

Digital Systems

Principles and Design

ANNA UNIVERSITY



Includes
Solved University
Model Question
papers

Raj Kamal

DIGITAL SYSTEMS

Principles and Design

This page is intentionally left blank.

DIGITAL SYSTEMS

Principles and Design

Raj Kamal

Senior Professor

School of Computer Sciences

Institute of Computer Sciences and Electronics

Devi Ahilya University

Indore

PEARSON

Chennai • Delhi • Chandigarh

Copyright © 2012 Dorling Kindersley (India) Pvt. Ltd

Licensees of Pearson Education in South Asia

No part of this eBook may be used or reproduced in any manner whatsoever without the publisher's prior written consent.

This eBook may or may not include all assets that were part of the print version. The publisher reserves the right to remove any material present in this eBook at any time.

ISBN 9788131768792

eISBN 9788131798904

Head Office: A-8(A), Sector 62, Knowledge Boulevard, 7th Floor, NOIDA 201 309, India

Registered Office: 11 Local Shopping Centre, Panchsheel Park, New Delhi 110 017, India

*Dedicated to my students at Devi Ahilya University, Indore,
and at Arulmigu Kalasalingam College of Engineering, Krishnankoil*

This page is intentionally left blank.

CONTENTS

<i>Preface</i>	xvii
<i>Acknowledgements</i>	xxi
1. Basic Digital Concepts	1.1
1.1 Concepts of ‘1’s, ‘0’s	1.1
1.1.1 Positive Logic	1.1
1.1.2 Negative Logic	1.2
1.1.3 Popular Representations of the Digital Circuits	1.2
1.2 Analog vs. Digital Circuits	1.2
2. Boolean Algebra and Theorems, Minterms and Maxterms	2.1
2.1 The NOT, AND, OR Logic Operations	2.1
2.1.1 The NOT Logic Operation	2.1
2.1.2 The AND Logic Operation	2.2
2.1.3 The OR Logic Operation	2.3
2.2 The NAND and NOR Logic Operations	2.3
2.2.1 NAND Gate	2.3
2.2.2 NOR Gate	2.4
2.3 The XOR, NOT-XOR, NOT-NOT Logic Operations	2.4
2.3.1 XOR Logic Operation	2.4
2.3.2 NOT-XOR (XNOR) Logic Operation	2.4
2.3.3 NOT-NOT Logic Operation	2.5
2.4 Boolean Algebraic Rules (for Outputs from the Inputs)	2.5
2.4.1 OR Rules	2.5
2.4.2 AND Rules	2.5
2.4.3 NOT Rules (Rules of Complementation)	2.6
2.5 Boolean Algebraic Laws	2.6
2.5.1 Commutative Laws	2.6

2.5.2	Associative Laws	2.6
2.5.3	Distributive Laws	2.6
2.6	Demorgan Theorems	2.6
2.7	The Sum of the Products (SOPs) as per Boolean Expression and Minterms	2.8
2.7.1	SOPs for Two Variables (Two Inputs) Case	2.8
2.7.2	SOPs for Three Variables (Three Inputs) Case	2.9
2.7.3	SOPs for Four Variables (Four Inputs) Case	2.10
2.7.4	Conversion of a Boolean Expression or Truth Table Outputs into the Standard SOP Format	2.12
2.8	Product of the Sums and Maxterms for a Boolean Expression	2.12
2.8.1	POS for Two Variables (Two Inputs) Case	2.13
2.8.2	POS for Three Variables (Three Inputs) Case	2.14
2.8.3	POS for Four Variables (Four Inputs) Case	2.14
2.8.4	Conversion of a Boolean Expression into Standard POS Format	2.16
3.	Karnaugh Map and Minimization Procedures	3.1
3.1	The Three-Variable Karnaugh Map and Tables	3.1
3.1.1	Karnaugh Map from the Truth Table	3.1
3.1.2	Karnaugh Map from the Minterms in a SOP	3.3
3.1.3	Karnaugh Map from the Maxterms in a POS	3.4
3.2	Four Variable Karnaugh Map and Tables	3.5
3.2.1	Karnaugh Map from the Truth Table	3.5
3.2.2	Karnaugh Map from the Minterms in an SOP	3.7
3.2.3	Karnaugh Map from the Maxterms in a POS	3.8
3.3	Five and Six Variable Karnaugh Maps and Tables	3.10
3.4	An Important Feature in the Design of a Karnaugh Map	3.11
3.4.1	Only Single Variable Changes into Its Complement in a Pair of Adjacent Cells	3.11
3.4.2	Only Two Variables Change into Their Complements in Adjacent Cells in a Square or Column of Four Cells	3.12
3.4.3	Three Variables Change into Their Complements in Adjacent Cells in Box of Eight Adjacent Cells	3.13
3.4.4	First and Last Columns for First and Last Rows and Purpose of Deciding Adjacency in a Karnaugh Map	3.13
3.4.5	Use of Don't Care (or Unspecified) Input Conditions for Purpose of Deciding Adjacencies in a Karnaugh Map	3.14
3.5	Simplification of Logic Circuit Relation by Minimization Using Adjacencies	3.15
3.5.1	Minimization of a Karnaugh Map Using Pairs of Adjacent Cells	3.15
3.5.2	Minimization of a Karnaugh Map Using Quads of Four Adjacent Cells	3.16
3.5.3	Minimization of a Karnaugh Map Using Octet of Eight Adjacent Cells	3.17

3.5.4 Minimization of a Karnaugh Map Using Offset Adjacencies and Diagonal Adjacencies	3.18
3.5.5 Minimization by Finding Prime Implicants	3.18
3.6 Drawing of Logic Circuit Using AND-OR Gates, OR-AND Gates, NAND's Only, NOR's Only	3.20
3.7 Representations of a Function (Cover) for a Computer-aided Minimization for Simplifying the Logic Circuits	3.21
3.7.1 Representation in Cube Format for Computer-aided Minimization	3.21
3.7.2 Representation in Four-Dimensional Hypercube Formats for a Computer-aided Minimization	3.26
3.7.3 Representation in Hypercube (Multi-dimensional Cube) Formats for Computer-aided Minimization	3.27
3.8 Multi-Output Simplification	3.28
3.8.1 Prime Implicants for Multi-Outputs Case	3.29
3.9 Two Outputs Simplification—Computer-Based Prime Implicants Using Star Product and Sharp Operations	3.31
3.9.1 Combination of Two Cubes Differing in One Variable into One Cube—A Star Product Operation	3.31
3.9.2 Finding Essential Prime Implicants Using Two Cubes—A Sharp Operation	3.32
3.9.3 Computer-Based Minimization Method to Find Minimum Required Cover (SOP function implicants)	3.33
3.10 Computer-Based Minimization—Quine-McCluskey Method	3.34
3.10.1 Quine-McCluskey Method of Finding Prime Implicants	3.34
3.10.2 Finding Minimal Sum from the Prime Implicants for an Output	3.37
3.10.3 Finding Minimal Sum for the Multi-Output Case Using Quine-McCluskey Method	3.38
4. Binary Arithmetic and Decoding and Mux Logic Units	4.1
4.1 Binary Arithmetic Units	4.1
4.1.1 Binary Addition of Two Bits	4.1
4.1.2 Addition of Two Arithmetic Numbers Each of 4 Bits	4.2
4.1.3 Subtraction of Two Arithmetic Numbers Each of 4 Bits	4.5
4.2 Decoder	4.7
4.2.1 Decoder (Line Decoder)	4.7
4.2.2 The 1 of 2 and 1 of 4 Line Decoders	4.9
4.2.3 The Four-line to 16-line Decoder	4.9
4.2.4 Function Specific Decoders	4.9
4.3 Encoder	4.12
4.3.1 Encoder (Line Encoder)	4.12
4.3.2 Encoder (Priority Encoder)	4.13
4.3.3 BCD 10 of 1 Four-bit Encoder	4.14
4.3.4 Octal 8 of 1 Three-bit Encoder and Hexadecimal Encoder	4.14
4.4 Multiplexer	4.15
4.4.1 Multiplexer (Line Selector)	4.15
4.4.2 Multiplexer with Outputs Enabling Control (gate) Pin(s)	4.16

4.5 Demultiplexer	4.20
4.5.1 Demultiplexer Definition	4.20
5. Code Converters, Comparators and Other Logic Processing Circuits	5.1
5.1 Code Converters	5.1
5.1.1 Codes for Decimal Numbers	5.1
5.1.2 Unit Distance Code Converter	5.3
5.1.3 ASCII (American Standard Code for Information Interchange) for the Alphanumeric Characters	5.4
5.2 Equality and Magnitude Comparators Between Two Four-bit Numbers	5.5
5.3 Odd Parity and Even Parity Generators	5.7
5.4 The 4-bit AND, OR, XOR Between Two Words	5.8
5.4.1 AND	5.8
5.4.2 OR	5.9
5.4.3 XOR	5.9
5.4.4 Test	5.9
6. Sequential Logic, Latches and Flip-Flops	6.1
6.1 Flip Flop and Latch	6.2
6.2 Sr Latch (Set-Reset Latch) Using Cross Coupled NANDs	6.3
6.2.1 SR Latch at Various Input Conditions	6.4
6.2.2 Difficulties in Using an SR Latch	6.6
6.2.3 Timing Diagrams of an SR Latch	6.6
6.2.4 Level Clocked SR Latch	6.8
6.3 JK Flip-Flop	6.9
6.3.1 Explanation of the State Table for the Logic Circuit of an Edge-Triggered JK FF	6.10
6.4 T Flip-Flop	6.12
6.4.1 T Flip-Flop with Clear and Preset	6.15
6.5 D Flip-Flop and Latch	6.16
6.5.1 D Flip-Flop	6.16
6.5.2 D Flip-Flop with Clear and Preset	6.18
6.5.3 D Latch	6.18
6.6 Master-Slave RS Flip-Flop	6.19
6.7 Master-Slave (Pulse Triggered) JK Flip-Flop	6.22
6.7.1 MS JK Flip-Flop with Clear and Preset	6.24
6.8 Clock Inputs	6.25
6.8.1 Level Clocking of a Clock Input	6.25
6.8.2 Edge Triggering at a Clock Input	6.25
6.9 Pulse Clocking of the Latches in the Flip-Flops	6.26
6.10 Characteristic Equations for the Analysis	6.26
7. Sequential Circuits Analysis, State Minimization, State Assignment and Circuit Implementation	7.1
7.1 General Sequential Circuit with a Memory Section and Combinational Circuits at the Input and Output Stages	7.2

7.2 Synchronous and Asynchronous Sequential Circuits	7.2
7.2.1 Synchronous Sequential Circuit	7.2
7.2.2 Asynchronous Sequential Circuits	7.3
7.3 Clocked Sequential Circuit	7.3
7.4 Classification of Sequential Circuit as Moore and Mealy State Machine Circuits	7.3
7.4.1 Classification of a Sequential Circuit as Moore Model Circuit	7.3
7.4.2 Classification of a Sequential Circuit as Mealy Model Circuit	7.4
7.5 Analysis Procedure	7.4
7.5.1 Excitation Table	7.5
7.5.2 Transition Table	7.7
7.2.3 State Table	7.7
7.5.4 State Diagram	7.8
7.6 Conditions of States Equivalency	7.8
7.6.1 State Reduction and Minimization Procedure	7.9
7.6.2 Assignment of Variables to a State	7.11
7.7 Implementation Procedure	7.12
8. Sequential Circuits for Registers and Counters	8.1
8.1 Registers	8.1
8.1.1 Bi-stable Latches as the Register	8.3
8.1.2 Parallel-In Parallel-Out Buffer Register	8.3
8.1.3 Number of Bits in a Register	8.4
8.2 Shift Registers	8.4
8.2.1 Serial-In Serial-Out (SISO) Unidirectional Shift Register	8.5
8.2.2 Serial-In Parallel-Out (SIPO) Right Shift Register	8.6
8.2.3 Parallel-In Serial-Out (PISO) Right Shift Register	8.8
8.3 Counter	8.9
8.4 Ripple Counter	8.12
8.4.1 Cascaded Divide-By- 2^n Circuit as a Ripple Counter	8.12
8.4.2 Modulo-6, Modulo-7 and Modulo-10 Counters	8.14
8.4.3 Ring Counter	8.15
8.4.4 Johnson Counter (Even Sequences Switch Tail or Twisted Ring Counter)	8.16
8.4.5 Odd Sequencer Johnson Counter (Odd Sequencer Switch Tail or Twisted Ring Counter)	8.18
8.5 Synchronous Counter	8.19
8.5.1 Synchronous Counter Using Additional Logic Circuit	8.19
8.6 Asynchronous Clear, Preset and Load (JAM) in a Counter	8.20
8.7 Synchronous Clear, Preset and Load Facilities in a Counter	8.20
8.8 Timing Diagrams	8.21
9. Fundamental Mode Sequential Circuits	9.1
9.1 General Asynchronous Sequential Circuit	9.2
9.2 Unstable Circuit Operation	9.2
9.3 Stable Circuit Asynchronous Mode Operation	9.4

9.4	Fundamental Mode Asynchronous Circuit	9.4
9.4.1	Tabular Representation of Excitation-cum-Transitions of States and Outputs	9.5
9.5	Analysis Procedure	9.7
9.5.1	Excitation Table	9.8
9.5.2	Transition Table	9.8
9.5.3	State Table	9.9
9.5.4	State Diagram	9.10
9.5.5	Flow Table	9.12
9.5.6	Example of an Excitation-cum-Transition Table	9.13
9.5.7	Flow Table from Excitation-Transition Table	9.13
9.5.8	Flow Diagram	9.14
9.6	Races	9.15
9.6.1	Cycles of the Races	9.17
9.7	Race-Free Assignments	9.18
10.	Hazards and Pulse Mode Sequential Circuits	10.1
10.1	Hazards	10.2
10.1.1	Static-0 Hazard	10.3
10.1.2	Static-1 Hazard	10.4
10.2	Identifying Static Hazards	10.5
10.2.1	Identification from the Boolean Expressions	10.6
10.2.2	Identification from the Karnaugh Map (Only One-variable Input Case)	10.6
10.2.3	Identification from the Karnaugh Map (Three-Variable Input)	10.8
10.2.4	Detecting Absence of Static 1 Hazard from the POS Form of Boolean Expression	10.9
10.2.5	Detecting Absence of Static 0 Hazard from the SOP Form of Boolean Expression	10.10
10.3	Eliminating Static Hazards	10.10
10.4	Dynamic Hazards	10.12
10.5	Hazards Free Circuits	10.13
10.6	Essential Hazards	10.13
10.7	Pulse Mode Sequential Circuit	10.15
11.	Implementation of Combinational Logic by Standard ICs and Programmable ROM Memories	11.1
11.1	Standard ICs for Design Implementation	11.2
11.1.1	Adder/Subtractor IC and Magnitude Comparator	11.2
11.1.2	Decoder IC	11.3
11.1.3	Encoder IC	11.3
11.1.4	Multiplexer IC	11.4
11.2	Programming and Programmable Logic Memories	11.4
11.2.1	ROM (Pre-Programmed Read Only Memory) and PROM (Programmable Read Only Memory)	11.4

12. Implementation of Combinational Logic by Programmable Logic Devices	12.1
12.1 Basics Points to Remember When Using the PLDs (PROMs, PALs, PLAs)	12.1
12.2 PAL (Programmable Array Logic)	12.6
12.3 PLA (Programmable Logic Arrays)	12.9
13. Logic Gates	13.1
13.1 Revision of the Important Gates	13.1
13.2 Diode Circuit	13.3
13.3 Bipolar Junction Transistors and Mosfets	13.3
13.3.1 <i>N-P-N</i> Transistor Common Emitter Circuit	13.3
13.3.2 MOSFET Circuits	13.6
13.4 RTL, DTL, TTL Logic Gates	13.8
13.4.1 Resistor–Transistor Logic (RTL)	13.8
13.4.2 Diode–Transistor Logic (DTL)	13.12
13.4.3 Transistor–Transistor Logic (TTL)	13.14
13.4.4 TTL Other than NAND Gate	13.19
13.5 Emitter Coupled Logic (ECL)	13.19
13.5.1 ECL OR/NOR Gate	13.19
13.6 Integrated Injection Logic (I^2L)	13.21
13.6.1 I^2L Circuit Internal Connections	13.22
13.6.2 I^2L Circuit Working	13.22
13.6.3 I^2L Circuit Switching Speed, Delay Times and Power Dissipation	13.22
13.7 High Threshold Logic (HTL)	13.24
13.7.1 HTL Connections for the Output at F	13.24
13.7.2 Logic Operation for the Output at F	13.25
13.8 NMOS	13.25
13.8.1 NMOS Circuit Connections and Working	13.25
13.8.2 Calculation of Fan Out	13.26
13.8.3 Calculation of Propagation Delay	13.26
13.8.4 Calculation of Power Dissipation	13.26
13.8.5 NMOS Circuit Voltage Levels	13.27
13.8.6 Unconnected Input(s) not Permitted	13.27
13.9 CMOS	13.27
13.9.1 Importance and Features of CMOS Logic Circuits	13.27
13.9.2 Operations as Inverter (NOT), NOR and NAND	13.28
13.9.3 Calculation of Fan out	13.29
13.9.4 Calculation of Propagation Delay	13.29
13.9.5 Calculation of Power Dissipation	13.29
13.9.6 CMOS Circuit Voltage Levels	13.30
13.9.7 MOS Logic Circuits (CMOSs) and Their Relative Advantages with Respect to TTLs	13.30
13.10 Meanings of Speed, Propagation Delay, Operating Frequency, Power Dissipated per Gate, Supply Voltage Levels, Operational Voltage Levels that Define Logic States 1 and 0	13.30

