11}. Q_1 is used to write the data and Q_2 and Q_3 are used to read the data from the memory. Q_4 acts as a switch, which is controlled by the write signal. Q_6 also acts as a switch, which is controlled by the read signal. Q_7 and Q_8 act as switch, which are controlled by A_Y. Figure 10.16 shows the circuit of a three-transistor dynamic memory cell.

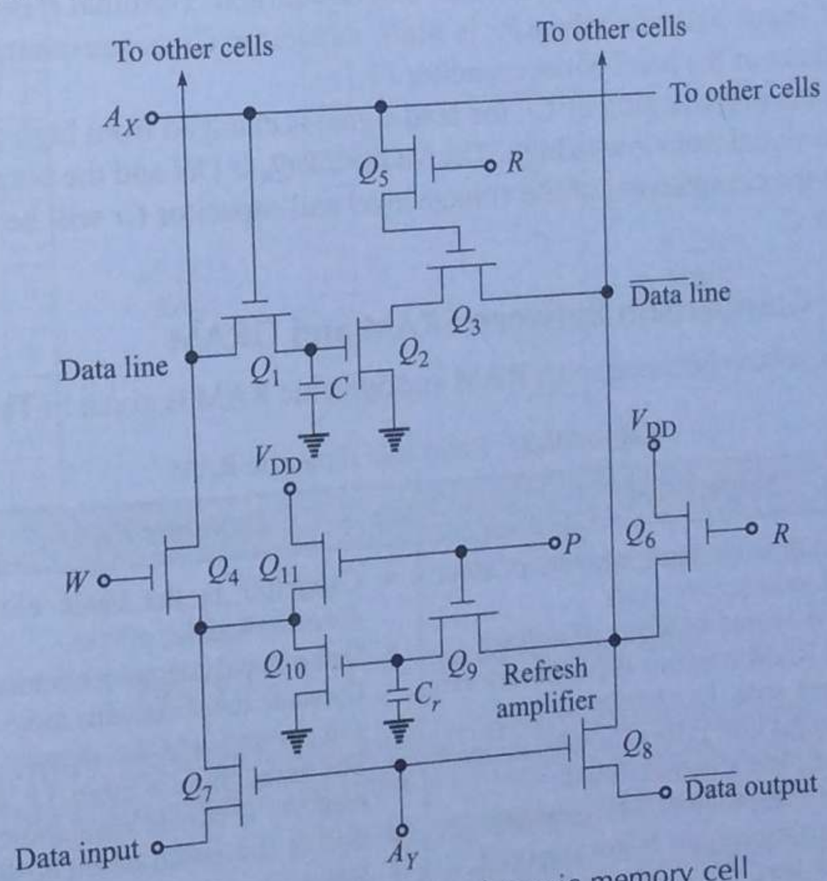


Fig.10.16 Three-transistor dynamic memory cell

Write operation For write operation, both A_X and A_Y are high, write signal is high and the data to be written is applied to the data input line and the refresh circuit is disconnected keeping P to logic 0. Since both A_X and A_Y are high and write signal is high, MOSFETs Q_7, Q_1 and Q_4 are ON. The data which has to be written will be loaded into the memory cell, i.e. when the data is logic 1, the capacitor is charged and Q_2 is ON, and when the data is logic 0, the capacitor is discharged to zero and Q_2 is OFF.

Read operation For read operation, both A_X and A_Y are high, read signal is high, the refresh circuit is disconnected keeping P to logic 0 and the data is read at $\overline{\text{data}}$ output line. Since both A_X and A_Y are high and read signal is high, MOSFETs Q_8, Q_6, Q_5 and Q_3 are ON and $\overline{\text{data}}$ output line is connected to the drain terminal of Q_2, and the contents of memory is available on $\overline{\text{data}}$ out line. When the data in memory is logic 1, Q_2 is ON, voltage at drain terminal is low, $\overline{\text{data}}$ out is equal to zero, i.e. data out is equal to logic 1. When the data in memory is zero, Q_2 is OFF, the voltage at drain terminal is high, $\overline{\text{data}}$ out is high and data out is low.

Refresh operation For refresh operation, A_X is high, A_Y is low, P is high and read signal is high. Since A_Y is low, Q_7 and Q_8 are OFF. Data input and $\overline{\text{data}}$ out lines are disconnected from the cells of memory. The MOSFETs Q_5, Q_3 and Q_9 are ON; the complement of C is stored in capacitor Cr. When voltage across the capacitor C is high, Q_2 is ON, the voltage at drain terminal is low, which is loaded into the capacitor Cr. When voltage across the capacitor C is low, Q_2 is OFF, voltage at drain terminal is high and capacitor Cr will be charged. Terminal P is called pre-charged input because when Pr is high, capacitor Cr is pre-charged to the complement of the level of the capacitor C.

After the pre-charging of Cr, the read signal is changed from high to low and the write signal from low to high. The MOSFET Q_4 is ON and the output of Q_{10}, which is the complement of the voltage level and capacitor Cr will be written to capacitor C.

10.8.3 Comparison Between SRAM and DRAM

The comparison between static RAM and dynamic RAM is given in Table 10.2.

Table 10.2 Static and dynamic RAM

Static RAM	*Dynamic RAM*
• Flip-flop is the basic element of static RAM. • Data is stored in terms of voltage. • Static RAM contains less memory cell per unit area, i.e. density is low. • The access time is less, i.e. static memory is faster than dynamic RAM. • Cell is composed of TTL or MOSFET. • Refreshing circuit is not required. • Cost is less. • Power consumption is less.	• Capacitor is the basic element of dynamic RAM. • Data is stored in terms of electrical charge. • Dynamic RAM contains more memory cell per unit area, i.e. density is high. • The access time is more, i.e. dynamic memory is slower than static RAM. • Cell is composed of MOSFETs. • Refreshing circuit is required. • Cost is high. • Power consumption is more.

10.8.4 Commonly Used ICs for RAM

Some commonly used ICs for RAM are listed below.