13.11 Speed, Propagation Delay, Operating Frequency, Power Dissipated per Gate, Supply Voltage Levels, Operational Voltage Levels that Define Logic States ‘1’ and ‘0’ for Various Families of Gates	13.32
14. CPLDs and FPGAs	14.1
14.1 CPLDS	14.1
14.2 Registered PAL	14.2
14.3 Array Logic Cell	14.3
14.4 Field Programmable Gate Arrays (FPGAs)	14.5
15. VHDL—RTL Design, Combinational Logic, Data Types, and Operators	15.1
15.1 VHDL	15.1
15.1.1 VHDL Standard IEEE 1076	15.2
15.1.2 VHDL Standard IEEE 1164	15.3
15.1.3 VHDL Libraries	15.4
15.1.4 VHDL Identifiers, Keywords, and Comments	15.4
15.1.5 VHDL Data Objects	15.5
15.2 RTL Design	15.6
15.2.1 Data Flow Model	15.7
15.2.2 Port	15.7
15.2.3 Finite State Machine (FSM)	15.8
15.2.4 Entity in RTL Model	15.8
15.3 Behaviour Model for Process in an RTL Design	15.9
15.3.1 RTL Model Architecture	15.10
15.4 RTL Design for Combination Logic	15.11
15.5 Data Types	15.14
15.5.1 Subtypes	15.16
15.5.2 Array	15.16
15.5.3 Type Checking	15.17
15.6 Operators	15.17
16. VHDL—Packages, Sub Programs, and Sequential Circuits	16.1
16.1 Package	16.1
16.1.1 Package Declarations	16.2
16.1.2 Package Body	16.4
16.2 Subprograms	16.5
16.2.1 Procedure	16.6
16.2.2 Function	16.8
16.2.3 Attribute	16.9
16.3 Design Library	16.13
16.4 Sequential Circuits	16.14
16.4.1 General Sequential Circuit—Entity, Components, Architecture, and Processes	16.15
16.4.2 Synchronous Sequential Circuit	16.17
16.4.3 Sequencing Clock Circuit—Entity, Architecture and Processes	16.18
16.4.4 Clock inputs for Flip-flop and Latch Synchronous Sequential Circuits	16.19
16.4.5 Multiple Clock Signals from Main Clock	16.19

17. VHDL—Test Benches	17.1
17.1 Processes and Subprograms	17.2
17.1.1 Statements	17.2
17.1.2 Vectors	17.3
17.1.3 Conversions from a Data Type to Another	17.4
17.1.4 Now and Wait	17.5
17.1.5 Files	17.6
17.1.6 Events and Sensitivity List	17.6
17.1.7 Assertion, Report, and Severity Functions	17.7
17.1.8 Instantiation During Structural Modeling	17.7
17.2 Testing of Combinational and Sequential Circuits	17.8
17.2.1 Testing of a Combinational Circuit	17.8
17.2.2 Testing of a Sequential Circuit	17.11
17.3 Test Benches	17.13
18. VHDL—Examples of Modeling of Adder, Counter, Flip-Flop, Finite State Machine, Multiplexer, and Demultiplexer	18.1
18.1 Adder	18.1
18.1.1 Adder Circuit	18.1
18.1.2 Test Bench for the Adder	18.3
18.2 Counter	18.6
18.2.1 Counter Circuit	18.6
18.2.2 Test Bench for the Counter	18.7
18.3 Flip-Flop	18.10
18.3.1 D-Flip-Flop	18.10
18.3.2 Test Bench for the D-Flip-Flop	18.10
18.3.3 JK-Flip-Flop	18.13
18.3.4 Test Bench for the JK-Flip-Flop	18.14
18.4 Finite State Machine	18.18
18.4.1 Finite State Machine Sequential Circuit	18.18
18.4.2 Test Bench for the Finite State Machine	18.22
18.5 Multiplexer	18.25
18.5.1 Multiplexer Circuit	18.25
18.5.2 Test Bench for the Multiplexer (2:1)	18.26
18.6 Demultiplexer	18.29
18.6.1 Demultiplexer Circuit	18.29
18.6.2 Test Bench for the Demultiplexer (2:1)	18.30

This page is intentionally left blank.



PREFACE

Digital electronics is a core subject in electronics and communication engineering, electrical engineering, instrumentation, information technology and computer engineering. Recently, there has been a shift in emphasis towards teaching digital principles in the earlier semesters with the topics that are needed and that must be understood for studying VLSI design.

An engineering student takes a course in digital principles at a very early stage in his or her academic life. In addition, he or she may not be from an English-medium school, and therefore, may face great difficulties in comprehending the subject and in perfecting their problem-solving skills. This present book is targeted at such students.

This book fulfils the requirements of students by detailing the fundamental building blocks of digital design with a large number of examples and prepares the students to study VLSI circuit design in their eighth or ninth semesters. Each chapter focuses on a single aspect and highlights points that the reader must remember. The solved examples and practice exercises provided in each chapter will enable a reader to master the principles of digital circuits and design easily.

The organization of the book is as follows:

Chapter 1 the concepts of logic 1 and 0 and the positive and negative logic assignments to the circuit conditions. It explains the standard ways by which the 1's and 0's describe the circuit conditions. It provides a comparison of digital versus analog circuits. It also offers many exemplary circuits. A reader will thus easily comprehend the representation of a circuit input or output condition in terms of a 1 or a 0.

Chapter 2 explains Boolean expressions and laws, and how to use these to simplify a logic circuit. It explains the steps that are required for obtaining minterms or maxterms in a two or three or four variables' Boolean expression for obtaining the SOP and POS formats, respectively.

Chapter 3 has an innovative presentation of Karnaugh maps and their importance in simplifying a logic circuit for implementing a Boolean expression originally in an SOP or POS format. It describes how to find the prime implicants (minterms or maxterms). It includes advanced topics such as—multi-output minimization, Quine-McCluskey and hypercube-representation-based computer minimization techniques.

Chapter 4 explains the designs of half adder, full adder and subtractor. It then explains the important circuits—the decoder, encoder, multiplexer and demultiplexer, which are the building blocks of

many combinational circuits. Examples and practice exercises help the students improve their problem solving skills in *combinational circuit design*.

Chapter 5 explains the design of code converters, comparators for finding equality, greater-than and less-than, parity generators and multi bit AND, OR, XOR, and NOT operation circuits.

Chapter 6 initiates the description of sequential circuits. It describes the SR, D, JK, and master slave sequential circuits through circuits, timing diagrams and state tables. It gives the characteristic equations for these. A reader will learn by examples how to draw the timing diagrams for the change of states in these flip-flops and latches.

Chapter 7 gives the advanced digital principles—Mealy machines and Moore machine sequential circuits, state minimization, drawing the excitation, transition and state tables and showing the state diagrams. Sequential circuits are used in many applications. Must-learn concepts are described in a student-friendly manner. The large number of tables and diagrams in the examples will enable the reader to hone his or her state minimization problem solving skill.

Chapter 8 describes the practical sequential-circuit building blocks—registers and counters. The common designs, registers and counters are explained.

Chapter 9 features an advanced topic—asynchronous sequential circuits and their analysis in fundamental mode. It explains the races and race free assignments. A reader will improve his or her asynchronous race free design skills by learning from the tables, diagrams and examples given in the chapter.

Chapter 10 covers the hazards present in the logic circuits—static, dynamic and essential. It explains how to identify these and then obtain a hazard-free design.

Chapter 11 describes the use of standard integrated circuits (ICs) and use of ROMs and PROMs. It gives an innovative presentation by figures and examples of the PROMs. Each example and the corresponding figure shows which are the fused and intact OR links that implement a particular combination circuit in a PROM.

Chapter 12 describes the use of PAL, PROM and PLAs. An innovative presentation of the PAL, PROM and PLA is presented through figures and examples. The fused and intact AND or AND-OR links in a PAL or PLA are explained clearly to the reader.

Chapter 13 describes the different types and families of logic gates—RTL, DTL, TTL, I2L, ECL, HTL, NMOS and CMOS and the circuits for the NOTs, NANDs or NORs. The provided examples clarify the parameters of these gates; for example, fan-outs, propagation delays, voltage levels and current levels.

Chapter 14 gives an overview of the advanced programmable logic devices and the FPGAs that are programmed for specific applications used in the computer.

Chapter 15 introduces VHDL. Models, data types and operators, which are used in VHDL, are also described with simple examples.

Chapter 16 describes how to write package declarations and subprograms in the design unit. It deals with how the digital design libraries enable the use of design units in other designs. Chapter 16 introduces one to VHDL coding for sequential circuits.

Chapter 17 describes the ways of testing a combinational or sequential circuit. How to model test-benches is studied.

Chapter 18 gives examples of modeling of the adder using arithmetic addition and concatenation operators. We will also study examples of modeling of the counter, flip-flop, finite state machine, multiplexer and demultiplexer. We shall also learn how to write test benches for these entities and components.

Every effort has been made to provide precise information and correct solutions for the examples in the book. Despite this, errors may be still present. The author will be grateful to the readers for pointing these out to him.

The author will be grateful for any suggestions. These can be sent to professor@rajkamal.org. Students, please note that the PowerPoint slides and solutions to the exercises will be available on the Web soon. Queries are heartily welcome and can be sent through a query-sending link at the author's Web site <http://www.rajkamal.org>.

Raj Kamal

This page is intentionally left blank.

ACKNOWLEDGEMENTS

I am grateful to Dr M. S. Sodha, a renowned teacher, and Kalvivallal T. Kalasalingam, Chairman of Arulmigu Kalasalingam College of Engineering, Krishnankoil.

I am grateful to our Vice Chancellor Mr C. S. Chadha for his full support and cooperation. I am also thankful to my colleagues, particularly the young faculty of Computer and Electronics and the laboratory colleagues at the university.

I would like to acknowledge my gratitude to Dr Chelliah Thangaraj, Dr S. Radhakrishnan and Prof. G. Sudhakar of Arulmigu Kalasalingam College of Engineering, Krishnankoil.

I am especially thankful to Ms S. Alagu and Ms Suganthi Lakshmanan for thoroughly reading the manuscript and Mr Annathurai for making the AutoCAD drawings, and Mr S. Murugan and Mr Manickandan for assistance.

Finally, I would like to thank my family members—Ms Sushil Mittal, Shalin Mittal, Needhi Mittal, Dr Shilpi Kondaskar, Dr Atul Kondaskar and baby Arushi Kondaskar—for their full cooperation and for encouraging me to write this book.

This page is intentionally left blank.

CHAPTER 1

Basic Digital Concepts

OBJECTIVE

In this chapter, we shall learn the concept of a digital circuit and its logic states, ‘1’ and ‘0’. We shall also understand the differences between analog and digital circuits.

■ BASIC CONCEPTS

1.1 CONCEPTS OF ‘1’s, ‘0’s

An electronic circuit in which a state switches (changes) between two distinct states when there is a change in the input states or conditions is called a digital circuit. A state of a circuit means either a distinct region of output or input voltages or currents or frequencies or phases or conditions of a circuit. One of the regions is represented by a logic ‘true’ or 1 or ‘high’ or ‘yes’ state and the other region is represented by a logic ‘false’ or 0 or ‘low’ or ‘no’ state.

1.1.1 Positive Logic

Let us assume, a logic state 1 is represented by a voltage, V between 5 V and 2.8 V and a state 0 by V between 0.8 V and 0 V. (This is the case of *positive logic*, when the lower voltage represents ‘low’ or 0 state and the higher voltage represents ‘high’ or 1 state).

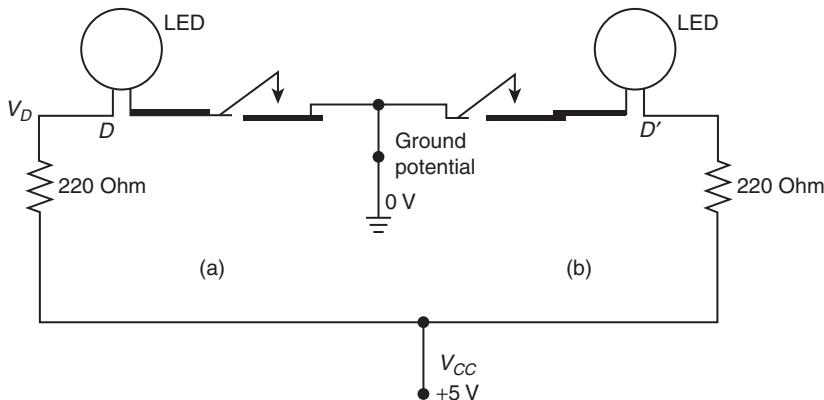


FIGURE 1.1 (a) A digital circuit consisting of a simple switch. (b) Another digital circuit consisting of a simple switch.

- (A) Consider a simple circuit given in Figure 1.1(a). When the switch is pressed, the voltage at D is $+5\text{ V}$ and when released, it is $\sim 0\text{ V}$. Logic state is ‘true’ or 1 when the switch is pressed and ‘false’ or 0 when released.
- (B) Now consider the circuit given in Figure 1.1(b). When the switch is pressed, the voltage at D' is $+0\text{ V}$ and when released, it is $\sim 5\text{ V}$. Logic state is ‘true’ or 1 when switch is released and ‘false’ or 0 when pressed.

1.1.2 Negative Logic

Now let us assume, a logic state 1 represented by a voltage, V between 0.8 V and 0 V and a state 0 by V between 5 V and 2.8 V . (This is called the case of ‘negative logic’, when a lower voltage represents ‘high’ or 1 state and the higher voltage represents ‘low’ or 0 state.)

- (A) Consider the circuit given in Figure 1.1(a). When the switch is pressed, the voltage at D is $+5\text{ V}$ and when released, it is $\sim 0\text{ V}$. Logic state is 0 when the switch is pressed and when released 1.
- (B) Now consider, the circuit given in Figure 1.1(b). When the switch is pressed, the voltage at D' is $\sim 0\text{ V}$ and when released, it is $\sim 5\text{ V}$. Logic state is 0 when the switch is released and 1 when pressed.

1.1.3 Popular Representations of the Digital Circuits

Table 1.1 shows two states, 1 and 0 in a few popular digital circuits.

1.2 ANALOG VS. DIGITAL CIRCUITS

Table 1.2 gives the advantages and disadvantages of a digital circuit vis-a-vis an analog circuit.

TABLE 1.1 1 and 0 state in a few popular digital circuits

Name of the corresponding representative digital circuit	Electronic state when logic state is called 1 in a digital circuit	Electronic state when logic state is called 0 in a digital circuit
TTL*	High +ve voltage (2.8 V to 5 V) between output and ground and low current ~ μ A sourcing from a circuit due to a 'OFF' output-stage transistor in a non-conducting state.	Low ~0 voltage (0 V to 0.8 V) between output (or input) and ground and high current ~mA sinking into a circuit due to the 'ON' output-stage transistor in a conducting state.
High Speed CMOS	High +ve voltage ($+5\text{ V} \pm 3.3\text{ V}$) between output (or input) and circuit supply ground.	Low +ve voltage ($+1.6\text{ V}$ to $+0\text{ V}$) between output (or input) and circuit supply ground.
RS232C	High -ve voltage (-3 V to -25 V) between output (or input) and ground and 'low current' through a circuit.	High +ve voltage ($+3\text{ V}$ to $+25\text{ V}$) between output (or input) and ground and 'low current' through a circuit.
Modem	Higher ~ 1240 Hz frequency at the output or input circuit.	Lower ~ 1040 Hz frequency at the output or input circuit.
Teletype loop	Higher ~ 16 to 20 mA current from the output to an input circuit.	Lower ~ 0 to 4 mA current from the output to an input circuit.