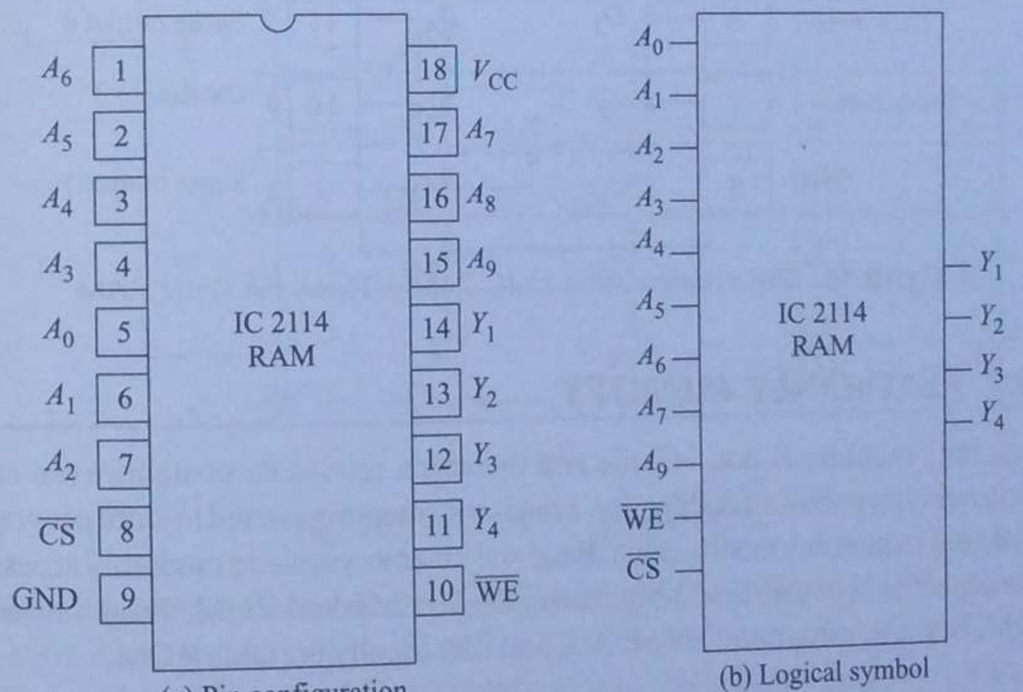
IC number	Description
2114	1K×4 bit static RAM
7489	16 ×4 bit static RAM
6116	2K×8 bit static RAM
6264	8K×8 bit static RAM
62256	32K×8 bit static RAM
2104	4K×1 bit dynamic RAM
2164	6K×1 bit dynamic RAM
4164	64K×1 bit dynamic RAM
41256	256K×1 bit dynamic RAM

10.8.5 Static RAM ICs

In this section, we will discuss a few commonly used RAM ICs.

IC 2114

It is a 1K×4 bit static RAM; total cells are $1024 \times 4 = 4096$. The IC uses n-channel silicon gate MOS transistors. It is available in a single 18-pin IC package. The pin configuration and the logical symbol are shown in Fig.10.17.



(a) Pin configuration (b) Logical symbol

Fig. 10.17 IC 2114—1K×4 bit static RAM

In Fig. 10.17, $\overline{\text{CS}}$ is active low chip select input signal. If this signal goes low, then the chip will be enabled. This signal is also used to expand the size of memory, as explained in Section 10.5. $\overline{\text{WE}}$ is the input signal, it differentiates the read and write operations. For read operation, $\overline{\text{CS}}$ must be low, $\overline{\text{WE}}$ must be high and for write operation, $\overline{\text{CS}}$ and $\overline{\text{WE}}$ must be low. The IC has ten address lines and four data lines.

IC 7489

It is a 16×4 bit static RAM; total cells are 16×4=64. The IC uses transistor-transistor logic (TTL). It is available in a single 16-pin IC package. The pin configuration is shown in Fig. 10.18.

In Fig. 10.18, $\overline{ME}$ is active low memory enable input signal. If this signal goes low, then the memory will be enabled. This signal is also used to expand the size of memory, as explained in Section 10.5. $\overline{WE}$ is the input signal, it differentiates the read and write operations. For read operation, $\overline{ME}$ must be low, $\overline{WE}$ must be high and for write operation, $\overline{ME}$ and $\overline{WE}$ must be low. The IC has four address lines, four data input lines, and four data output lines.

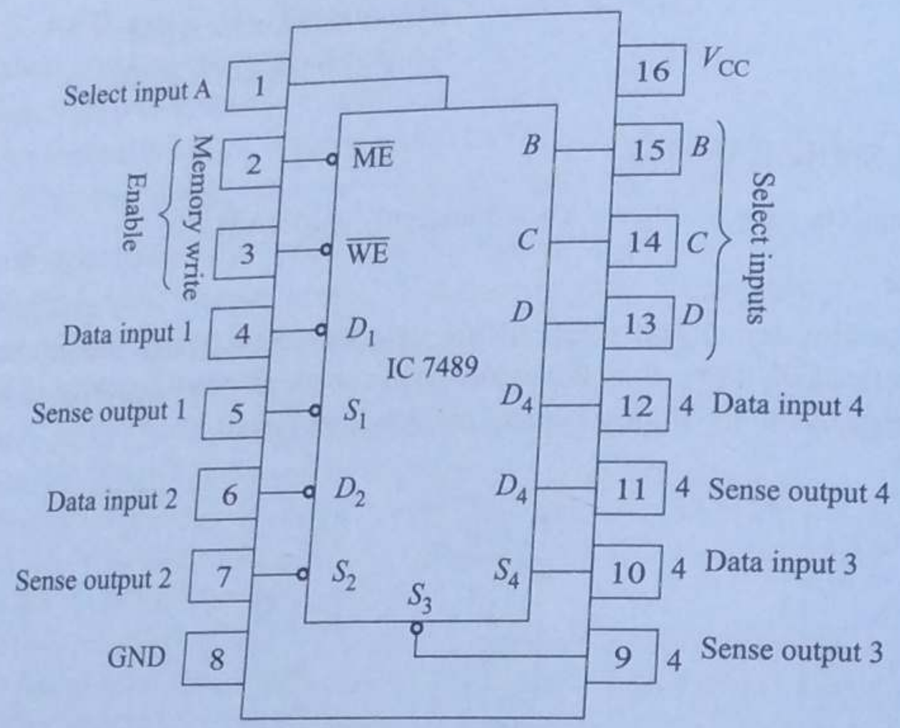


Fig.10.18 *Pin configuration of IC 7489—16×4 bit static RAM*

10.9 READ-ONLY MEMORY

Read-only memory is non-volatile and therefore, retains its contents even after the power is switched off. Generally, a read-only memory is used to store programs that do not require any modification. Read-only memory is also a randomly accessed memory. The types of read-only memories are: Masked ROM, Programmable ROM, Erasable-programmable ROM, and Electrically erasable ROM.

10.9.1 Masked ROM

Masked ROM is permanently programmed during the manufacturing process. It is programmed by the manufacturer according to the application. Once it is programmed, it cannot be changed, hence the binary information must be specified by the designer and then embedded in the memory unit. Figure 10.19 shows an 8×4 bit masked ROM.

Figure 10.19 shows a 8×4 bit masked ROM. The unit has eight rows and four columns (eight locations and four words) and a 3:8 decoder. The outputs of the decoder are active low and they are connected to memory cells which are arranged in rows.

The four data lines are connected to memory cells and these are arranged in columns through the tristate buffer.

The rows are connected with the columns through the diode, where the data to be stored as '0'. If the data is to be stored as '1', then the rows and columns are not connected.