*Worst TTL case is $2.4\text{ V} \pm 0.4\text{ V}$ for logic 1 and $0.4\text{ V} \pm 0.4\text{ V}$ for logic 0. Note: Different families of logic circuits have different noise immunity than 0.4 V . The level of voltage changes that will not affect the input or output logic state is defined as the noise immunity.

TABLE 1.2 Advantages and disadvantages of a digital circuit vis-a-vis an analog circuit

Digital circuit		Analog circuit	
Advantages	Disadvantages	Advantages	Disadvantages
1. More close to logic and number systems and a logic state has a definiteness and thus the circuit inputs and outputs have precisionness and reliability.	1. Physical values are reflected by a complex digital to analog conversion circuit.	1. More close to physical system and physical values.	1. Logical values are reflected by a complex analog to digital conversion circuit.
2. Capabilities of logical decisions and arithmetic, logic and Boolean operations.	2. The circuits have high complexities. To represent a big decimal number, a large number of components are needed.	2. Capabilities of representing directly a physical value. For example, the value of temperature, wind, speed, etc can be represented by a corresponding voltage or current or frequency or phase.	2. A circuit shows deviation with time due to circuit's temperature changes or physical condition changes.

3. Noise in voltage or current does not matter because an output or input is measured in terms of its logic state(s), not in terms of a value of the voltage or current or frequency or phase.
3. Slower speed due to greater number of components needed to represent a state. For example, a bipolar junction transistor as inverter will act faster than a circuit in a TTL inverter.
3. Noise in voltage or current does matter because an output or input is measured in terms of a value of voltage or current or frequency or phase.
3. Faster speed due to lesser number of components needed to represent a physical value.

■ EXAMPLES

Example 1.1

Using the circuit in Figure 1.1, fill the Table 1.3 given below: (Write logic state 1 when $V_D > 2.8$ V and less than 5 V and 0 when $V < 0.8$ V. Write * when indeterminate).

TABLE 1.3

V_D	Logic state at D	V_D	Logic state at D
5 V	1	1 V	*
4 V	1	0.75 V	0
3.5 V	1	0.5 V	0
2 V	*	~0 V	0

Example 1.2

Consider a circuit in Figure 1.2. The circuit consists of two switches, SL and SU. One is at the ground floor and the other at the first floor. When the lamp is ON let us assume logic state = True and when the lamp is OFF let us assume logic state = False. Find the logic state in different conditions of the switches and show these in Table 1.4.

TABLE 1.4

SL	SU	Lamp	Logic state of the lamp
ON	ON	ON	1
OFF	OFF	OFF	0
ON	OFF	OFF	0
OFF	ON	OFF	0

Example 1.3

Consider the circuit in Figure 1.3. It consists of diodes, fused diodes (unconnected) and the resistances in the digital circuit outputs. It shows four inputs, Y_0 , Y_1 , Y_2 and Y_3 . What will the inputs at D_0 , D_1 when the output $Y_0 = 1$? Let 1 mean any voltage output between 2.8 V and 5 V and 0 mean any voltage between 0 and 0.8 V. Let 1 mean any input voltage between 2.4 V and 5 V. (Note: Assume worst-case 1 to be $2.4 \text{ V} \pm 0.4 \text{ V}$ and let worst-case 0 be $0.4 \text{ V} \pm 0.4 \text{ V}$). Assume except one all other

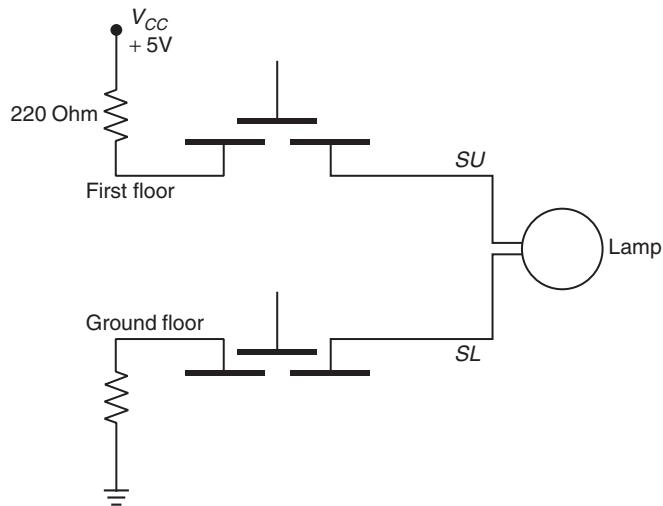


FIGURE 1.2 A circuit consisting of two switches SL and SU at the ground floor and first floor. When the lamp is on let us assume logic state = 1 and when the lamp is off let us assume logic state = 1.

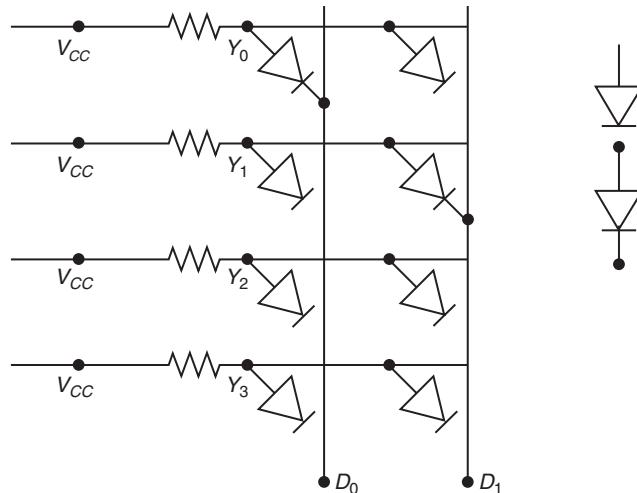


FIGURE 1.3 A digital circuit with four inputs, Y_0 , Y_1 , Y_2 and Y_3 and two outputs, D_0 and D_1 . Here, 1 means output voltage between 2.8 V and 5 V and 0 mean any output voltage between 0 and 0.8 V and 1 also means any input voltage between 2.4 V and 5 V. Supply operational voltage = ~ 5.0 V.

three inputs are unconnected. Supply operational voltage, $V_{CC} = 5.0 \pm 0.25$ V = 5.0 ± 0.25 V). A dot represents a junction.

Solution

- When $Y_0 = 1$, the diode connected to D_0 will be forward biased above the diode threshold voltage. Assume threshold voltage = 0.35 V, then voltage at

D_0 will be just 0.35 V less than the voltage at Y_0 . Since $Y_0 = 1$ (> 2.8 V) therefore input at $D_0 = 1$ (> 2.4 V).

- (ii) When $Y_0 = 1$, because the diode is not connected to D_1 , the voltage at D_1 will be ~ 0 V, therefore $D_1 = 0$ (< 0.8 V).

Example 1.4 Consider the circuit in Figure 1.3. What will the inputs D_0, D_1 when the output $Y_1 = 1$?

Solution

- (i) When the $Y_1 = 1$, the diode is not connected to D_0 , then the voltage at D_0 will be ~ 0 V, therefore $D_0 = 0$ (< 0.8 V).
- (ii) When the $Y_1 = 1$, the diode connected to D_1 will be forward biased above the diode threshold voltage. Assume threshold voltage = 0.35 V, then the voltage at D_1 will be just 0.35 V less than the voltage at Y_1 . Since $Y_1 = 1$ (> 2.8 V) therefore $D_1 = 1$ (> 2.8 V).

Example 1.5 Consider the circuit in Figure 1.3. What will the input D_0 when the $Y_1 = 1$ but the resistance R_0 from D_0 is fused (unconnected) or is of very high value ($>> 1$ M Ω)? (Note: Ω is a symbol for Ohm).

Solution

When the $R > 1$ M Ω the input, the $V_{CC} = 5$ V is not connected to D_0 , then the voltage at D_0 and Y_1 cannot be defined because there is no connection to the potential source through R . D_0 is at the indeterminate (indefinable) logic state. A state can only be defined if R is not disconnected.

Example 1.6 Consider third the thick pin in an electrical plug at the mains. It connects to the electrical earth connection of the laboratory. A wire is taken from it. Can it be called at logic state 0?

Solution

No, because it is a hanging connection, not connected to the ground potential and logic circuit power supply. Existence of merely the 0 V, does not mean logic state = 0.

Example 1.7 The circuit in Figure 1.3 consists of diodes, fused diodes and resistances. It shows four inputs, Y_0, Y_1, Y_2 and Y_3 to the diodes. What will the states at D_0 and D_1 when the supply voltage to the logic circuit suddenly falls from 5 V to 2.5 V?

Solution

Between 2.8 V and 5 V output only, state is 1 as per the definition given for the digital circuit in Figure 1.3. Since supply voltage is now 2.5 V, the input is 1 from the worst-case consideration ($V > 2.4$ V). Circuit will function improperly as a digital circuit whenever the supply voltage to the logic circuit falls such that an input Y_0 or Y_1 or Y_2 or Y_3 is less than 2.4 V.

Example 1.8

Assume that the circuit operational voltage $V_{DD} = 5.0 \pm 0.25$ V. Let us assume that in a logic circuit, 1 means any voltage between (2/3) V_{DD} and V_{DD} , and 0 mean any voltage between (1/3) V_{DD} and V_{SS} . Assume that the circuit is same as in Figure 1.3. What will D_0 and D_1 when the output $Y_0 = 1$ (>3.7 V)? Assume other outputs unconnected. (Assume voltages are with respect to V_{SS} and V_{SS} connects to supply ground potential).

Solution

- When $Y_0 = 1$, the diode connected to D_0 will be forward biased above the diode threshold voltage. Assume threshold voltage = 0.35 V, then the voltage at D_0 will be just 0.35 V less than the voltage at Y_0 . Since $Y_0 = 1$ (>3.7 V) therefore the input $D_0 = 1$ (>3.3 V). Note: When the $Y_0 = 1$ such that <3.7 V but >3.3 V, it will function improperly as a digital circuit because voltage drop across the diode will make the input D_0 below 3.33 V and thus less than 2/3 V_{DD} .
- When $Y_0 = 1$, the diode is not connected to D_1 , then voltage at D_1 will be ~ 0 V, therefore $D_1 = 0$ (<1.67 V).

Example 1.9

Let us assume that in a logic circuit, state 1 defines by a frequency 2250 Hz and 0 by frequency 2050 Hz. Assume that only Y_0 is connected and Y_1 , Y_2 and Y_3 are not connected. Assume that the circuit is same as in Figure 1.3. What will the D_0 and D_1 when the $Y_0 = 1$ (2250 Hz)?

Solution

- When $Y_0 = 1$ (means R connected to a 2250 Hz 1 V source in place of V_{CC}), the diode connected to D_0 will be forward biased when the input is above the diode threshold voltage. Frequency at D_0 will therefore also be 2250 Hz. Since input $Y_0 = 1$ (2250 Hz) therefore output $D_0 = 1$ (2250 Hz).
- When $Y_0 = 1$, the diode is not connected to D_1 , then voltage at D_1 will be indeterminate as no signal of either 2250 Hz or 2050 Hz is appearing at D_1 .

The circuit will function improperly and is not a digital circuit as output D_1 is indefinable. It just behaves as an analog circuit.

Example 1.10

Circuit in Figure 1.3 consists of diodes, fused diodes and resistances. It shows four inputs, Y_0 , Y_1 , Y_2 and Y_3 . What will the D_0 , D_1 when the $Y_0 = 1$? Assume a digital circuit with a negative logic in which 1 means any voltage input between 0 and 0.4 V and 0 mean any voltage output between 2.8 V and 5 V. Assume other outputs unconnected. Supply operational voltage $V_{CC} = 5.0 \pm 0.25$.

Solution

- When $Y_0 = 1$ (<0.4 V), the diode connected to D_0 will not be forward biased as the diode is at V below threshold voltage. Since $Y_0 = 1$ (<0.4 V) therefore input $D_0 = 1$ (<0.8 V).

- (ii) When $Y_0 = 1 (< 0.8 \text{ V})$, because the diode is not connected to D_1 , the voltage at D_1 will be $\sim 0 \text{ V}$, therefore $D_1 = 1 (< 0.8 \text{ V})$.

The logic states at D_0 and D_1 are (1, 1) in case of negative logic and (0, 1) in case of positive logic.

■ EXERCISES

- Using the circuit in Figure 1.1, fill the Table 1.5 given below: [Assume negative logic circuit and write logic state 1 when $V < 0.8 \text{ V}$ and 0 when $V > 2.8 \text{ V}$ to supply voltage. Write * when indeterminate.]

TABLE 1.5

V_D	Logic state at D	V_D	Logic state at D
5 V		1 V	
4 V		0.75 V	
3.5 V		0.5 V	
2 V		$\sim 0 \text{ V}$	

- Redraw the circuit in Figure 1.2 so that outputs are as per Table 1.6 below:

TABLE 1.6

SL	SU	Lamp	Logic state of the Lamp
ON	ON	ON	1
OFF	OFF	OFF	0
ON	OFF	ON	1
OFF	ON	ON	1

- Consider the circuit in Figure 1.3. What will the states at D_0 , D_1 when the input $Y_2 = 1$?
- Using a pencil draw in a row horizontally the seven small-unfilled circles, like the one in character small *o*. Repeat this nine times vertically such that circles are in nine rows and seven columns. Now erase the circles in each row such that English character A becomes visible. Write the sequence of 0s and 1s in all the nine rows. Assume that a circle means 1 and no circle means 0. (Hint: The first row will have only one circle in the center filled for the character A. Hence the first row sequence answer will be 0001000).
- Using the circuit in Figure 1.3, fill the Table 1.7 given below: (Use negative logic).

TABLE 1.7

Y	D_1	D_0
$Y_0 = 1$		
$Y_1 = 1$		
$Y_2 = 1$		
$Y_3 = 1$		

6. Let us assume that for the logic states in a circuit,
- (i) 1 defines by an output signal voltage 5 V and
 - (ii) 0 defines by the output signal voltage 0.4 V.

Assume output logic state changes every ms in the sequence 1, 1, 0, 1, 1, 0, 0 and 0. Show the waveform within the period 0 to 8 ms.

7. Let us assume that in the logic signals from a circuit,
- (iii) 1 means the output signal is of voltage -12 V and
 - (iv) 0 means the output signal is of voltage +12 V.

Assume output logic changes every ms in the sequence 1, 1, 0, 1, 1, 0, 0 and 0. Show the waveform within the period 0 to 8 ms.

8. Let us assume that in the logic signals from a modem,
- (v) 1 means the output signal is of frequency 2 kHz and
 - (vi) 0 means the output signal is of frequency 1 kHz.

Assume output signal is a sine wave. Show the waveform within the period 0 to 8 ms when digital bits are in sequence 1, 1, 0, 1, 1, 0, 0 and 0.

9. Let us assume that in the logic signals from a modem,
- (vii) 1 means phase of the output signal is +90° and
 - (viii) 0 means phase of the output signal is -90°.

Assume that the output signal is a sine wave of frequency = 1 kHz. When the logic state is 1 between 0 to 1 ms, 2 to 3 ms and 4 to 5 ms, and logic state is 0 between 1 to 2 ms, 3 to 4 ms and 5 to 6 ms, show the output waveform between 0 to 6 ms on a graph paper.

10. Let us assume that in the 1 kHz signals from a modem,
- (i) when the logic pair of bits = 0 and 1 means the phase of the output signal is 0°,
 - (ii) when the logic pair of bits = 0 and 0 means the phase of the output signal is +90°,
 - (iii) when the logic pair of bits = 1 and 0 means the phase of the output signal is +180°, and
 - (iv) when the logic pair of bits = 1 and 1 means the phase of the output signal is -90°.

Show the waveform within the period 0 to 4 ms when the digital bits are in the sequence 1, 1, 0, 1, 1, 0, 0 and 0.

■ QUESTIONS

1. Define a digital circuit.
2. Define analog circuit.
3. What does 1 means in a logic circuit of TTL family? (Hint: refer Table 1.1).
4. What does 0 mean in a logic circuit of TTL family? (Hint: refer Table 1.1).
5. What does 1 mean in a negative logic circuit operating at 5 V?
6. What does 1 mean in a RS232C logic circuit? (Hint: refer Table 1.1).
7. What does 1 mean in a logic circuit of a modem operating at 1040 Hz and 1240 Hz? (Hint: refer Table 1.1).
8. Draw a +5 V circuit in which when the switch is pressed, output is 1 and when release a the output is 0.
9. Draw a +5 V negative logic circuit in which when the switch is pressed, output is 1 and when release the output is 0.
10. What does 0 means in a logic circuit of a modem operating at 2050 Hz and 2250 Hz? (Hint: refer Example 1.9).

CHAPTER 2

Boolean Algebra and Theorems, Minterms and Maxterms

OBJECTIVE

In this chapter, we shall learn logic operations and the concept of truth table. We shall learn Boolean algebraic laws and theorems. Topics such as sum of products (SOPs) and product of sums (POSs) and minterms and maxterms will also be studied.