Suppose the data to be stored in location '0' (000) is 1010, the columns 0 and 2 are connected to the rows through the diode and columns 1 and 3 are not connected.

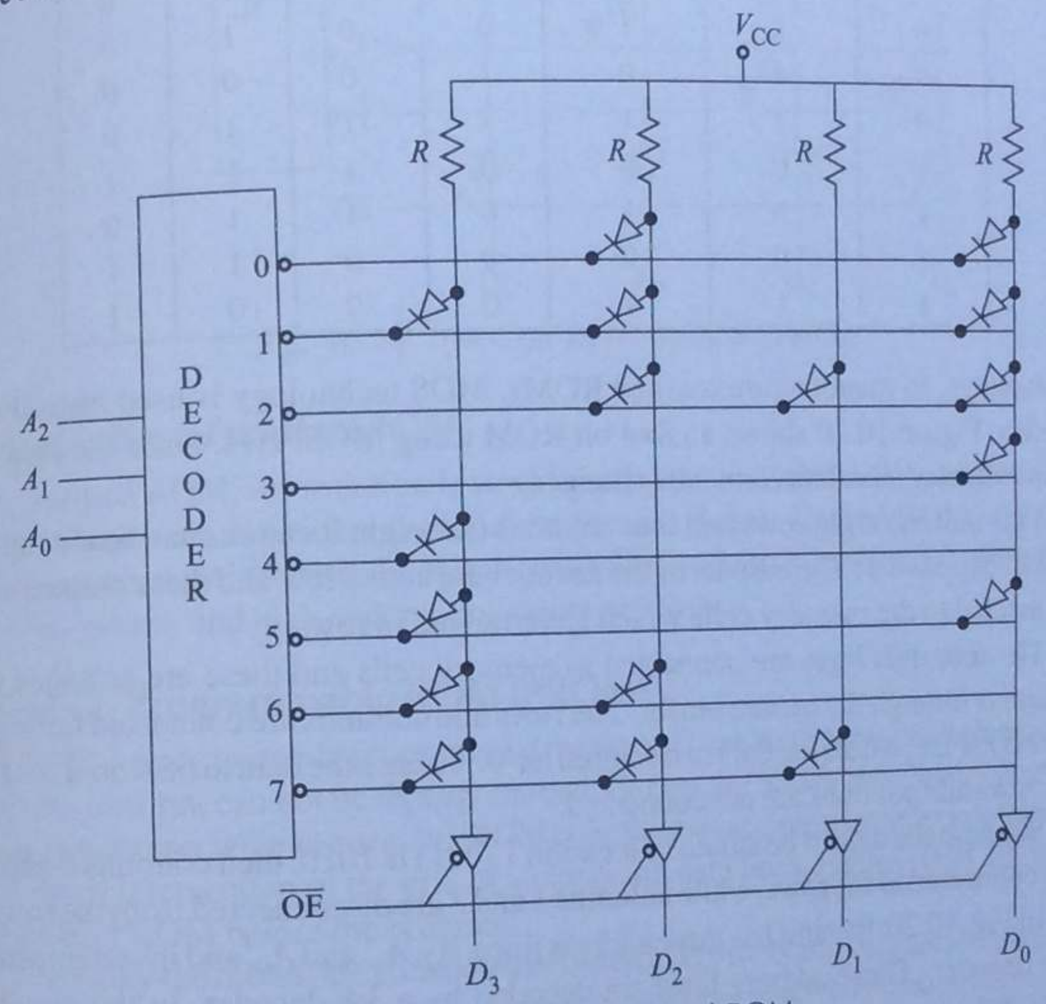


Fig. 10.19 8×4 bit Masked ROM

This unit has three address lines, A_2, A_1, and A_0. These address lines are decoded by a 3:8 decoder.

For read operation, the following actions take place: (a) chip select signal goes low, (b) the address loaded over the address lines is decoded by the decoder, (c) the corresponding output of the decoder goes low, (d) one of the row is enabled, (e) diodes which are connected between the columns and the enable row are forward biased and the data on the line is 0, otherwise the data is 1. The data is read by applying an active low read signal to the $\overline{OE}$ pin.

Assume the address loaded over the address lines is 0101. The output $\overline{Q_5}$ of the decoder goes low. The diodes which are connected between the columns and row

number 5 are enabled. The data is loaded over the internal data lines, which can be read by applying the $\overline{read}$ signal. Since columns 0 and 3 are connected to row 5 through the diode and columns 1 and 2 are not connected to row 5, the data available over the data lines is 0110.

Table 10.2 shows the address of the location and the contents of the corresponding memory location for Fig. 10.19.

Table 10.2 Address and Contents of memory location for Fig.10.19

Address of the location			*Contain of the location*			
A_2	A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	1	0	1	0	0	0
0	1	1	1	1	1	0
1	0	0	0	1	1	1
1	0	1	0	1	1	0
1	0	0	0	0	1	1
1	1	1	0	0	0	1

Nowadays, to manufacture masked ROMs, MOS technology is used instead of diodes. Figure 10.20 shows an 8×4 bit ROM using MOSFETs. Once the chip is manufactured, the data cannot be changed.

This unit has eight rows and four columns (i.e. eight locations and four words) and a 3:8 decoder. The outputs of the decoder are active low and these outputs are connected to the memory cells which are arranged in rows.

The four data lines are connected to memory cells and these are arranged in columns through the tristate buffer. The rows and columns are connected through the MOSFET, where the data to be stored as '0'. In case the data to be stored as '1', the rows and columns are not connected.

Suppose the data to be stored in location 1 (001) is 1010, then columns 0 and 2 are connected to the rows, while columns 1 and 3 are disconnected from the rows.

In Fig. 10.20, the unit has three address lines, A_2, A_1, and A_0, and inverters after the decoder. These address lines are decoded by a 3:8 decoder. In the circuit, MOSFETs are controlled by the output of the decoder and Q_4, Q_3, Q_2, and Q_0 act as registers.

For read operation, the following actions take place: (a) the address of the location is loaded over the address lines, (b) the select lines go high, as inverters are used after the decoder, (c) MOSFETs are ON, (d) the data is read by applying an active low read signal. If the MOSFETs are connected between the selected rows, then the data on the line is 0. Else, if the MOSFETs are not connected, the data on the line is 1.

Assume the address loaded over the address lines is 010. Row 2 is selected and it goes high. (b) The MOSFETs connected to row 2 are ON and connect the corresponding columns to ground. As a result, the data on the line will be 0, otherwise it is 1. For the circuit, the data available over the bus will be 0101.

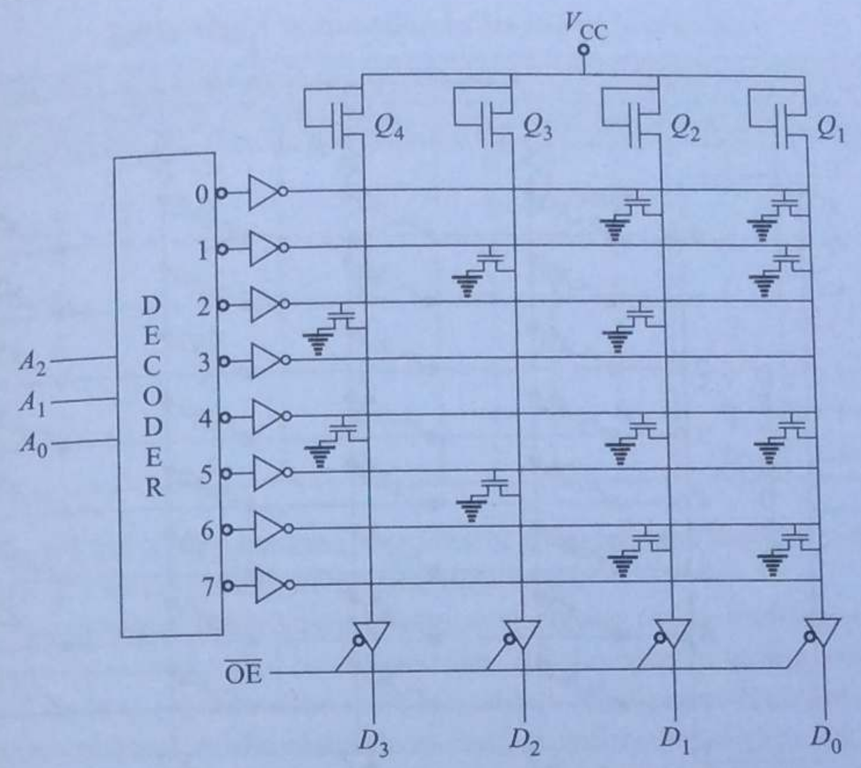


Fig. 10.20 8×4 bit ROM using MOSFETs

Application of masked ROM

In masked ROM, information is stored during the manufacturing process. Once the information is written, it cannot be altered. Hence, masked ROM is used to store permanent information. It is used in microprocessor-based toys, computers, video games, and high-volume consumer products.

10.9.2 Programmable ROM (PROM)

PROM is a permanently programmed memory. It can be programmed electrically by the user but can not be reprogrammed. Hence, the binary information must be ready with user without error. In PROM chip, the manufacturer includes a connection at every intersection of the grid of address and data lines. Figure 10.21 shows an 8×4 bit PROM before the program.

Initially the diodes are connected between every rows and columns through the fusible links, therefore the initial information stored in all the location and all the bits are zero; i.e. at the time of fabrication, the information in memory is zero.

The fuse material used is either nichrome or polycrystalline silicon, the fusible links can be melted by passing a large current through it or these fuse can be blown OFF by passing a large current through it. A memory link is fused open or intact to represent a binary number 0 or 1. The user can selectively burnout some of the fuses by passing enough current through them and can program the PROM according to the required binary information.

If nichrome is used as the fuse material, then 20 to 50 mA current is required to blow OFF. The fusing time varies between 5 μs and 200 μs and the programming rate is 5 ms per bit.

The information to be stored in memory is given in Table 10.3. Figures 10.21(a) and (b) show the 8×4 bit PROM before and after the program, respectively.