Six digital-electronics logic-gates are used as basic element in a simple as well as complex circuit. These are NOT, NAND, AND, OR, NOR and XOR. An interesting combination is NOT-XOR.

2.1 THE NOT, AND, OR LOGIC OPERATIONS

Figure 2.1 gives the logic symbols as well as the truth tables of NOT, NAND, AND, OR gates.

2.1.1 The NOT Logic Operation

NOT operation [Figure 2.2(a)] means that the output is the complement of the input. If input is logic 1, an output F is logic 0 and if input A is logic 0, the output is logic 1. In other words,

$$F = \bar{A} \quad \dots(2.1)$$

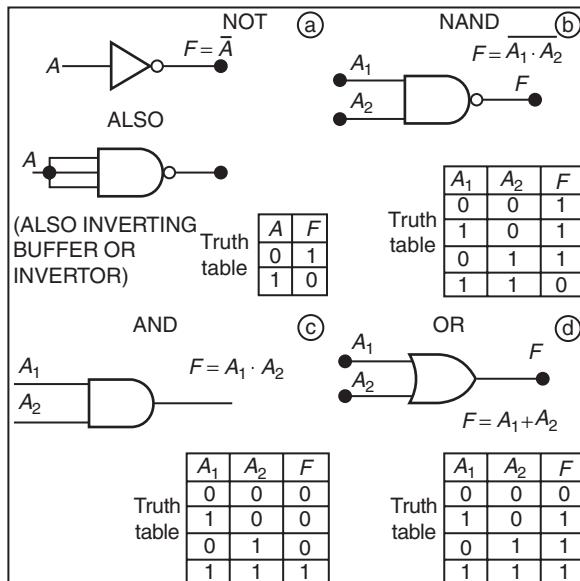


FIGURE 2.1 The logic symbols in a circuit and the truth tables of NOT, NAND, AND, OR operations.

‘ A ’ is a Boolean variable, that represents an input. It can have two values, 1 or 0. ‘ F ’ is also a Boolean variable, that represents an output. It have two states, 1 or 0. Bar over A denotes a NOT logic operation on A .

NOT in a circuit is generally represented by a triangle followed by a bubble or a bubble followed by a triangle (refer Figure 2.1(a)).

Point to Remember

NOT gate unique property is that output is 1 if input is at 0 logic state and output is 0 if input is at 1 logic state.

2.1.2 The AND Logic Operation

The unique property of AND is that its output is 0 unless all the inputs to it are at the logic 1s. It is represented by the symbol in Figure 2.1(c). Two-input (A_1 and A_2) AND-gate has the following way of writing its operations for an output F .

$$F = A_1 \cdot A_2 \quad \dots(2.2)$$

F, A_1 and A_2 are the Boolean variable representations and can either take the state = 1 or 0. Dot between two states indicates AND logic operation using these. Operation is such that when both inputs are 1s, the output is 1 else it is 0.

Three-input (A_1, A_2 and A_3) AND gate has the following way of writing its operations for the output F .

$$F = A_1 \cdot A_2 \cdot A_3 \quad \dots(2.3)$$

Operation is again such that when all three inputs are 1s, the output is 1 else it is 0.

Point to Remember

AND gate unique property is that output is 1 only if all of the inputs are at 1.

We will note later that AND symbol differs from NAND only by the omission of the bubble (circle) in that. We can note the differences in symbols and truth tables of NOT and AND in Figures 2.1(a) and (c).

Point to Remember

A truth table has 2^n rows for 2^n combination of n inputs. It gives in each of its row m outputs for a given combination. It gives the logic output(s) after the logical operations under different possible conditions of the input(s).

2.1.3 The OR Logic Operation

An OR operation means that the output is 0 only if all the inputs are 0. It is represented by the symbol in Figure 2.1(d). If any of the inputs is 1, the output F is 1. A two input OR gate has the following way of showing the operation.

$$F = A_1 + A_2 \quad \dots(2.4)$$

Three inputs OR gate has the following way of showing the operation.

$$F = A_1 + A_2 + A_3 \quad \dots(2.5)$$

Sign of + between the two logic states indicates an OR logic operation.

2.2 THE NAND AND NOR LOGIC OPERATIONS**2.2.1 NAND Gate**

NAND gate property is that output is 1 if any of the input is at 0 logic state. Let us consider two inputs with the states A_1 and A_2 at a NAND gate.

$$(\text{Output}) F = \overline{A_1 \cdot A_2} \quad \dots(2.6)$$

Bar denotes a NOT logic operation after the dot operation, $A_1 \cdot A_2$. The meaning of dot in $A_1 \cdot A_2$ is AND operation, which is explained in section 2.1.2.

Let us consider three inputs with the states A_1 , A_2 and A_3 at the NAND gate.

$$(\text{Output}) F = \overline{\overline{A_1 \cdot A_2} \cdot A_3} \quad \dots(2.7)$$

1. NAND gate with all inputs made common gives us the NOT operation. [Figure 2.1(a)]
2. NAND gate preceded by a NOT gate gives us AND gate.

Points to Remember

NAND gate unique property is that output is 1 if any of the input is at 0 logic state. NAND operation is also a NOT operation after an AND operation.

Figure 2.2 gives the logic symbols as well as truth tables of NOR, XOR NOT-XOR and NOT-NOT (Buffer) gates.

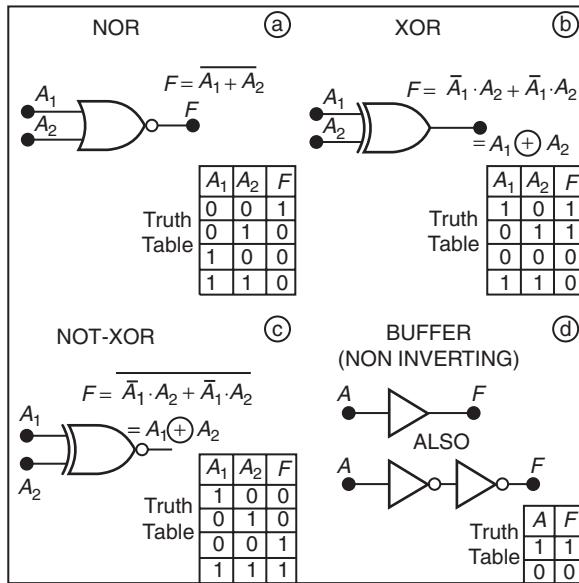


FIGURE 2.2 The logic symbols as well as truth tables of NOR, XOR NOT-XOR and Buffer gates.

2.2.2 NOR Gate

An OR circuit followed a NOT circuit gives a NOR gate and is shown in Figure 2.2(a). Its unique property is that its output is 0 if any of the inputs is 1. A two input NOR has

$$F = \overline{A_1 + A_2} \quad \dots(2.8)$$

2.3 THE XOR, NOT-XOR, NOT-NOT LOGIC OPERATIONS

2.3.1 XOR Logic Operation

An XOR gate (Figure 2.2(b)) is called Exclusive OR gate. Its unique property is that the output is 1 only if odd numbers of the inputs at it are 1s. The output—

$$F = A_1 \cdot \overline{A_2} + \overline{A_1} \cdot A_2 \text{ or } A_1 \oplus A_2 \quad \dots(2.9)$$

Exclusive OR operation symbol is \oplus (Equation (2.9)) XOR symbol in a logic circuit is as shown in Figure 2.2(b). XOR is important in the circuits for addition of two binary numbers.

2.3.2 NOT-XOR (XNOR) Logic Operation

A NOT-XOR gate (Figure 2.2 (c)) has a unique property that the output is 0 only if odd numbers of the input are 1s. It is like a NOR gate but differs in the output when even number of its inputs are 1s.

2.3.3 NOT-NOT Logic Operation

A NOT-NOT (also called a BUFFER gate) (Figure 2.2(d)) simply gives same state at output as that at the input.

Its output $F = \bar{\bar{A}}$ instead of $F = \bar{A}$ in the NOT logic operation.

Its symbol in a circuit is shown in Figure 2.2(d) along with its truth table.

2.4 BOOLEAN ALGEBRAIC RULES (FOR OUTPUTS FROM THE INPUTS)

George Boole in 1854 developed mathematics now referred as Boolean algebra. It differs from usual mathematics. It is based upon the logic. For example, $A + A = A$ because true or true remains (remember that a + stands for OR logic). Two truths cannot negate to false. In Boolean algebra, there are only two numbers 1 and 0 corresponding to true and false or high and low or OFF and ON. Inversion (complementation) of true is false, and vice versa. In digital electronic logic circuits design, OR the algebraic rules are highly useful. The following laws (rules) can be said to be associated with the Boolean algebra.

2.4.1 OR Rules

The OR laws are described by following equations:

$$A + 1 = 1 \quad \dots(2.10a)$$

$$A + 0 = A \quad \dots(2.10b)$$

$$A + A = A \quad \dots(2.10c)$$

$$A + \bar{A} = 1 \quad \dots(2.10d)$$

An OR operation is denoted by a plus sign. OR laws means (i) any number (0 or 1) is a first input to an OR gate and another number at the second input is 1 then answer is 1, (ii) if another is 0 then answer is same as first input and (iii) If two inputs to an OR gate complements then output is '1'. These can be observed in an inset of the truth table in Figure 2.1(d).

2.4.2 AND Rules

Let us recall an AND operation. It is denoted by the dot sign. True and true make true. True and false make false. False and false also remain false.

$$A \cdot 1 = A \quad \dots(2.11a)$$

$$A \cdot 0 = 0 \quad \dots(2.11b)$$

$$A \cdot A = A \quad \dots(2.11c)$$

$$A \cdot \bar{A} = 0 \quad \dots(2.11d)$$

These rules can be observed in an inset of the truth table in Figure 2.1(c).

A useful Boolean rule, which derives from equations (2.10a), (2.10b) and (2.11a), is as follows:

$$X + X \cdot Y = X \quad \dots(2.11d)$$

2.4.3 NOT Rules (Rules of Complementation)

A NOT operation is denoted by putting a bar over a number. A NOT true means false. A NOT false means true.

$$\bar{0} = 1 \quad \dots(2.12a)$$

$$\bar{1} = 0 \quad \dots(2.12b)$$

$$\bar{\bar{A}} = A \quad \dots(2.12c)$$

Equation (2.12c) means that if A is inverted (complemented) and then again inverted, we get the original number (Figure 2.2(d)). Let us refer to an inset of the truth table in Figure 2.1(a) for equations (2.12a and b).

2.5 BOOLEAN ALGEBRAIC LAWS

2.5.1 Commutative Laws

These laws mean that order of a logical operation is immaterial.

$$A_1 + A_2 = A_2 + A_1 \quad \dots(2.13a)$$

$$A_1 \cdot A_2 = A_2 \cdot A_1 \quad \dots(2.13b)$$

2.5.2 Associative Laws

These Laws allow a grouping of the Boolean variables.

$$X + (Y + Z) = (X + Y) + Z \quad \dots(2.14a)$$

$$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z \quad \dots(2.14b)$$

2.5.3 Distributive Laws

These laws simplify the problems in the logic designs.

$$X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z) \quad \dots(2.15a)$$

$$X + (Y \cdot Z) = (X + Y) \cdot (X + Z) = (X + Y) \cdot (X + Y + Z) \quad \dots(2.15b)$$

$$X + (\bar{X} \cdot Y) = X + Y \quad \dots(2.15c)$$

The last two equations are typical to Boolean algebra, and are not followed in the usual algebra.

2.6 DEMORGAN THEOREMS

First theorem shows an equivalence of a NOR gate with an AND gate having bubbled inputs (Figure 2.3(a)), and is given by the equation:

$$\overline{A_1 + A_2} = \overline{A_1} \cdot \overline{A_2} \quad \dots(2.16a)$$

Point to Remember

First DeMorgan theorem states, complement of two or more variables and then AND operation on these is equivalent to NOR operation on these variables. (NOR means complement of two or more variables OR).

Second theorem shows an equivalence of a NAND gate with an OR having bubbled inputs as shown in Figure 2.3(b) and is given by the equation:

$$\overline{A_1 \cdot A_2} = \overline{\overline{A_1}} + \overline{\overline{A_2}} \quad \dots(2.16b)$$

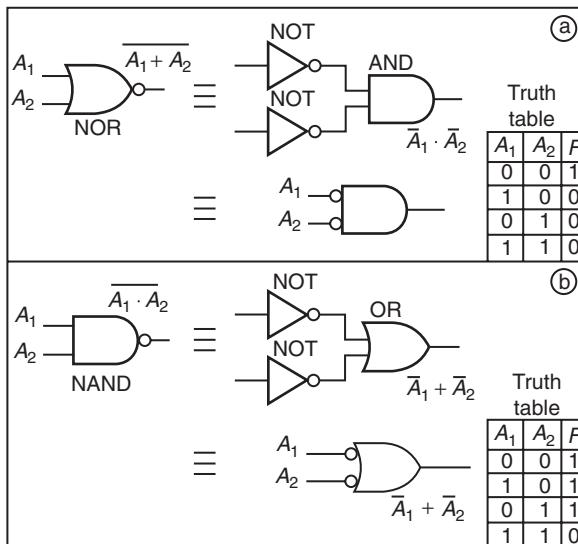


FIGURE 2.3 (a) DeMorgan first theorem showing an equivalence of NOR gate (b) DeMorgan second theorem showing an equivalence of NAND gate.

Point to Remember

Second DeMorgan theorem states that complement of two or more variables and then OR operation on these is equivalent to a NAND operation on these variables (NAND means complement of two or more variables AND).

In fact both the equations (2.16a and 2.16b) also hold for the cases of the multiple (more than two) inputs.

$$\overline{A_1 + A_2 + A_3 + \dots} = \overline{\overline{A_1} \cdot \overline{A_2} \cdot \overline{A_3} \dots} \quad \dots(2.17a)$$

$$\overline{A_1 \cdot A_2 \cdot A_3 \dots} = \overline{\overline{A_1} + \overline{A_2} + \overline{A_3} \dots} \quad \dots(2.17b)$$

These theorems find wide use in the digital logic circuit's design. A circuit is implementable by one single basic logic gate used as a basic building gate. (A house is constructed easily using the identical bricks). Similar is the criteria for choosing only the NANDs or the NORs as basic building units in the digital ICs.

Simple Formulae to Remember for the Applications

Tables 2.1 and 2.2 give methods using three steps for introducing a complement over whole term and for removal of complete bar over the whole term, respectively.

We note the following from DeMorgan theorems in Equations (2.16) and (2.17), if we use three steps:

1. complement each term on right hand side, then
2. convert the dot (AND) operation to + (OR) operation or + (OR) operation to dot (AND) operation, dot sign to + sign and vice versa and then
3. complement the remaining expression on the whole, we get the left hand side terms.

We can remove bar over the entire expression, by complementing each variable under it and then changing sign if + to a dot and if dot then to a +.

TABLE 2.1 Three step method for introducing complement over whole term in a Boolean expression

Exemplary term	Step 1: complementation	Step 2: Sign change	Step 3: Complementation over whole term
$\bar{C} + D + \bar{E}$	$C + \bar{D} + E$	$C \cdot \bar{D} \cdot E$	$\overline{C \cdot \bar{D} \cdot E}$
$(A + B) + \bar{C}$	$\overline{(A + B)} + C$	$\overline{(A + B)} \cdot C$	$\overline{\overline{(A + B)} \cdot C}$

TABLE 2.2 Removal of complete bar over the whole term in a Boolean expression

Exemplary term	Step 1: Complementation	Step 2: Sign change	Step 3: Remove
$\overline{C \cdot D \cdot E}$	$\overline{C \cdot \bar{D} \cdot \bar{E}}$	$\overline{C + \bar{D} + \bar{E}}$	$C + \bar{D} + \bar{E}$
$\overline{\bar{A} + C + D}$	$\overline{A + \bar{C} + \bar{D}}$	$\overline{A \cdot \bar{C} \cdot \bar{D}}$	$A \cdot \bar{C} \cdot \bar{D}$

2.7 THE SUM OF THE PRODUCTS (SOPs) AS PER BOOLEAN EXPRESSION AND MINTERMS

An output from the logic gates can be represented as the sum of the Minterms. Advantage of using the SOP form is that the functions of any combinatin of logic gates can be represented by the AND gates at the inputs and an OR gate at the outputs.

2.7.1 SOPs for Two Variables (Two Inputs) Case

Consider a two variable case, A and B in Table 2.3. Advantage of using the two variable SOP form is that functions of any two input logic gate functions or a truth table be represented by four ANDs at the inputs and an OR at an output.