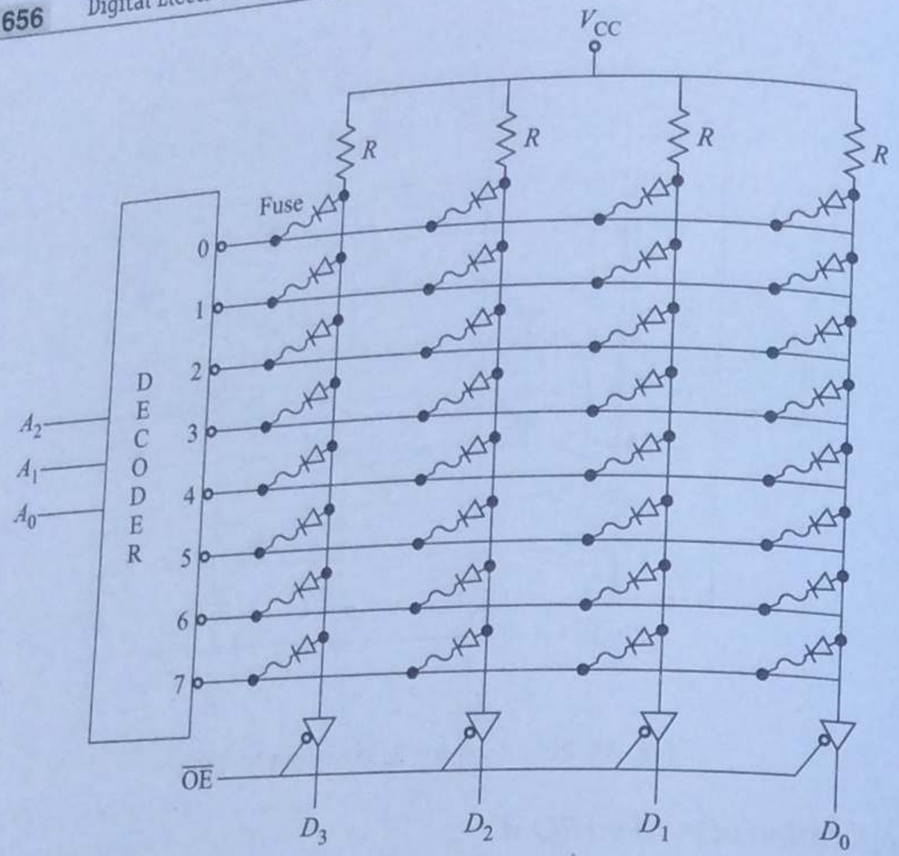


Fig. 10.21(a) 8×4 bit PROM before the program

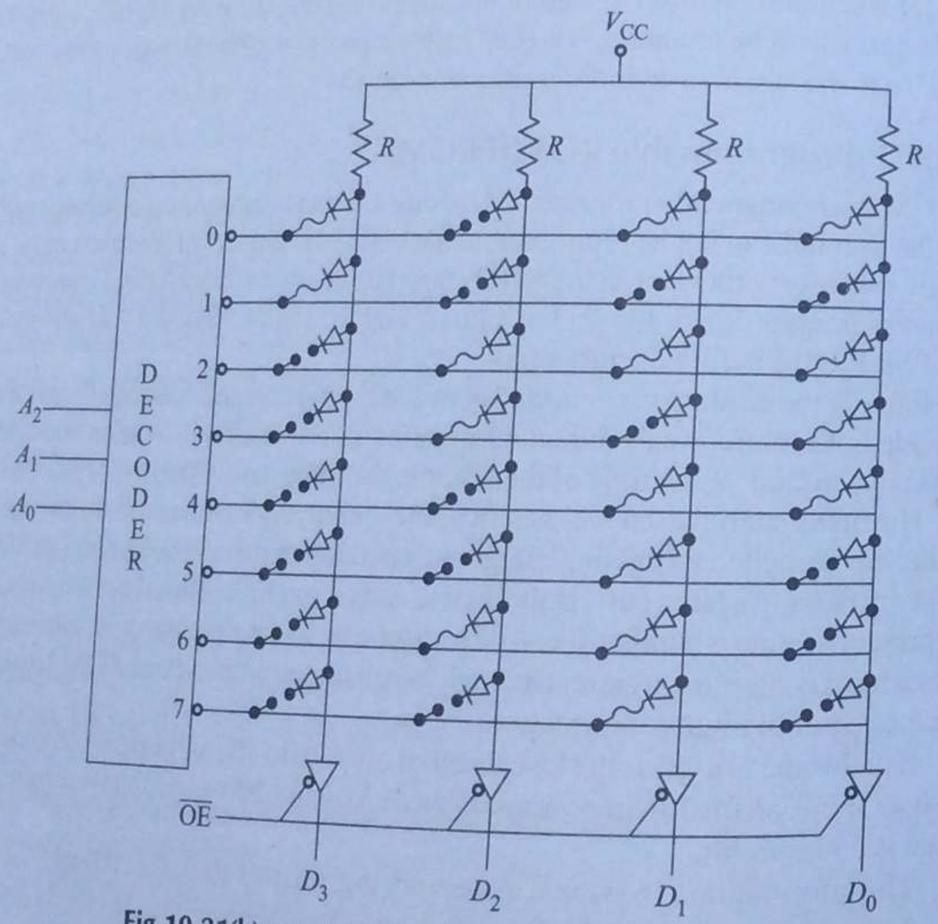


Fig.10.21(b) 8×4 bit PROM after the program

Table 10.3 Information to be stored in memory

Address			Data			
A_2	A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	1	0	0
0	0	1	0	1	1	1
0	1	0	1	0	0	0
0	1	1	1	0	1	0
1	0	0	1	1	0	0
1	0	1	1	1	0	1
1	1	0	1	0	0	1
1	1	1	1	1	0	1

Once a PROM is programmed, it cannot be changed, and therefore, it has to be done carefully and correctly in the first time itself.

For read operation, the following actions take place: (a) the address loaded over the address lines is decoded by the decoder, (b) the corresponding output of the decoder goes low, (c) one of the rows is enabled, (d) diodes which are connected between the columns and the enabled row conduct and the data on line is 0, otherwise the data is 1. The data is read by applying an active low read signal to the $\overline{OE}$ pin of memory.

Fuse technology used in PROM

We studied that the fuse material used is either nichrome or polycrystalline silicon. If nichrome is used as the fuse material, then 20 to 50 mA current is required to blow OFF and if polycrystalline silicon is used as the fuse material, then 20 to 30 mA current is required to blow OFF.

The third technology used for fuse is the *p-n* junctions technology. This technology is also referred to as *shorted junction* or *avalanche-induced migration*. It consist of two *p-n* junctions arranged back to back. For this technology, 200 to 300 mA current is required to shot the junction. These fuse technologies are irreversible. These technologies work only for diode and transistor memory devices. It cannot work with MOS memory devices where high current levels, required for the fusing process, are incompatible with impedance levels.

PROM programming

PROM is programmed by plugging it into a special device called PROM programmer. During the programming, the address is loaded over the address lines, which enable a selected location, and then a pulse is applied to those output lines where 1s are to be stored. The PROM starts with all 0s; these current pulses blow the fuse links and create the desired bit pattern. The next address is then selected and the process is repeated; this is done automatically by the PROM.

10.9.3 Erasable Programmable ROM (EPROM)

It is a semi-permanently programmed memory. It can be erased and reprogrammed. The basic structure of a typical EPROM cell is shown in Fig.10.22.

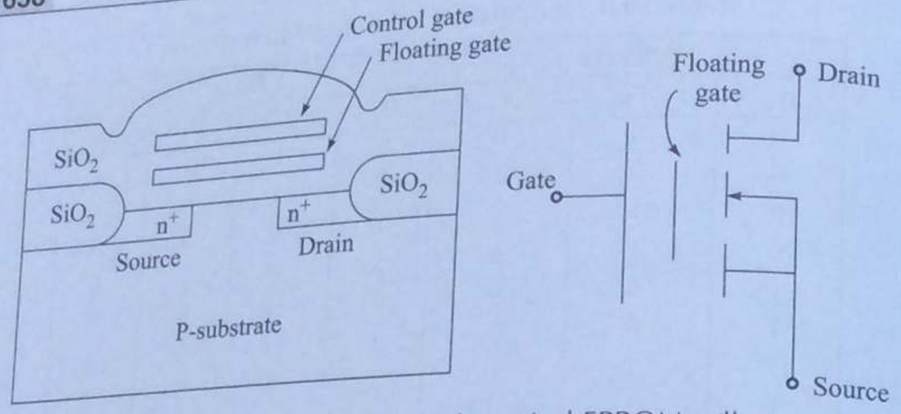


Fig.10.22 Basic structure of a typical EPROM cell

An additional floating gate is formed within the silicon dioxide (SiO_2) layer. The floating gate is disconnected and the normal gate is connected to the row decoder output. The basic principle of operation is to accumulate charge on the gate using external high voltage and while erasing, illuminate it with ultraviolet light. The data bits are represented by the presence or absence of a stored charge. The initial values of EPROM cells may be all 0s or all 1s.

The user can program the EPROM with a special programmer, known as EPROM programmer. All the contents of the memory can be erased by exposing the chip to ultraviolet light through its window which is available on top of the chip for a duration of 15 to 20 minutes, as shown in Fig.10.23. The EPROM window should be covered with an opaque sticker to protect the memory from unwanted exposure to UV rays.

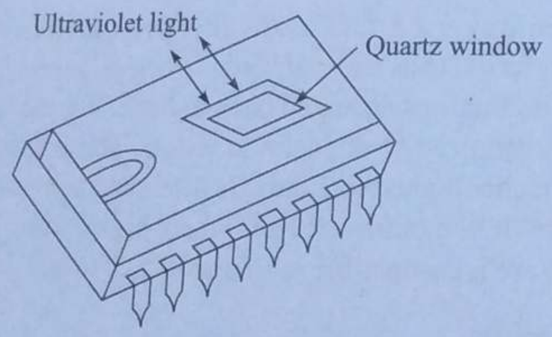


Fig. 10.23 EPROM

This chip can be reprogrammed. Therefore, this memory is suitable for the monitor program in microprocessor systems. Disadvantages of this chip are that, we cannot erase selective information; entire information is erased because of ultraviolet light, and the process of reprogramming cannot take place with the memory in circuit.

EPROM programming

Before reprograming the EPROM, all the cells of EPROM must be erased. This is done by illuminating the cells by ultraviolet light, having a wavelength of around 253.7 nm for about 20 minutes. Once the EPROM is erased, the values in the entire

cell are 1s (i.e. no charge over the floating gate).To program or store a 0 in such a cell, the floating gate must be charged. To charge the floating gate, a high voltage of about 16 to 20 V is applied between the source and drain, and a voltage of about 25 to 50 V is applied to the control gate for typically 50 ms per address location. Due to high voltage, an electric field is established and the charge is stored on the floating gate. Programming actually involves the selection of a desired cell gate by loading the address over the address bus, and then injecting charge onto the floating gate by applying the high voltages. Once the charge is stored on the floating gate, it remains same till the cell is not illuminated by an ultraviolet light because the gate is surrounded by SiO_2.

10.9.4 Electrically Erasable Programmable ROM (EEPROM)

In EEPROM, the electrical signal will erase the content of memory. It is also known as electrically alterable programmable ROM (EAPROM). EEPROMs also use MOS technology; it is similar to that of EPROM. The data is stored as charge or no charge on an insulated layer or an insulated floating gate in the device. The insulating layer is very thin (<200 A) and hence, a voltage as low as 20 to 25 V can be used to move the charges across the thin barrier in either direction for programming or erasing.

In EEPROM, we can erase the content of the memory at the register level rather than erasing all the information. The EEPROM memory also has a special chip erase mode by which the entire chip can be erased in 10 ms. The time required to erase an EEPROM is less as compared to the time required to erase an EPROM, and it can be erased and reprogrammed with a device right in the circuit. EPROMs are most expensive and the least dense ROMs.

10.9.5 NVRAM

NVRAM is a non-volatile RAM. It is a combination of a static RAM and an electrically erasable PROM (EEPROM) in the same chip. The EEPROM and the static RAM are the mirror memory structure of each other. When power is ON, it functions as a normal RAM. Whenever the power is switched OFF, it saves the stored data in the EEPROM. The entire content of the RAM can be transferred in parallel to the EEPROM in less than 4 ms. When the power is switched ON, the entire content of EEPROM is transferred back into the RAM. Thus EEPROM is used for temporary data storage during power failure.

The $\overline{\text{STORE}}$ and $\overline{\text{RECALL}}$ signals are used to store the data in EEPROM and transfer it back into RAM, respectively. The main advantage of NVRAM is that the battery backup is not required to save the working data in case of power failure.

The important data such as time, date, monitor setting, and other configurations are normally stored in the CMOS static RAM or NVRAM of a computer.

10.9.6 Commonly Used ICs for ROM

Some commonly used ICs for ROM are listed below.

IC number	Description
7488	32 × 8 bit ROM
74187	256 × 4 bit ROM
2316	2K×8 bit ROM
2332	4K×8 bit ROM
2364	8K×8 bit ROM
65256	32K×8 bit ROM
23512	64K×8 bit ROM
531000	128K×8 bit ROM
74186	64 × 8 bit PROM
2708	1K×8 bit EPROM
2716	2K×8 bit EPROM
2732	4K×8 bit EPROM
2764	8K×8 bit EPROM
2816	2K×8 bit EPROM
27128	16K×8 bit EPROM
27256	32K×8 bit EPROM
27512	64K×8 bit EPROM
571000	128K×8 bit EPROM

SUMMARY

- It is possible to expand the word size of a memory by connecting two or more ICs in parallel.
- Connecting two or more ICs in series can increase the word capacity of a memory.
- On the basis of principle of operations, memories are classified as (a) sequential memories, (b) read and write memories, (c) read only memories, and (d) content addressable memories.
- On the basis of physical characteristics, memories are classified as (a) erasable or non-erasable memories and (b) volatile or non-volatile memories.
- On the basis of accessing mode, memories are classified as (a) sequential access memories and (b) random access memories.
- On the basis of fabrication technology, memories can be classified as (a) bipolar memories and (b) unipolar memories.
- ROM is further classified as (a) masked ROM, (b) programmable ROM, (c) erasable and programmable ROM, and (d) electrically erasable programmable ROM.

KEY TERMS

Memory A memory unit is a collection of storage cells with associated circuits needed to transfer data in and out of the device. The data is stored in memory in terms of groups of binary bits called words.

Write cycle time (t_{wc}) It is the minimum amount of time for which the valid address is present for writing the data in memory location. It is in the order of 200 ns.

Write pulse time (t_w) It is the minimum duration of the write pulse. It is in the order of 120 ns.

Write release time (t_{WR}) It is the minimum amount of time for which the address is valid before the write pulse.

Data set up time (t_{DW}) It is the minimum amount of time for which the data is valid before the write pulse ends. It is in the order of 120 ns.

Data hold time (t_{DH}) It is the minimum amount of time for which the data is valid after the write pulse ends.

Read operation It is nothing but reading the data from a selected memory location.

Read cycle time (t_{RC}) It is the minimum amount of time for which the valid address is present for reading a data from the memory location. It is in the order of 200 ns.

Access time (t_A) It is the time required to access the data from its memory location. It is the maximum time from the start of valid address of the read cycle to the availability of valid data at the data outputs. It is in the order of 200 ns.

Read to output active time (t_{RDX}) It is the minimum time required to enable the output buffer after the beginning of the read pulse. It is in the order of 20 ns.

Read to output valid time (t_{RD}) It is the maximum delay time between the start of the read pulse and the availability of valid data at the data outputs. It is in the order of 70 ns.

IC select to output valid time (t_{CO}) It is the maximum time delay between the start of an IC select signal and the availability of valid data outputs. It is in the order of 70 ns.

IC select to output active time (t_{CX}) It is the maximum time delay between the start of an IC select signal and the output buffer coming to active state. It is in the order of 20 ns.

Output tristate from read (t_{TOTD}) It is the maximum time delay between the end of the read pulse and the output buffers going to tristate. It is in the order of 60 ns.

Data hold time (t_{OHA}) It is the minimum time for which the valid data is available at the data output after the address ends. It is in the order of 50 ns.

Random access In random access mode, any memory location can be accessed in any order in the same amount of time. Semiconductor memories are the example of random access memories.

Sequential access In sequential access mode, memories can be accessed in a predefined sequence.

Direct access Direct access mode is also referred as semi-random mode. In certain cases, the information is accessed neither randomly nor in sequence, but something in between.