For two input case, an output S for an XOR gate can be written by S_0 as follows: (Table 2.3 shows how to select the minterms for the XOR, AND, OR and NAND gate outputs.)

$S_0 = \bar{A} \cdot B + A \cdot \bar{B}$; ($S_0 = 1$ when $A = 0$ and $B = 1$, and $S_0 = 1$ when $A = 1$ and $B = 0$). In the table, for an AND the ($S_1 = 1$ when $A = 1$ and $B = 1$). In the table, for an OR and ($S_2 = 1$ when $A = 0$ and $B = 1$ or $A = 0$ and $B = 1$ or $A = 1$ and $B = 1$).

From fourth column in Table 2.3 for an XOR we find that only second and third minterm are contributing in giving the output $S_0 = 1$. There is simple way of writing the above SOP terms as follows:

$$S_0 = mn1 + mn2 = \sum m(1, 2)$$

Therefore, the SOP for two inputs AND is $S_1 = A \cdot B$. From fifth column in Table 2.3, we find that only fourth minterm is contributing in giving the output = 1

$$S_1 = (m(3))$$

The SOP for two inputs OR is $S_2 = \bar{A} \cdot B + A \cdot \bar{B} + A \cdot B$. From sixth column in Table 2.3, we find that second, third and fourth minterm are contributing in giving the output = 1. Therefore, the S output is as follows:

$$S_2 = \sum m(1, 2, 3)$$

The SOP for two inputs NAND is $S_3 = \bar{A} \cdot \bar{B} + \bar{A} \cdot B + A \cdot \bar{B}$. From seventh column in Table 2.3, we find that first, second and third minterm are contributing in giving the output = 1. Therefore, the S output is as follows:

$$S_1 = \sum m(0, 1, 2).$$

TABLE 2.3

A Input	B Input	Minterm and Boolean function	Minterm selected when output required			
			XOR S_0	AND S_1	OR S_2	NAND S_3
0	0	$mn0 = \bar{A} \cdot \bar{B}$	0	0	0	1
0	1	$mn1 = \bar{A} \cdot B$	1	0	1	1
1	0	$mn2 = A \cdot \bar{B}$	1	0	1	1
1	1	$mn3 = A \cdot B$	0	1	1	0

Note: Columns 1, 2, 4 are as per truth table for XOR in Figure 2.2(b). Columns 5, 6 and 7 are as per truth table of AND, OR and NAND.

2.7.2 SOPs for Three Variables (Three Inputs) Case

Consider a three-variables case, A , B and C (Table 2.4). Advantage of using SOP an form is that functions of any three input logic gate functions can be represented by maximum eight ANDs at the inputs and eight input OR at an outputs. Using column 6, the SOP for a logic circuit defined by S_1 minterms can be written as follows:

$$S_1' = \bar{A} \cdot B \cdot \bar{C} + A \cdot \bar{B} \cdot \bar{C} + A \cdot B \cdot C = \sum m(2, 4, 7) \quad (2.18)$$

The SOP output using column 7 for the S_2 minterms can be written as follows:

$$S_2' = \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot \bar{C} + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C} = \sum m(3, 4, 5, 6) \quad (2.19)$$

2.10 Digital Systems: Principles and Design

The SOP using column 8 for the $S3'$ minterms can be written as follows:

$$S3' = \bar{A} \cdot \bar{B} \cdot C + \bar{A} \cdot B \cdot \bar{C} + \bar{A} \cdot B \cdot C + ABC = \sum m(1, 2, 3, 7) \quad \dots(2.20)$$

The SOP using column 9 for the $S4'$ minterms can be written as follows:

$$S4' = A \cdot B \cdot C = \sum m(7) \quad \dots(2.21)$$

TABLE 2.4

A	B	C	Minterm	Boolean function	Minterm selected when output required			
					S1' (Eq. 2.18)	S2' (Eq. 2.19)	S3' (Eq. 2.20)	S4' (Eq. 2.21)
0	0	0	<i>mn0</i>	$\bar{A} \cdot \bar{B} \cdot \bar{C}$	0	0	0	0
0	0	1	<i>mn1</i>	$\bar{A} \cdot \bar{B} \cdot C$	0	0	1	0
0	1	0	<i>mn2</i>	$\bar{A} \cdot B \cdot \bar{C}$	1	0	1	0
0	1	1	<i>mn3</i>	$\bar{A} \cdot B \cdot C$	0	1	1	0
1	0	0	<i>mn4</i>	$A \cdot \bar{B} \cdot \bar{C}$	1	1	0	0
1	0	1	<i>mn5</i>	$A \cdot \bar{B} \cdot C$	0	1	0	0
1	1	0	<i>mn6</i>	$A \cdot B \cdot \bar{C}$	0	1	0	0
1	1	1	<i>mn7</i>	$A \cdot B \cdot C$	1	0	1	1

Note: Columns 1, 2, 3 and 6 define the truth table for $S1'$.

2.7.3 SOPs for Four Variables (Four Inputs) Case

Consider a four -variables case, A , B , C and D in Table 2.5. Advantage of using SOP form 4-inputs in logic circuit design is that functions of any four input logic gate functions can be represented by maximum sixteen ANDs at the inputs and sixteen input OR at the outputs. Using column 7, the SOP for the $S5$ minterms can be written as some of minterms $mn2$, $mn10$ and $mn15$ as follows:

$$S5 = \bar{A} \cdot \bar{B} \cdot C \cdot \bar{D} + A \cdot \bar{B} \cdot C \cdot \bar{D} + A \cdot B \cdot C \cdot D = \sum m(2, 10, 15) \quad \dots(2.22)$$

Above equation shows that the logic circuit to implement truth table in Table 2.5 can be drawn as shown in Figure 2.4 using 3 four-input NANDs and one three-input OR gate.

TABLE 2.5

A	B	C	D	Minterm	Boolean function	Minterm selected when output
0	0	0	0	<i>mn0</i>	$\bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \bar{D}$	S5 Eq. 2.22
0	0	0	1	<i>mn1</i>	$\bar{A} \cdot \bar{B} \cdot C \cdot \bar{D}$	0
0	0	1	0	<i>mn2</i>	$\bar{A} \cdot \bar{B} \cdot C \cdot \bar{D}$	1

0	0	1	1	<i>mn3</i>	$\bar{A} \cdot \bar{B} \cdot C \cdot D$	0
0	1	0	0	<i>mn4</i>	$\bar{A} \cdot B \cdot \bar{C} \cdot \bar{D}$	0
0	1	0	1	<i>mn5</i>	$\bar{A} \cdot B \cdot \bar{C} \cdot D$	0
0	1	1	0	<i>mn5</i>	$\bar{A} \cdot B \cdot C \cdot \bar{D}$	0
0	1	1	1	<i>mn7</i>	$\bar{A} \cdot B \cdot C \cdot D$	0
1	0	0	0	<i>mn8</i>	$A \cdot \bar{B} \cdot \bar{C} \cdot \bar{D}$	0
1	0	0	1	<i>mn9</i>	$A \cdot B \cdot \bar{C} \cdot \bar{D}$	0
1	0	1	0	<i>mn10</i>	$A \cdot \bar{B} \cdot C \cdot \bar{D}$	1
1	0	1	1	<i>mn11</i>	$A \cdot \bar{B} \cdot C \cdot D$	0
1	1	0	0	<i>mn12</i>	$A \cdot B \cdot \bar{C} \cdot \bar{D}$	0
1	1	0	1	<i>mn13</i>	$A \cdot B \cdot \bar{C} \cdot D$	0
1	1	1	0	<i>mn14</i>	$A \cdot B \cdot C \cdot \bar{D}$	0
1	1	1	1	<i>mn15</i>	$A \cdot B \cdot C \cdot D$	1

Note: Columns 1, 2, 3, 4, 8 define the truth table for outputs as per Boolean expression in Equation (2.22).

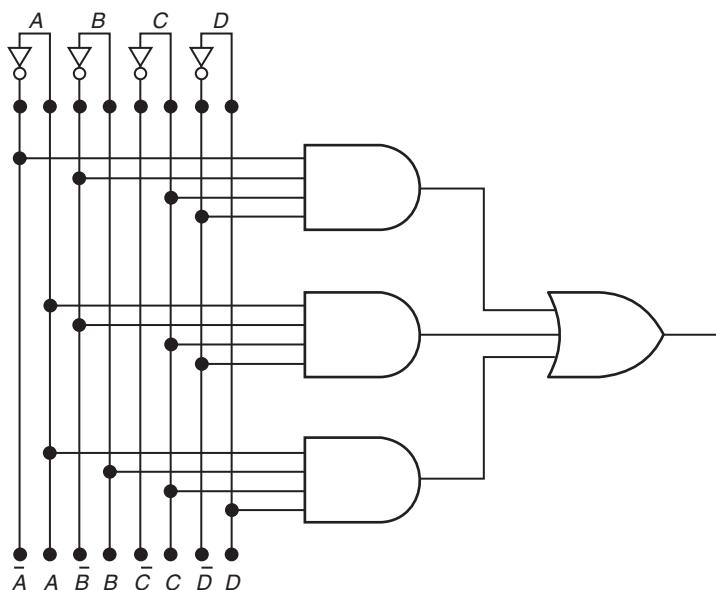


FIGURE 2.4 A logic circuit to implement Boolean expression (Eq. (2.22), truth table in Table 2.5) using the three four-input NANDs and one three-input OR gate.

Points to Remember

A simple way for laying the complement sign over a variable in m^{th} minterm of an n -variable SOP is as follows.

1. Expand m in the n -variable binary form.
2. At a place where the binary number is 0, put the complement sign and else no sign.

Let minterm is 5th and $S = \sum m(5)$ and SOP is in four variable form.

1. Expand $m: 5$ as 0101.
2. Lay the complement signs at places wherever it is 0 in $m: \bar{A} \cdot B \cdot \bar{C} \cdot D$.

Let minterm is 7th and $S = \sum m(7)$ and SOP is in four variable form.

1. Expand $m: 1$ as 0111.
2. Put complement signs at places wherever it is 0: $\bar{A} \cdot B \cdot C \cdot D$.

2.7.4 Conversion of a Boolean Expression or Truth Table Outputs into the Standard SOP Format

Suppose in a four variable SOP, there is a term with two variables only, $SOP = C \cdot D$. We perform AND operation with $(\bar{A} + A) \cdot (\bar{B} + B)$. (Using OR rule in equation (2.10d) that OR with a complement of the same variable or term with itself is always 1).

$$\begin{aligned} SOP = C \cdot D &= (\bar{A} + A) \cdot (\bar{B} + B) \cdot C \cdot D = (\bar{A} + A)(\bar{B} \cdot C \cdot D + B \cdot C \cdot D) \\ &= \bar{A} \cdot \bar{B} \cdot C \cdot D + A \cdot \bar{B} \cdot C \cdot D + A \cdot \bar{B} \cdot C \cdot D + A \cdot B \cdot C \cdot D \\ &= \sum m(3, 7, 11, 15) \end{aligned}$$

Points to Remember

When finding the minterms and converting to an n -variable SOP standard format, *in any term of the expression if a variable is not present, then do the followings:*

1. Perform additional AND operation with a term containing that variable and its complement.
2. Continue ANDing till all n variable are present in each term of the n -variable SOP.
3. Repeat the process for each term that has a missing variable in the Boolean expression.

2.8 PRODUCT OF THE SUMS AND MAXTERMS FOR A BOOLEAN EXPRESSION

An output from the logic gates can be represented as sum of the maxterms. Advantage of using POS form is that functions of any logic gates functions can be represented by OR at the inputs and an AND gate at the outputs.

2.8.1 POS for Two Variables (Two Inputs) Case

Consider a two variable case— A and B in Table 2.6. Advantage of using z -input variables POS form is that functions of any two inputs logic gates functions can be represented by four ORs at the inputs and an AND at the output.

Table 2.6 shows how to select the maxterms or XOR, AND, OR and NAND gates outputs.

TABLE 2.6

A	B	Maxterm and Boolean function	Maxterm selected when output required			
			XOR	AND	OR	NAND
			P0	P1	P2	P3
0	0	$Mx0 = A + B$	0	0	0	1
0	1	$Mx1 = A + \bar{B}$	1	0	1	1
1	0	$Mx2 = \bar{A} + B$	1	0	1	1
1	1	$Mx3 = \bar{A} + \bar{B}$	0	1	1	0

For two input case, an output P for an XOR gate can be written as follows: $\overline{P_0} = (A + B) \cdot (\bar{A} + \bar{B})$. The $\overline{P_0} = 0$ when $A = 0$ and $B = 0$ and when $A = 1$ and $B = 1$. From fourth column in Table 2.6, we find that only first and fourth maxterms are contributing in giving the output $P = 0$. Also in maxterm notations, it can be written as follows:

$$\overline{P_0} = \prod Mx(0, 3)$$

The POS for two-input AND is

$$\overline{P_1} = \prod (A + B) \cdot (A + \bar{B}) \cdot (\bar{A} + B)$$

From fifth column in Table 2.6, we find that first, second and third maxterms are contributing in giving the output = 0

$$\overline{P_1} = \prod Mx(0, 1, 2)$$

The POS for two inputs OR is

$\overline{P_2} = (A + B)$. From sixth column in Table 2.6, we find that only first maxterm is contributing in giving the output = 0.

$$\overline{P_2} = \prod Mx(0)$$

The POS for two inputs NAND is

$\overline{P_3} = (\bar{A} + \bar{B})$. From seventh column in Table 2.6, we find that only fourth maxterm is contributing in giving the output = 0.

$$\overline{P_3} = \prod Mx(3)$$

2.8.2 POS for Three Variables (Three Inputs) Case

Consider a three-variables case, A , B and C in Table 2.7. Advantage of using 3 input variables POS form is that functions of any three inputs logic gates functions can be represented by maximum eight ORs at the inputs and the AND at an output. Using column 6, the POS for the $P1'$ maxterms can be written as follows:

$$\overline{P1'} = (A + \bar{B} + C) \cdot (\bar{A} + B + C) \cdot (\bar{A} + \bar{B} + \bar{C}) = \prod M(2, 4, 7) \quad \dots(2.23)$$

The POS using column 7 for the $\overline{P2'}$ maxterms can be written as follows:

$$\overline{P2'} = (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + C) \cdot (\bar{A} + B + \bar{C}) \cdot (\bar{A} + \bar{B} + C) = \prod M(3, 4, 5, 6) \dots(2.24)$$

The POS using column 8 for the $\overline{P3'}$ maxterms can be written as follows:

$$\overline{P3'} = (A + B + \bar{C}) \cdot (A + \bar{B} + C) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + \bar{B} + \bar{C}) = \prod M(1, 2, 3, 7) \dots(2.25)$$

The POS using column 9 for the $\overline{P4'}$ maxterm can be written as follows:

$$\overline{P4'} = (\bar{A} + \bar{B} + \bar{C}) = M(7) \quad \dots(2.26)$$

TABLE 2.7

A	B	C	Maxterm	Maxterm selected when output required				
				$\overline{P1'}$	$\overline{P2'}$	$\overline{P3'}$	$\overline{P4'}$	
				Eq. 2.23	Eq. 2.24	Eq. 2.25	Eq. 2.26	
0	0	0	$Mx0$	$A + B + C$	1	1	1	1
0	0	1	$Mx1$	$A + B + \bar{C}$	1	1	0	1
0	1	0	$Mx2$	$A + \bar{B} + C$	0	1	0	1
0	1	1	$Mx3$	$A + \bar{B} + \bar{C}$	1	0	0	1
1	0	0	$Mx4$	$\bar{A} + B + C$	0	0	1	1
1	0	1	$Mx5$	$\bar{A} + B + \bar{C}$	1	0	1	1
1	1	0	$Mx6$	$\bar{A} + \bar{B} + C$	1	0	1	1
1	1	1	$Mx7$	$\bar{A} + \bar{B} + \bar{C}$	0	1	0	0

2.8.3 POS for Four Variables (Four Inputs) Case

Consider a four-variables case, A , B , C and D in Table 2.8. Advantage of using 4-input variables POS form is that functions of any four inputs logic gates functions can be represented by maximum sixteen ORs at the inputs and the AND at an output. Using column 7, the POS for the $\overline{P5}$ maxterms can be written as follows:

$$\overline{P5} = (A + B + \bar{C} + D) \cdot (\bar{A} + B + \bar{C} + D) \cdot (\bar{A} + \bar{B} + \bar{C} + D) = \prod M(2, 10, 15) \dots(2.27)$$

TABLE 2.8

A	B	C	D	Maxterm	Boolean function	Maxterm selected when output required
$\bar{P}5$ (Eq. 2.27)						
0	0	0	0	Mx0	$A+B+C+D$	1
0	0	0	1	Mx1	$A+B+C+\bar{D}$	1
0	0	1	0	Mx2	$A+B+\bar{C}+D$	0
0	0	1	1	Mx3	$A+B+\bar{C}+\bar{D}$	1
0	1	0	0	Mx4	$A+\bar{B}+C+D$	1
0	1	0	1	Mx5	$A+\bar{B}+C+\bar{D}$	1
0	1	1	0	Mx6	$A+\bar{B}+\bar{C}+D$	1
0	1	1	1	Mx7	$A+\bar{B}+\bar{C}+\bar{D}$	1
1	0	0	0	Mx8	$\bar{A}+B+C+D$	1
1	0	0	1	Mx9	$\bar{A}+B+C+\bar{D}$	1
1	0	1	0	Mx10	$\bar{A}+B+\bar{C}+D$	0
1	0	1	1	Mx11	$\bar{A}+B+\bar{C}+\bar{D}$	1
1	1	0	0	Mx12	$\bar{A}+\bar{B}+C+D$	1
1	1	0	1	Mx13	$\bar{A}+\bar{B}+C+\bar{D}$	1
1	1	1	0	Mx14	$\bar{A}+\bar{B}+\bar{C}+D$	1
1	1	1	1	Mx15	$\bar{A}+\bar{B}+\bar{C}+\bar{D}$	0

Points to Remember

A simple way for laying the complement sign over a variable in an M^{th} maxterm of an n -variable POS is to as follows.