Access time It is the time from the valid read/write signal to the data available or written at the desired location. The access time depends on the physical characteristics and the accessing mode used for that device.

Read-write memory It supports both read and write operations.

Read only memory (ROM) ROM allows the user to read the data, but write operation is denied.

Content addressable memory It is a type of RAM where the locations are identified by their content.

Erasable memory Memory is said to be erasable, if its content can be erased and new data can be written.

Volatile memory Memory is said to be volatile in which the content of memory is lost when the power is switched off. Read-write memory is the example of volatile memories.

Static RAM In static RAM, the content does not change with time and the data is stored in terms of voltage. Flip-flop is the basic cell of static RAM.

Dynamic RAM It is the highest density RAM. Capacitor is the basic cell of dynamic RAM.

EXERCISE

Review Questions

1. Explain the functional diagram of a 2K×8 memory chip.
2. Draw the WRITE cycle waveform and define the following terms: Write Cycle Time, Write Pulse Time, Write Release Time, Data Set Up Time, and Data Hold Time.
3. Draw the READ cycle waveforms and define the following terms: Read Cycle Time, Access Time, Read To Output Active Time, Read To Output Valid Time, IC Select To Output Valid Time, IC Select To Output Active Time, Output Tristate From Read, and Data Hold Time.
4. With suitable example, explain expanding the word size and word capacity of the memory.
5. Explain the classification of memory chips.
6. Write notes on (a) sequential memory, (b) TTL bipolar RAM cell, and (c) dynamic RAM cell.
7. Write notes on EEPROM and NVROM.
8. With the help of a circuit diagram, explain the read and write operation of a TTL bipolar RAM cell.
9. With the help of a circuit diagram, explain the read and write operation of a dynamic RAM cell.
10. Differentiate static RAM and dynamic RAM.
11. Explain the 16×4 masked ROM memory organization.

Multiple-Choice Questions

1. As compared to MOS memories, bipolar memories have
   (a) Slower access times, but are cheaper
   (b) Slower access time, and are costly
   (c) Faster access time, and are cheaper
   (d) Faster access time, and are costly

2. The data bus width of 2048 bits is
   (a) 8 (b) 10 (c) 11 (d) 16
3. Four memory chips of 16×4 size have their address bus connected together. The system will be of size
   (a) 64×64 (b) 16×16 (c) 32×8 (d) 256×1
4. Each cell of a static random access memory contains
   (a) 6 MOS transistors
   (b) 4 MOS transistors and two capacitors
   (c) Two 2-input NORs and 1 EX-NOR gate
   (d) EX-OR gate and shift register
5. A dynamic RAM consists of
   (a) 6 transistors (b) 2 transistors and 2 capacitors
   (c) 1 transistor and 1 capacitor (d) 2 capacitors only
6. Which of the following is an example of volatile memory?
   (a) ROM (b) RAM (c) PROM (d) Hard-disk
7. The basic memory cell of a dynamic RAM consists of
   (a) a capacitor (b) a transistor
   (c) a flip-flop (d) a transistor acting as a capacitor
8. RAM is
   (a) Fastest memory (b) Slowest memory
   (c) Random memory (d) Write-only memory
9. A MOSRAM has
   (a) Less memory density
   (b) Large memory density
   (c) Less area, less head dissipation, and less access time of 1 ms.
   (d) Both (b) and (c)
10. How many MOS transistors are required to implement a typical cell for a dynamic RAM?
    (a) Six (b) Five (c) One (d) Two

Answers

1. (d)	2. (c)	3. (b)	4. (a)
5. (c)	6. (b)	7. (d)	8. (c)
9. (d)	10. (c)		

INDEX

access modes 641
  direct access 641
  random access 641
  sequential access 641
active pull-up 17
ADC
    584, 599, 601, 602, 603, 605, 607
  continuous type 602
  counter type 601
  dual slope 605
  parallel comparator 599
  specifications for 607
  successive-approximation 603
adders 202
  carry look-ahead adder 205
  full-adder 202
  half-adder 202
  parallel adder 204
  algebraic simplification 145
  alphanumeric codes 110
  AND matrix 531
  AND operation 131
  arithmetic 81, 84
    1's complement 81
    2's complement 84
  arithmetic logic unit 226
  arithmetic overflow 91
  ASCII code 111
  ASICs 561, 562
    full-custom 561
    programmable 561
    semi-custom 561
    standard cell-based 562
ASM charts 464
asynchronous counter 346, 347
  modulus-M 347
  up/down 346
asynchronous sequential circuits 431, 432, 440
  fundamental mode 432
  pulse mode 440
auto-zero capacitor 616

BCD 94, 96
  addition 94
  subtraction 96
BCD adder 219
BCD code 93,102
  4-bit 102
  5-bit 102
  biquinary 103
BCD subtractor 220
binary 78, 79
  addition 78
  division 79
  multiplication 79
  subtraction 78
  arithmetic 77
binary number 68
bits 51
  boolean algebra 129, 133
  boolean theorems 133
    associative law 134
    commutative law 134
    distributive law 134
  branch statements 483
    conditional branch 484
    unconditional branch 484
byte 51

clocked S-R flip-flop 297
clocked sequential circuit 371, 375
  analysis of 371
  design of 375
CMOS 37, 38, 39
  characteristics of 39
  inverter 37
  NAND gate 38
  NOR gate 39
code 101, 103
  2-4-2-1 101
  biquinary 103
  excess-3 103
code conversion 106, 108
  binary-to-gray 106
  gray-to-binary 108

code converter 264, 276
  BCD to seven-segment 276
codes 92
  alphanumerical 93
  binary 51
  classification of 92
  error detecting and correcting 93
  self-complementary 92
  unit distance 93
  weighted 92
comparator 229
complement binary arithmetic 81
conditional box 459
connection operation statement 483
conversion
    53, 54, 58, 59, 60, 63, 64, 65
  binary to decimal 53
  binary to hexadecimal 65
  binary to octal 60
  decimal to binary 54
  decimal to hexadecimal 63
  decimal to octal 58
  hexadecimal to binary 64
  hexadecimal to decimal 63
  octal to binary 59
  octal to decimal 58
converter
    264, 268, 270, 271, 273, 274, 335
  BCD to excess-3 270
  BCD-to-binary 268
  binary to BCD 264
  binary-to-gray code 273
  excess-3 to BCD 271
  gray-to-binary code 274
  parallel-to-serial 335
  serial-to-parallel 335
CPLD 551, 552
  architecture 551
  family 552
current hogging 10
cycles 449

D flip-flop 303
DAC 584, 585, 586, 590, 593, 594
  basic principle of 585
  R/2R ladder network 586, 590
  resistor divider 586
  specifications of 594
data objects 511
  constant 511
  signal 511
  variable 511
decision box 458
DeMorgan's theorem 140
demultiplexer 257, 258
differential input 614
differential reference 615
digital circuits 198
  combinational logic circuit 198
  sequential logic circuit 198
digital IC 4
diode-transistor logic 11
direct-coupled transistor logic 9
don't-care condition 178
dynamic RAM 647, 650

EBCDIC 111
emitter-coupled logic 29
encoding 598
error-correcting codes 114
error-detecting codes 112
  block parity codes 113
  hamming code 118
  linear block code 115
  parity codes 112
excitation table 356, 357
  D flip-flop 357
  J-K flip-flop 356
  R-S flip-flop 356
  T flip-flop 357

flip-flop 304, 306
  characteristics of 306
  conversions 307
  edge-triggered 304
  hold-up time 306
  maximum clock frequency 306
  propagation delay 306
  set-up time 306
  excitation table 357
flip-flops 323
  applications of 323
  bounce elimination switch 323
  counters 323
  random access memory 323
  registers 323
FPGA 555, 558
  architecture 555
  XC 4000 series 558
  Xilinx 555

gates 137
  AND gate 137
  EX-NOR gate 137
  EX-OR gate 137
  NAND gate 137
  NOR gate 137

NOT gate 137
OR gate 137
generic array logic 549
gray code 106

handshake mode 619
hazards 450
dynamic hazard 450, 452, 454
essential hazard 452
static hazard 450, 452

I2L 32, 33
NAND gate 33
NOR gate 33
I2L inverter 32
identifiers 510
input buffer 531
integrated circuits 1
integrating capacitor 615
integrating resistor 615
interfacing 624
direct mode 625
handshake mode 625
invert/non-invert matrix 532

J-K flip-flop 299, 301
master-slave 301
Johnson counter 336

Karnaugh map simplification 148

libraries 510
lockout condition 398
logic family 1, 2
bipolar 2
unipolar 2
logical operation statement 481
ANDing operation 482
inversion operation 481
ORing operation 482

main memory 630
maxterm 143
Mealy circuit 370
memory 631, 632
functional diagram 631
operations 632
memory 642, 643
bipolar 643
content addressable 642
erasable or non-erasable 643
random access 643
read and write 642
read only 642
sequential access 643
unipolar 643
volatile or non-volatile 643
memory cell 630
memory size 635
minterm 142
modeling 493, 498, 502, 507
behavioural 493, 502
data flow 493, 498
mixed style of 493, 507
Moore circuit 369
MOSFET logic 34
multiplexer 241, 242, 244

nibble 51
NMOS 34, 35, 37
noise margin 26
NOT operation 132
number system 52, 53
binary 52
decimal 52
hexadecimal 52
octal 53

OR matrix 532
OR operation 131
output buffer 533

PAL 539, 540, 542
parity checker 235
even parity 235
odd parity 237
parity generator 233, 239
even parity 233
odd parity 234
passive pull-up 14
phases 614
auto-zero 614
de-integrate 614
signal integrate 614
PLA 529, 530
PLD circuits 550, 551
bipolar 550
CMOS 550, 551
prime implicants 185
product of sums 142

quantization 598
error 598
level 598
Quine-McCluskey method 185

race-around condition 300
races 449
critical races 449
non-critical 449
read only memory 630
read operation 634
access time 634
data hold time 635
output tristate from read 635
read cycle time 634
read-write memory 630
readout 642
destructive 642
non-destructive 642
reference capacitor 616
reference voltage 616
register transfer language 478
register transfer statement 479
rotate operation 481
shift operation 480
resistor-transistor logic 8
ring counter 335
ringing effect 19
ripple counter 343
ROM 652, 655, 657, 659
electrically erasable programmable 659
erasable programmable 657
masked 652
programmable 655
sample-and-hold circuit 595
secondary memory 630
sequence detector 340, 406
sequence generator 338, 401
sequential circuit 293
seven-segment code 109
shift register 327, 329, 332, 333, 334
bidirectional 333
PISO 329
PIPO 332
SIPO 329
SISO 327
sink current 18
source current 18
standard TTL 26
state assignment 379
state box 457
state diagram 371, 374, 376
state machine 457
state reduction 378
state table 371, 377
statement 484, 485
bus connection 485
conditional transfer 484
count 485
declaration 485
statement 503, 504, 505, 506
case 504
exit 506
if 503
loop 505
next 506
wait 506
static RAM 644, 650
subtractor 211
half-subtractor 211
parallel subtractor 213
sum of products 141, 156
switch matrix 555
synchronous counter 355

T flip-flop 303
totem-pole output 15
transistor 3
switch OFF 3
switch ON 3
transmission gates 41
tri-state inverter 22
tri-state logic 22
tri-state TTL 21
TTL 13, 27
fast 27
high speed 27
low power 27
low-power Schottky 27
Schottky 27
families 29
parameters 23
twisted ring counter 336

unit distance code 106
universal register 334

VHDL 491, 512

wired-AND logic 20
wired-OR logic 31
word capacity 637
word size 636
write operation 633

DIGITAL ELECTRONICS

Digital Electronics is specially designed as a textbook for undergraduate students of electronics, communication, computer science, and electrical and instrumentation engineering for an introductory course on digital electronics or digital system and design. The book will also be useful for students pursuing postgraduate courses in electronics and computer science.

Beginning with the fundamental concepts such as logic families, number systems and codes, Boolean algebra and logic gates, and combinational circuits, the book moves on to cover the applied aspects such as sequential logic, asynchronous sequential circuits, algorithmic state machines, programmable logic devices, converters, and semiconductor memories.

Through its lucid presentation, the book seeks to impart a thorough understanding of the basic concepts of digital electronics and its applications.

Key Features

- Establishes a balance between the theoretical concepts of digital electronics and its applications
- Includes a large number of solved examples and numerous illustrations supporting the text for better understanding of complex topics
- Includes separate chapters on the number system, synchronous and asynchronous circuits, and K-maps
- Includes plenty of multiple choice questions and review questions for practice
- Provides a glossary of the important terms at the end of each chapter

Dr G. K. Kharate is currently Principal, Matoshri College of Engineering and Research Centre, Nashik. A Ph D from the University of Pune, Dr Kharate has more than 20 years of teaching experience. He is also a fellow member of the Institution of Electronics and Telecommunication Engineers (IETE) and a life member of many other professional bodies of repute like the Indian Society for Technical Education (ISTE), the Institution of Engineers (India), and the Computer Society of India.

Dr Kharate has also published a number of articles in national and international journals of repute and organized several conferences and workshops in his areas of research.

PUBLISHING IN INDIA SINCE 1912 ~ 100 YEARS

OXFORD
UNIVERSITY PRESS

www.oup.com

ISBN 0-19-806183-8

9 780198 061830

₹ 435

DIGITAL ELECTRONICS

KHARATE



OXFORD