1. Expand M in n -variable binary form.
2. At a ever place where the binary number is 1 put the complement sign else no sign such.

Let maxterm is 11th and $\bar{P} = \sum M(11)$ and POS is in four variable form.

1. Expand $M 11$: 1011.

2. Lay complement signs at places where ever it is 1 in M : POS = $\bar{A} + B + \bar{C} + \bar{D}$

Let maxterm is 6th and $\bar{P} = \sum M(6)$ and POS is in four variable form.

1. Expand $M 6$: 0110.

2. Lay the complement signs at places where ever it is 1: $A + \bar{B} + \bar{C} + D$.

2.8.4 Conversion of a Boolean Expression into Standard POS Format

When finding the maxterms and converting to an n -variable POS standard format, in any maxterm of the expression if a variable is not present, then do the followings:

1. Perform additional OR operation in the maxterm with a term containing that variable ANDed with complement of that and get two POS maxterms using a distributive law $X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$ with Y and Z as variable and its complement, respectively.
2. Continue ORing till all n variables are present in each term of the n -variable POS.
3. Repeat the process for each term that has a missing variable in the Boolean expression.

The above steps are clarified below:

Assume that a standard POS in four-variable format is to be found for the following:

$$\bar{P} = (A + C + D) \cdot (A + \bar{B} + C + D) \text{ (Missing } B \text{ in first maxterm)}$$

First maxterm = $(A + \bar{B} \cdot B + C + D)$. Use ANDing rule that AND with the complement is 0, and ORing with 0 has no effect on an expression. Refer equations (2.11d) and (2.10b).

First term = $(A + \bar{B} + C + D)(A + B + C + D)$ (Using distributive law $X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$ in equation (2.15b) by taking $X = A + C + D$, $Y = \bar{B}$ and $Z = B$).

Two terms in the last expression are in standard POS form of the first maxterm with variable B not present. Therefore,

$$\begin{aligned}\bar{P} &= M(u) \cdot M(0) \cdot M(u) \\ &= \prod M(0, 4) \text{ using Equation (2.11c)}\end{aligned}$$

■ EXAMPLES

Example 2.1

Use DeMorgan theorem to simplify $F = \overline{A + B} + \overline{C \cdot \bar{D} \cdot E}$. (Note: This is an example to remove the bar over the individual terms).

Solution

Use a remove whole bar formula for both terms as follows:

Step 1: Complement each variable and change sign: R.H.S. First term = $\bar{A} \cdot \bar{B}$

Second term = $\bar{C} + D + \bar{E}$.

R.H.S = $\bar{A} \cdot \bar{B} + \bar{C} + D + \bar{E}$

Example 2.2

Use DeMorgan theorem to simplify $F = \bar{A} \cdot \bar{B} + \bar{C} \cdot \bar{D}$. (Note: This is an example to place the bar over the terms).

Solution

Use a three step formula for both terms as follows:

Steps 1, 2 and 3: Complement, change sign and put bar over whole:

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

For further simplification, use the three step formula once again as follows:

Steps 1, 2 and 3: $\overline{(A+B) \cdot (C+D)}$

Example 2.3

Prove that $F = \bar{A} \cdot B + A \cdot \bar{B}$ is exclusive OR operation and it equals

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

Solution

There is a dot (AND) operation between \bar{A} and B . It means that A has to be 0 when $B = 1$ to get logic state 1 from the first term. There is a dot (AND) operation between A and \bar{B} . It means that B has to be 0 when $A = 1$ to get logic-state 1 from the second term. XOR gate gives answer 1 only when two variable A and B are distinct from each other. Hence, it is proved that the $\bar{A} \cdot B + A \cdot \bar{B}$ is an XOR operation.

Now we prove the second part.

Use a *remove whole bar* formula for topmost bar as follows:

Step A: Complement each variable or term and convert sign:

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

Use a remove whole bar formula for the bar as follows:

Step B: Complement each variable or term and convert sign:

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

Use distributive law [Eq. (2.15a)] to separately AND the A in first term and B in the second term as follows:

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

Use ANDing rule described by Equation (2.11d) that an AND between a variable and its complement is always 0. Hence, first and fourth terms are 0s. The ORs with 0 has no effect [Eq. (2.10b)]. Therefore, first and last terms can be removed

We get the second and third terms as follows:

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

Note: Commutative law [Eq. (2.13b)] gives $\bar{B} \cdot A = A \cdot \bar{B}$.

Hence the L.H.S of expression for F is same as R.H.S. for F after simplification.

Example 2.4

Prove that for constructing XOR from NANDs we need four NAND gates.

Solution

We first prove the expression in Example 2.3. We have four bars over the ANDs. [($A \cdot B$) is counted once only.] Therefore, we need four NANDs for constructing an XOR gate.

Example 2.5 Prove $X + (\overline{X} \cdot Y) = X + Y$ a distributive law in Equation (2.15c).

Solution

Let $F = \overline{X} \cdot (\overline{X} \cdot Y)$ (Using a three step formula for DeMorgan theorem application).

Solve for \overline{F} first as follows:

$\overline{F} = \overline{X} \cdot (X + \overline{Y})$ (Using the remove whole bar formula on the second term within the bracket).

$= \overline{X} \cdot X + \overline{X} \cdot \overline{Y}$ (Use distributive law in Equation (2.15a)).

$= \overline{X} \cdot \overline{Y}$ (Use ANDing rule described by Equation (2.11d) that AND between a variable and its complement is always 0 and OR rule that OR of A with 0 gives A (Equation 2.10b)).

$\overline{F} = \overline{X + Y}$ (Using remove whole bar formula on the second term within the bracket).

$F = X + Y$ (Taking complement of L.H.S. and R.H.S. both)

Example 2.6 Prove $X + (Y \cdot Z) = (X + Y) \cdot (X + Z) = (X + Y) \cdot (X + Y + Z)$ a distributive law mentioned in Equation (2.15b). Use DeMorgan theorems.

Solution

It means

R.H.S. = $\overline{(X + Y)} + \overline{(X + Z)}$ (Using a three step formula for DeMorgan theorem application).

$= \overline{(X + Y)} + \overline{X} \cdot \overline{Z}$ (Using remove whole bar formula on the second term within the bracket).

$= (\overline{X} \cdot \overline{Y}) + (\overline{X} \cdot \overline{Z})$ (Using remove whole bar formula on the first term).

$= (\overline{Y} + \overline{Z}) \cdot \overline{X}$ (Using the distributive law in Equation (2.15a)). ... (2.28a)

L.H.S. = $X + Y \cdot Z$

$= X + \overline{Z} + \overline{Y}$ (Using deMorgan Theorem)

$= \overline{X} \cdot (\overline{Z} + \overline{Y})$ (Using deMorgan Theorem)

... (2.28b)

L.H.S. = R.H.S. because Equation (2.28a) equals Equation (2.28b)

We proved that $X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$.

It also means it is proved $(X + Y) + (Y \cdot Z) = (X + Y + Y) \cdot (X + Y + Z) = (X + Y) \cdot (X + Y + Z)$ (Replace X by $X + Y$ on both sides. Because if a law is true for X then it should also be true for $X + Y$).

Example 2.7 Prove $X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$ a distributive law [Equation (2.15b)] by using AND and OR Boolean rules and Boolean laws.

Solution

Alternative solution of the problem in Example 2.6 is as follows:

$$(X + Y) \cdot (X + Z)$$

$$\begin{aligned}
 &= X \cdot X + X \cdot Z + Y \cdot X + Y \cdot Z \text{ (Use distributive law in Equation 2.15a).} \\
 &= X + X \cdot Z + Y \cdot X + Y \cdot Z \text{ (Using ANDing rule Equation (2.11c)).} \\
 &= X(1+Z) + Y \cdot X + Y \cdot Z \text{ (Using the distributive law and taking } X=X.1 \text{ due to} \\
 &\text{the rule that AND with 1 returns the same term. [Equation (2.11a)]}) \\
 &= X + Y \cdot X + Y \cdot Z \text{ (Using OR rule in equation (2.10a). OR with 1 give 1).} \\
 &= X(1+Y) + Y \cdot Z \text{ (Using as above the equation (2.11a)).} \\
 &= X + Y \cdot Z \text{ (Using as above the equation (2.10a)).}
 \end{aligned}$$

Example 2.8 Simplify $A \cdot C + A \cdot (C + B) + C \cdot (C + B)$ using Boolean rules. and draw the simplest possible logic circuit.

Solution

$$\begin{aligned}
 \text{Expression} &= A \cdot C + A \cdot (C + B) + C \cdot (C + B) \\
 &= A \cdot C + A \cdot C + A \cdot B + C \cdot C + C \cdot B \text{ (Using distributive law in} \\
 &\text{equation 2.15a))} \\
 &= A \cdot C + A \cdot B + C \cdot C + C \cdot B \text{ (Using OR rule that OR with itself} \\
 &\text{gives the same term; equation (2.10c). So } A \cdot C + A \cdot C = A \cdot C) \\
 &= A \cdot C + A \cdot B + C + C \cdot B. \text{ (Using rule that AND operation by itself} \\
 &\text{gives the same term. Equation (2.11c))} \\
 &= A \cdot C + A \cdot B + (1+B) \cdot C \text{ (Using disvtributive law and taking} \\
 &\text{C} = C \cdot 1 \text{due to the rule that AND with 1 returns the same term.} \\
 &\text{Equation (2.11a))} \\
 &= A \cdot C + A \cdot B + C \text{ (Using OR rule in equation (2.10a). OR with 1 give 1)} \\
 &= A \cdot B + C \cdot (A+1). \text{(Using distributive law between first and third term)} \\
 &= AB + C \text{ (Using OR rule in Equation (2.10a). OR with 1 give 1)}
 \end{aligned}$$

Figure 2.5 shows the simplest logic circuit for $A \cdot C + A \cdot (C + B) + C(C + B)$.

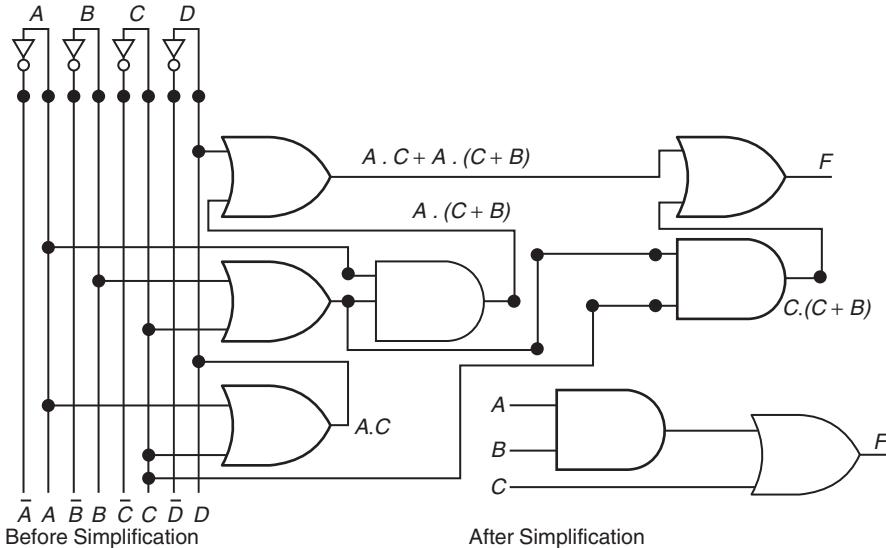


FIGURE 2.5 $A \cdot C + A \cdot (C + B) + C(C + B)$ after its simplification as $A \cdot B + C$.

Example 2.9

Find minterms and give SOP form of a combinational logic circuit that has the truth table as in Table 2.9.

TABLE 2.9

Term number	A	B	C	D	Output S1
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	1

Solution

We note that output is 1 in line number 0, 4, 8 and 15. There are four minterms present

$$\begin{aligned} \text{Therefore } \text{SOP} &= \sum m (0, 4, 8, 15) \\ &= \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + A \cdot B \cdot C \cdot D \end{aligned}$$

We use a simple way for laying the complement signs at the 0th, 4th, 8th and 15th terms. In an m^{th} minterm in an n -variable SOP it is as follows.

1. Expand $m = 0, 4, 8$ and 15 in n variable binary form. 0000, 0100, 1000, 1111.
2. Where ever the binary number is 0 put the complement sign over the variable else leave as such.

Example 2.10

Find maxterms and give POS form of a combinational logic circuit that has the truth table as in Table 2.10.

TABLE 2.10

Term number	A	B	C	D	Output P1
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

Solution

We note that output is 0 in line number 0, 4, and 8. There are thus three maxterms present

$$\text{Therefore } \text{POS} = \Sigma M(0, 4, 8) \\ = (A + B + C + D) \cdot (A + \bar{B} + C + D) \cdot (\bar{A} + B + C + D)$$

We use a simple way for putting complement signs at the 0th, 4th, and 8th. In an M^{th} maxterm in a four variable POS is to as follows.

1. Expand $M = 0, 4$ and 8 in four variable binary form $0000, 0100, 1000$
2. When binary number is 1 lay the complement sign over the variable else leave as such.

Example 2.11 Convert $A \cdot B \cdot C + A \cdot D$ expression into standard SOP format.

Solution

Since there are four variables A, B, C and D in the expression, we have to obtain four variable SOP for $ABC + AD$

First term = $A \cdot B \cdot C \cdot (\bar{D} + D)$ Using rule that ORing a variable with its complement is always 1. [Equation (2.10d)]

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

$$\begin{aligned} \text{Second Term} &= A \cdot D \\ &= A \cdot (B + \bar{B}) \cdot D \end{aligned}$$

First for missing variable B , we use a rule that ORing a variable with its complement is always 1. [Equation (2.10d)]

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

Now for missing variable C in both terms, we use a rule that ORing a variable with its complement is always 1. [Equation (2.10d)]

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

$A \cdot B \cdot C + A \cdot D =$ First term SOP standard form + Second term SOP standard form

$$\begin{aligned} &= A \cdot B \cdot C \cdot \bar{D} + A \cdot B \cdot C \cdot D + A \cdot \bar{B} \cdot \bar{C} \cdot D + A \cdot \bar{B} \cdot C \cdot D \\ &\quad + A \cdot B \cdot C \cdot D + A \cdot B \cdot \bar{C} \cdot D. \\ &= m(14) + m(15) + m(9) + m(11) + m(15) + m(13) \\ &= \sum m(9, 11, 13, 14, 15) \end{aligned}$$

Example 2.12 Convert $(A + B + C) \cdot (A + D)$ expression into standard POS format.

Solution

Since there are four-variables A, B, C and D in the expression, we have to obtain four variable POS.

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

$$\text{First term} = (A + B + C + \bar{D}) \cdot D$$

Using rule that ANDing a variable with its complement is always 0; [Equation (2.11d)] Further ORing with 0 has no effect.

$$= (A + B + C + \bar{D}) \cdot (A + B + C + D) \quad [\text{Using distributive law in equation (2.15b) that } X + Y \cdot Z = (X + Y) \cdot (X + Z).]$$

$$\text{Second Term} = A \cdot D$$

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

Using rule that ANDing variable B with its complement is always 0; equation (2.11d). Further ORing with 0 has no effect.

$$\begin{aligned} &= (A + \bar{B} + D) \cdot (A + B + D) \quad [\text{Using distributive law in equation (2.15b) that } X + Y \cdot Z = (X + Y)(X + Z).] \\ &= (A + \bar{B} + \bar{C} \cdot C + D) \cdot (A + B + \bar{C} \cdot C + D) \end{aligned}$$

Using rule that ANDing variable C with its complement is always 0; [Equation (2.11d).] Further ORing with 0 has no effect.

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

[Using distributive law in equation (2.15b) that $X + Y \cdot Z = (X + Y)(X + Z)$.]

$$(A + B + C) \cdot (A + D) = (\text{First term POS standard form}) \cdot (\text{Second term POS standard form})$$

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

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

[Simplifying using rule that ANDing with itself returns the same variable or term; equation (2.11c).]

$= \prod M(0, 1, 2, 4, 6)$ using Table 2.8.

■ EXERCISES

1. Use DeMorgan theorem to simplify $F = \overline{A + A \cdot B} + \overline{C \cdot D \cdot E}$.
(Hint: Use a remove whole bar formula for both terms).
2. Use DeMorgan theorem to simplify $F = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{D}$.
(Hint: Use three step formula for both terms).
3. Prove that $\overline{F} = \overline{A} \cdot B + A \cdot \overline{B} + A \cdot B$ is NAND operation using DeMorgan theorems.
4. Construct XOR from NORs
5. Prove that $(A + B) \cdot (C + D) = \overline{A} \cdot \overline{B} + (\overline{C} + \overline{D})$.
6. Solve $\overline{A} \cdot \overline{B} + \overline{C} \cdot \overline{D} + E$ applying DeMorgan theorems to each term.
7. Prove $A + (B \cdot C \cdot D) = (A + B) \cdot (A + C) \cdot (A + D)$. Use DeMorgan theorems.
8. Prove $A + (B \cdot C \cdot D) = (A + B) \cdot (A + C) \cdot (A + D)$ by using AND and OR Boolean rules and laws.
9. Simplify $A \cdot B \cdot C + A(C + B) + C(C + B)$ using Boolean rules and draw the simplest possible logic circuit.
10. Find minterms and give SOP forms for S_1 , S_2 and S_3 of three combinational logic circuits that has the truth table outputs as in Table 2.11.

TABLE 2.11

Term number	A	B	C	D	Required output		
					S1	S2	S3
0	0	0	0	0	0	1	0
1	0	0	0	1	1	0	0
2	0	0	1	0	0	0	0
3	0	0	1	1	0	0	1
4	0	1	0	0	0	1	0
5	0	1	0	1	1	0	0
6	0	1	1	0	0	0	0
7	0	1	1	1	0	0	1
8	1	0	0	0	0	0	0
9	1	0	0	1	1	0	0
10	1	0	1	0	0	0	0

Table 2.11 Contd.

2.24 Digital Systems: Principles and Design

11	1	0	1	1	0	0	1
12	1	1	0	0	0	1	0
13	1	1	0	1	1	0	0
14	1	1	1	0	0	0	0
15	1	1	1	1	0	0	1

11. Find maxterms and give three POS form P_1 , P_2 and P_3 of three combinational logic circuits that has the truth table as in Table 2.12.

TABLE 2.12

Term number	A	B	C	D	Required output		
					$\overline{P_1}$	$\overline{P_2}$	$\overline{P_3}$
0	0	0	0	0	1	1	0
1	0	0	0	1	1	0	1
2	0	0	1	0	0	1	1
3	0	0	1	1	1	1	1
4	0	1	0	0	1	1	1
5	0	1	0	1	1	1	0
6	0	1	1	0	1	0	1
7	0	1	1	1	0	1	1
8	1	0	0	0	1	1	1
9	1	0	0	1	1	1	1
10	1	0	1	0	1	1	0
11	1	0	1	1	1	0	1
12	1	1	0	0	0	1	1
13	1	1	0	1	1	1	0
14	1	1	1	0	1	0	1
15	1	1	1	1	0	1	1

12. Convert SOP = AB expression into four variable standard SOP format.
 13. Convert $(\overline{A} + \overline{B} + C)$. expression into standard four variables POS format four variables.
 14. Convert SOP = $\overline{AB} + \overline{BC} + \overline{AC}$ expression into three variable standard SOP format after simplifying using DeMorgan Theorem.
 15. Convert POS $(A + B + C \cdot D)$ expression into standard four variables POS format four variables.
 16. Construct SOP expression and POS expression for a four input NAND.

■ QUESTIONS

1. Give the OR and NOT rules in Boolean algebra?
2. Write the AND rules in Boolean algebra?
3. Explain associative, commutative and distributive laws with two examples each for their uses.
4. What is the advantage of simplifying using Boolean rules and laws?
5. Why do we use NANDs in fabricating a complex TTL based logic circuits?
6. Why do we use NORs in fabricating a complex CMOS based logic circuits?
7. How do we make a maximum four input circuit using ANDs and ORs using SOPs?
8. How do we make a maximum four input circuit using ORs and ANDs using POs?
9. How do we convert POS form to SOP form? Explain by an exemplary four-variable case.
10. How do we convert SOP form to POS form? Explain by an exemplary four-variable case.

This page is intentionally left blank.

CHAPTER 3

Karnaugh Map and Minimization Procedures

OBJECTIVE

In Chapter 2, we studied Boolean algebraic rules, laws and theorems and their uses in simplifying a logic circuit. We also discussed the concept of SOPs and POSs to write Boolean expression in terms of a standard format for implementation by AND gates and OR gates. In this chapter, we shall learn how to develop a Karnaugh map, tabulate the minterms, and minimize a circuit using the map. Computer-aided minimization procedure will also be studied.

3.1 THE THREE-VARIABLE KARNAUGH MAP AND TABLES

3.1.1 Karnaugh Map from the Truth Table

Recall the truth tables at the insets in Figures 2.1(a) to (d) and 2.2(a) to (d) and in Tables 2.3 to 2.12. We note that truth table has all possible inputs at the columns on the left side and the required or observed output(s) at the right hand side.

Table 3.1 shows an unfilled Karnaugh map. A three variable Karnaugh map is a two-dimensional map built from a truth table with cells arranged as per Table 3.1. Since number of rows in a three variable (three inputs) truth table are 8, the map has 8 cells; two cells horizontal and four cells vertical. It can also be vice versa.

Points to Remember

- Row 1st has cells for AB as 00. It corresponds $\bar{A}.\bar{B}$.
 Row 2nd has cells for AB as 01. It corresponds $\bar{A}.B$
 Row 3rd has cells for AB as 11 (Complement of row 1st). It corresponds $A.B$.
 Row 4th has cells for AB as 10. (Complement of row 2nd). It corresponds $A.\bar{B}$.
 Column 1st has cells for $C = 0$.
 Column 2nd has cells for $C = 1$.

TABLE 3.1 Three variable Karnaugh Map

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

Let in a three variable truth table, when $A = 0, B = 1, C = 0$, the output = 1. We place 1 in the 2nd row and column 1st. This is because 2nd row is for AB as 01 and column 1st is for $C = 0$.

When $A = 1, B = 1, C = 1$, we place 1 in 3rd row column 2nd. This is because 3rd row is for AB 11 and column 2nd is for $C = 1$.

Table 3.2 is a filled map with the above states of A, B and C at the truth table.

TABLE 3.2 Three variable Karnaugh Map for $S = 1$ an output only when $A=0, B=1, C=0$, and $A=1, B=1, C=1$ at a truth table of a given logic circuit

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

Remember the Following Operations

1. When output is 1 for a given combination of A, B and C , we place 1 at the corresponding cell.
2. Complete the step 1 for all the rows of truth table with output states = 1.

3.1.2 Karnaugh Map from the Minterms in a SOP

Recall Sections 2.7.1 to 2.7.3. Recall the Tables 2.3 to 2.5. A truth table for the output from a logic circuit can also be expressed as SOP minterms. Therefore, Karnaugh map can also be made from the minterms corresponding to an output.

Table 3.1 showed an unfilled Karnaugh map. A three variable Karnaugh map can be a two-dimensional map built from the minterms for a truth table. The cells are arranged as per Table 3.3. It shows the minterm numbers corresponding to the cells. Since maximum number of minterms in a three variables (three inputs) SOP can be 8, the map has 8 cells; two cells horizontal and four cells vertical (it can also be vice versa).

Points to Remember

- Row 1st has cells for AB as 00. It corresponds $\bar{A}.\bar{B}$ part containing the minterms $m(0)$ and $m(1)$.
- Row 2nd has cells for AB as 01. It corresponds $A.B$ containing minterms $m(2)$ and $m(3)$.
- Row 3rd has cells for AB as 11 (complement of row 1st). It corresponds $A.B$ containing minterms $m(6)$ and $m(7)$.
- Row 4th has cells for AB as 10 (complement of row 2nd). It corresponds $A.B$ containing minterms $m(4)$ and $m(5)$.
- Column 1st has cells for $C = 0$.
- Column 2nd has cells for $C = 1$.

TABLE 3.3 Three variable Karnaugh Map

$AB \backslash C$	C	\bar{C}	C
AB		0	1
$\bar{A} \bar{B}$	00	$m(0)$	$m(1)$
$\bar{A} B$	01	$m(2)$	$m(3)$
$A \bar{B}$	11	$m(6)$	$m(7)$
$A B$	10	$m(4)$	$m(5)$

Let in a three variable minterm, $\bar{A}\bar{B}C$ is present and $A.\bar{B}.\bar{C}$ is present in a particular three input logic circuit output under different input states. It means $S = \Sigma m(3, 4)$. It means terms $m(3)$ and $m(4)$ are present. Therefore we place 1 in 2nd row column 2nd and place 1 in row 4th column 1.

Table 3.4 is a filled map with the above SOP output and above set of minterms.

Remember the Following Operation

1. When minterm is present in a SOP expression, we place 1 at the corresponding cell for the term using Table 3.3.
2. Complete the step 1 for all the rows for all minterms present in SOP of a Boolean expression for the logic circuit.

3.4 Digital Systems: Principles and Design

TABLE 3.4 Three variable Karnaugh Map for $S = 1$ an output when $SOP = \sum m(3, 4) = \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot \bar{C}$

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

3.1.3 Karnaugh Map from the Maxterms IN A POS

Recall Sections 2.7.4 to 2.7.6. Recall the Tables 2.6 to 2.8. A truth table for the output from a logic circuit can also be expressed as POS maxterms. Therefore, Karnaugh map can also be made from the maxterms corresponding to an output.

Table 3.5 shows an unfilled Karnaugh map. A three variable Karnaugh map can be a two-dimensional map built from the maxterms for a truth table. The cells are arranged as per Table 3.5. It shows the maxterm numbers corresponding to the cells. Since maximum number of minterms in a three variables (three inputs) POS can be 8, the map has 8 cells; two cells horizontal and four cells vertical (It can also be vice versa).

Points to Remember

Row 1st has cells for AB as 00. It corresponds $(A + B)$ part containing the maxterm.

Row 2nd has cells for AB as 01. It corresponds $(A + \bar{B})$ containing minterm.

Row 3rd has cells for AB as 11. (complement of row 1st). It corresponds $(\bar{A} + \bar{B})$ containing minterm.

Row 4th has cells for AB as 10. (complement of row 2nd). It corresponds $(\bar{A} + B)$ containing minterm.

Column 1st has cells for $C = 0$.

Column 2nd has cells for $C = 1$.

TABLE 3.5 Three variable Karnaugh Map

$AB \backslash C$	C	C	\bar{C}
AB	0	0	1
$A + B$	00	$M(0)$	$M(1)$
$A + \bar{B}$	01	$M(2)$	$M(3)$
$\bar{A} + \bar{B}$	11	$M(6)$	$M(7)$
$\bar{A} + B$	10	$M(4)$	$M(5)$

Note: For map of maxterms, we just replace the dot (AND) sign operation with + (OR) sign and replace each variable A, B and C by the respective complements.

Let in a three variable map the maxterm, $(A + \bar{B} + \bar{C})$ is present and the $(\bar{A} + B + C)$ is present in a particular three input logic circuit output under different input states. It means $P = \Pi M(3, 5)$. [$M(3)$ and $M(5)$ are present.] Therefore, we place 0 in 2nd row column 2nd and place 0 in row 4th column 2.

Table 3.6 is a filled map with the above POS output and above set of Maxterms in the cells.

TABLE 3.6 Three variable Karnaugh Map for \bar{P} = an output when $POS = \Pi M(3, 5) = (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + C)$

$AB \backslash C$	C	C	\bar{C}
AB	0	1	
$A + B$	00		
$A + \bar{B}$	01		0
$\bar{A} + B$	11		
$\bar{A} + B$	10		0

Note: For map pf maxterms, we just fill 0s at the map using cells for maxterms defined in Table 3.3.

Remember the Following Operations

- When maxterm is present in a POS expression, we place 0 at the corresponding cell for the term.
- Complete the step 1 for all the rows for the maxterms of a Boolean expression or logic circuit corresponding to that.
- For map of the POS maxterms from minterms K-map, we just consider a dot (AND) sign operation replaced with the + (OR) sign and each variable A, B and C just replaced by their respective complements. Also instead of placing 1s, we place 0s in POS Karnaugh maps (Compare maps Tables 3.5 with 3.3 and expands in Tables 3.6 and 3.4.).

3.2 FOUR VARIABLE KARNAUGH MAP AND TABLES

3.2.1 Karnaugh Map from the Truth Table

From Sections 3.1.1 to 3.1.3, we have learnt that for a three variable (input) case, a Karnaugh map is another way of representation of the truth table and the circuit logic outputs under various logic states of the inputs. Now consider the *four variables* (inputs) case.

Table 3.7 shows an unfilled four-variable Karnaugh map. A four variable Karnaugh map is a two-dimensional map built from a truth table with cells arranged as per Table 3.7. Since number of rows in a four variables (four inputs) truth table are 16, the map has 16 cells; four cells horizontal and four cells vertical.

Points to Remember

Row 1st has cells for AB as 00. It corresponds to $\bar{A}\bar{B}$.
 Row 2nd has cells for AB as 01. It corresponds to $\bar{A}B$.
 Row 3rd has cells for AB as 11. (Complement of row 1st). It corresponds to $A\bar{B}$.
 Row 4th has cells for AB as 10. (Complement of row 2nd). It corresponds to $A\bar{B}$.
 Column 1st has cells for CD as 00. It corresponds to $\bar{C}\bar{D}$.
 Column 2nd has cells for CD as 01. It corresponds to $\bar{C}D$.
 Column 3rd has cells for CD as 11. (Complement of column 1st). It corresponds to $C\bar{D}$.
 Column 4th has cells for CD as 10. (Complement of column 2nd). It corresponds to $C\bar{D}$.

Table 3.7 Four variable Karnaugh Map

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

Let in a four variable truth table there are four rows in which output = 1,

- When $A = 0, B = 0, C = 0, D = 1$, the output = 1. We place 1 in the 1st row and column 2nd. This is because 1st row is for AB as 00 and column 2nd is for $CD = 01$.
- When $A = 0, B = 0, C = 1, D = 1$, the output = 1. We place 1 in the 1st row and column 3rd. This is because 1st row is for AB as 00 and column 3rd is for $CD = 11$.
- When $A = 0, B = 1, C = 0, D = 1$, the output = 1. We place 1 in the 2nd row and column 2nd. This is because 2nd row is for AB as 01 and column 2nd is for $CD = 01$.
- When $A = 0, B = 1, C = 1, D = 1$, the output = 1. We place 1 in the 2nd row and column 3rd. This is because 2nd row is for AB as 01 and column 3rd is for $CD = 11$.

Table 3.8 is a filled map with the above states at a truth table.

Points to Remember

- When output is 1 for a given combination of A, B, C and D , we place 1 at the corresponding cell.
- Complete the step 1 for all the rows of truth table with outputs = 1.

TABLE 3.8 Four variable Karnaugh map for output $S = A$ logic circuit in which the output = 1 when A, B, C , and $D = 0001, 0011, 0101$ and 0111 , respectively

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

3.2.2 Karnaugh Map from the Minterms in an SOP

Four variables (input) Karnaugh map can also be made from the minterms corresponding to an output expression.

Table 3.9 shows an unfilled Karnaugh map. A four variable Karnaugh map can be a two-dimensional map built from the minterms for a truth table. The cells are arranged as per Table 3.9. It shows the minterm numbers corresponding to the cells. Since maximum number of minterms in a four variables (four inputs) SOP can be 16, the map has 16 cells, four cells horizontal and four cells vertical.

Remember

Row 1st has cells for AB as 00. It corresponds $\bar{A}\bar{B}$ part containing the minterm $m(0)$ to $m(2)$.

Row 2nd has cells for AB as 01. It corresponds $\bar{A}B$ containing minterms $m(4)$ to $m(7)$.

Row 3rd has cells for AB as 11 (Complement of row 1st). It corresponds $A.B$ containing minterms $m(12)$ to $m(15)$.

Row 4th has cells for AB as 10. (Complement of row 2nd). It corresponds $A\bar{B}$ containing minterms $m(8)$ to $m(11)$.

Column 1st has cells for CD as 00. It corresponds $C.D$ part containing the minterms.

Column 2nd has cells for CD as 01. It corresponds to $C.D$ containing minterms. Column 3rd has cells for CD as 11. (Complement of column 1st). It corresponds to $C.D$ containing minterms.

Column 4th has cells for CD as 10. (Complement of column 2nd). It corresponds to $C.\bar{D}$ containing minterms.

Let in a four variable the minterms— $\bar{A}\bar{B}\bar{C}\bar{D}$ is present, $\bar{A}\bar{B}C.D$ is present, $\bar{A}B.C.D$ is present, $A.B.C.D$ is present, $A.B.C.\bar{D}$ is present and $A.B.C.D$ is present in a particular four inputs logic circuit output under different input states. It means

3.8 Digital Systems: Principles and Design

TABLE 3.9 Four variables Karnaugh Map with minterms of the SOP expression

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

$S = \sum m (2, 3, 6, 7, 14, 15)$. It means 6 terms 2, 3, 6, 7, 14 and 15 are present. There we place 1s as shown in six cells of Table 3.10.

Table 3.10 is a filled map with the above SOP output and above set of six minterms.

TABLE 3.10 Four Variable Karnaugh Map for $S = 1$ —an output when $SOP = \sum m (2, 3, 6, 7, 14, 15)$

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

Remember the following Operations

- When minterm is present in an SOP expression, we place 1 at the corresponding cell for the term.
- Complete the step 1 for all the rows for the minterms of a Boolean expression or logic circuit corresponding to that.

3.2.3 Karnaugh Map from the Maxterms in a POS

Recall Sections 2.7.4 to 2.7.6. Recall the Tables 2.6 to 2.8. A four-input truth table for the output from a logic circuit can also be expressed as maximum 16 POS maxterms. Therefore, Karnaugh map can also be made from the maxterms corresponding to a four input logic circuit.

Table 3.7 showed an unfilled Karnaugh map. A four variable Karnaugh map can be a two-dimensional map built from the 16 maxterms for a four input truth table. The cells are arranged as per Table 3.11. It shows the maxterm numbers corresponding to the cells. Since maximum number of minterms in a four variables (four inputs) POS can be 16, the map has 16 cells; four cells horizontal and four cells vertical.

TABLE 3.11 Four variables Karnaugh Map for maximum 16 Maxterms in a POS expression

$A + B \backslash C + D$	$C + D$	$C + D$ 00	$C + \bar{D}$ 01	$\bar{C} + \bar{D}$ 11	$\bar{C} + D$ 10
$A + B$	00	$M(0)$	$M(1)$	$M(3)$	$M(2)$
$A + \bar{B}$	01	$M(4)$	$M(5)$	$M(7)$	$M(6)$
$\bar{A} + \bar{B}$	11	$M(12)$	$M(13)$	$M(15)$	$M(14)$
$\bar{A} + B$	10	$M(8)$	$M(9)$	$M(11)$	$M(10)$

Note: For map of maxterms, we just replace in Table 3.9 dot (AND) sign operation with + (OR) sign and replace each variable A, B, C and D by their respective complements.

Points to Remember

Row 1st has cells for AB as 00. It corresponds $A + B$ part containing the maxterms $M(0)$ to $M(3)$.

Row 2nd has cells for AB as 01. It corresponds $A + \bar{B}$ containing maxterm $M(4)$ to $M(7)$.

Row 3rd has cells for AB as 11. (Complement of row 1st). It corresponds $\bar{A} + \bar{B}$ containing maxterms $M(12)$ to $M(15)$.

Row 4th has cells for AB as 10. (Complement of row 2nd). It corresponds $A + B$ containing maxterms $M(8)$ to $M(11)$.

Column 1st has cells for CD as 00. It corresponds $C + D$ part containing maxterms.

Column 2nd has cells for CD as 01. It corresponds $C + \bar{D}$ containing maxterms.

Column 3rd has cells for CD as 11. (Complement of column 1st). It corresponds $\bar{C} + \bar{D}$ containing maxterms.

Column 4th has cells for CD as 10. (Complement of column 2nd). It corresponds $\bar{C} + D$ containing maxterms.

Let in a four variable Maxterms present are 0, 1, 2, 3 and 10 and 11— $(A + B + C + D)$ is present, $(A + B + C + \bar{D})$ is present, $(A + B + \bar{C} + \bar{D})$ is present, $(A + B + \bar{C} + D)$ is present, $(\bar{A} + B + \bar{C} + D)$ is present and $(\bar{A} + B + \bar{C} + \bar{D})$ is present in a particular four input logic circuit output under different input states. It means $\bar{P} = \prod M(0,1,2,3,10,11)$. We place 0s as per Table 3.12.

Table 3.12 is a filled map with the above POS output and above set of six maxterms.

Remember the following Operations

- When maxterm is present in a POS expression, we place 0 at the corresponding cell for the term.
- Complete the step 1 for all the rows for the maxterms of a Boolean expression 0 or logic circuit corresponding to that.

3. From a map for SOP, for a map of the POS maxterms, we just consider a dot (AND) sign operation replaced with the + (OR) sign and each variable A, B, C and D just replaced by their respective complements. Also instead of placing 1s, we place 0s in POS Karnaugh maps.

TABLE 3.12 Four variable Karnaugh Map for $\bar{P} = 0$ an output when $POS = \prod M(0, 1, 2, 3, 10, 11)$

$A + B \backslash C + D$	$C + D$	$C + D$	$C + \bar{D}$	$\bar{C} + \bar{D}$	$\bar{C} + D$
$A + B$	00	00	01	11	10
$A + \bar{B}$	01				
$\bar{A} + \bar{B}$	11				
$\bar{A} + B$	10			0	0

Note: For map of maxterms, we just fill 0s at the map using Table 3.11.

3.3 FIVE AND SIX VARIABLE KARNAUGH MAPS AND TABLES

When the number of variables are five, we can make two maps, each with four variables, B, C, D and E with fifth variable $A = 0$ and $A = 1$, respectively. One map is considered as a upper layer map and other is considered as a lower layer map, when performing the minimization from the cell adjacencies. Table 3.13 shows a five variable Karnaugh map set.

Table 3.14 shows how to construct a six variable map. A six variable 64 cells Karnaugh map can be made a pair of two five variable map one upper for $A = 0$ and other lower for $A = 1$ each with 32 cells. It can also be made from a quad of four 4-variable maps; left upper for $A = 0, B = 0$, right upper for $A = 0, B = 1$, left lower for $A = 1, B = 1$ and right lower for $A = 1$ and $B = 0$, respectively. We can place one layer over another of four variable map in three dimension in the order, left upper, right upper, left lower and right lower, respectively. Total maximum minterms can be 64 for a six variable map.

TABLE 3.13 Five variables Karanaugh Map set with 32 minterms of the SOP expression and upper layer for $A = 0$ on the left side and lower layer for $A = 1$ on the right side

$A = 0$					$A = 1$				
$DE \backslash DE$	$\bar{D} \bar{E}$	$\bar{D} E$	$D \bar{E}$	$D E$	$DE \backslash DE$	$\bar{D} \bar{E}$	$\bar{D} E$	$D \bar{E}$	$D E$
BC	00	01	11	10	BC	00	01	11	10
$BC\ 00$	$m(0)$	$m(1)$	$m(3)$	$m(2)$	BC	$m(16)$	$m(17)$	$m(19)$	$m(18)$
$BC\ 01$	$m(4)$	$m(5)$	$m(7)$	$m(6)$	BC	$m(20)$	$m(21)$	$m(23)$	$m(22)$
$BC\ 11$	$m(12)$	$m(13)$	$m(15)$	$m(14)$	BC	$m(28)$	$m(29)$	$m(31)$	$m(30)$
$BC\ 10$	$m(8)$	$m(9)$	$m(11)$	$m(10)$	BC	$m(24)$	$m(25)$	$m(27)$	$m(26)$

TABLE 3.14 Six-variables Karnaugh Map set with 64 minterms of the SOP expression and upper layer for $A = 0$ on the upper side and lower layer for $A = 1$ on the lower side

$A = 0, B = 0$					$A = 0, B = 1$				
$\bar{E}F$	$E\bar{F}$	$\bar{E}\bar{F}$	EF	$E\bar{F}$	$\bar{E}F$	$\bar{E}\bar{F}$	$\bar{E}\bar{F}$	EF	$E\bar{F}$
CD	00	01	11	10	CD	00	01	11	10
$\bar{C}D$	00	$m(0)$	$m(1)$	$m(3)$	$m(2)$	$\bar{C}D$	00	$m(16)$	$m(17)$
$\bar{C}D$	01	$m(4)$	$m(5)$	$m(7)$	$m(6)$	$\bar{C}D$	01	$m(20)$	$m(21)$
CD	11	$m(12)$	$m(13)$	$m(15)$	$m(14)$	CD	11	$m(28)$	$m(29)$
$\bar{C}D$	10	$m(8)$	$m(9)$	$m(11)$	$m(10)$	$\bar{C}D$	10	$m(24)$	$m(25)$
$A = 1, B = 1$					$A = 1, B = 0$				
$\bar{E}F$	$E\bar{F}$	$\bar{E}\bar{F}$	EF	$E\bar{F}$	$\bar{E}F$	$\bar{E}\bar{F}$	$\bar{E}\bar{F}$	EF	$E\bar{F}$
CD	00	01	11	10	CD	00	01	11	10
$\bar{C}D$	00	$m(48)$	$m(49)$	$m(51)$	$m(50)$	$\bar{C}D$	00	$m(32)$	$m(33)$
$\bar{C}D$	01	$m(52)$	$m(53)$	$m(55)$	$m(54)$	$\bar{C}D$	01	$m(36)$	$m(37)$
CD	11	$m(60)$	$m(61)$	$m(63)$	$m(62)$	CD	11	$m(44)$	$m(45)$
$\bar{C}D$	10	$m(56)$	$m(57)$	$m(59)$	$m(58)$	$\bar{C}D$	10	$m(40)$	$m(41)$

3.4 AN IMPORTANT FEATURE IN THE DESIGN OF A KARNAUGH MAP

3.4.1 Only Single Variable Changes into Its Complement in a Pair of Adjacent Cells

We note an important observation in the Karnaugh map designs (Tables 3.1 to 3.13). In a cell adjacent to any cell, whether on left or right or on up or down (not diagonal) has only a single-variable changes into its complement.

In three-variable map, first row first column cell only \bar{C} changes to C on moving to second column. In fact, each row; \bar{C} is changing C in the right side adjacent cell (Tables 3.1 to 3.4).

In cells of row 1 and row 2, each column, \bar{B} is changing to B at the down side adjacent cell. From row 2 to 3, \bar{A} is changing to A at the down side cell. From 3 to 4, B is changing to \bar{B} at the down side cell.

In four variable map also, there is only variable change in adjacent cell to any cell, whether on left or right or on up or down (not diagonal). (Tables 3.7 to 3.12). A single-variable change occurs into its complement.

Table 3.15 left side upper portion shows the adjacent pairs ($\bar{A}B\bar{C}D$ and $\bar{A}BCD$) and ($\bar{A}BC\bar{D}$ and $\bar{A}B\bar{C}D$).

Therefore, we can form in a specific case a pair of adjacent cells having 1s at left or right or at up or down (not diagonal) and adjacency condition is that only one variable is distinct by its complement.

3.12 Digital Systems: Principles and Design

TABLE 3.15 Adjacent cell pairs, adjacent four cells, adjacent eight cells in four variable Karnaugh Maps with 16 minterms each in the SOP expression

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

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

Note: $\bar{A} \bar{B} \bar{C} \bar{D}$ cell is also taken as adjacents to $A \bar{B} \bar{C} \bar{D}$ cell, $\bar{A} \bar{B} \bar{C} \bar{D}$ and $\bar{A} \bar{B} \bar{C} \bar{D}$ is a pair cell as there is only one variable complementation on going to adjacent cell.

3.4.2 Only Two Variables Change into Their Complements in Adjacent Cells in a Square or Column of Four Cells

In four adjacent cells, whether horizontally or vertically or in a square with four cells, two variables change into their complements.

In three-variable map (Table 3.3), in a set (1st row 1st column, 1st row 2nd column, 2nd row 1st column, 2nd row 2nd column) two variables \bar{C} and \bar{B} change to C and B in the square with four cells and A is same for all the four cells. In four cells of first column, two variables, A and B change into their complements and \bar{C} remains same. In second column C remains same.

In four variable map also, there are only two variable changes in a square or column change within adjacent cell to any cell, whether on left or right or on up or down (not diagonal) and other two remain identical (Table 3.5 to 3.8). The first column cell and last column cell in a row are considered adjacent cells.

Similary, first row cell and last row cell in a column are also the adjacent cells.

Therefore, we can from in a specific case either a square or column of four adjacent cells having 1s in all and only two variables are distinct by their complements and remaining ones remain unchanged. Adjacency condition is that only two variable changes occur within the adjacent cells.

Table 3.15 right side upper portion shows examples of the square and column of the four adjacent cells.

3.4.3 Three Variables Change into Their Complements in Adjacent Cells in Box of Eight Adjacent Cells

In eight adjacent cells in a box, only three variables have their complements also and remaining variables (present in four variable or higher variable maps) remain common to all the cells (Table 3.1 to 3.8).

Therefore, we can form in a specific case a 8-cell box of adjacent cells having 1s in all and only three variables are distinct by their complements and remaining one(s) remains unchanged.

Table 3.15 left side lower shows example of the box of adjacent eight cells and right lower shows the adjacent 4 cells and 2 cells.

3.4.4 First and Last Columns for First and Last Rows and Purpose of Deciding Adjacency in a Karnaugh Map

We note an important observation in the Karnaugh map design. (Tables 3.1 to 3.13).

1. Two cells, one each at the upper most row and the lower most row can also be considered as adjacent if we wrap the map in a horizontal axis cylindrical form and there is only a single-variable that changes into the complement when we consider two cells of the same column in the upper most and lower most rows.
2. Two cells, one each at the left most column and the right most column can also be considered as adjacent if we wrap the map in a vertical axis cylindrical form and there is only a single-variable that changes into its complement when we consider two cells of the same row in the left most and rightmost columns.
3. Four cells distributed at the upper rows and lower rows can also be considered as adjacent if we wrap the map in a horizontal axis cylindrical form and there are only two variable(s) that change into their complements in these cells. Other variable(s) remain common in these cells.
4. Four cells distributed at the left-most column and the right-most column can also be considered as adjacent if we wrap the map in vertical axis cylindrical form and there are only two variables that change into their complements in these cells. Other variable(s) remain common in these cells.
5. Eight cells distributed in the upper rows and lower rows can also be considered as adjacent if we wrap the map in a horizontal axis cylindrical form and there are three variables that change into the complements when we consider four cells at same column(s) in the upper most and lower most rows.
6. Eight cells distributed in the left most column and right most column can also be considered as adjacent if we wrap the map in a vertical axis

cylindrical form and there are only three variables that change into the complements when we consider four cells at the same row(s) in left and right columns.

Table 3.16 shows wrapping adjacencies between the two cells, four cells and eight cells at four Karnaugh maps.

TABLE 3.16 Wrapping adjacencies of three cell pairs (left upper), two four cell quads (right upper), two eight cell octets (left lower) and two wrapping adjacencies (right-lower) at the four different four variable Karnaugh Maps with 16 minterms each for the SOP expressions

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

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

3.4.5 Use of Don't Care (or Unspecified) Input Conditions for Purpose of Deciding Adjacencies in a Karnaugh Map

Don't care condition means that a combination of input states do occur and whether the outputs for those states taken as 1s or 0s, it does not matter. We can thus place 1s at the corresponding minterm places in SOP form of the Karnaugh map. The 1s are placed only at the places where it leads to make or improve adjacencies. We can also place 0s at the corresponding maxterm places in POS form of the Karnaugh map. The 0s are placed only at the places where it leads to make or improve adjacencies. Better way is that we place an 'x' for the don't care input states. Use it at places where it leads to simplification and leave it where it does not lead to simplification. The improved adjacencies lead to simplification of the circuit design.

We can take x for determining the adjacent pairs or quads or octets. Table 3.17 shows the use of don't care in selecting conditions the adjacencies. Assume that $A.B.C.D$ and $A.B.C.D$ and $A.B.C.D$ are don't care input states $[(A = 0, B = 1, C = 0 \text{ and } D = 0), A = 0, B, C \text{ and } D = 1], \text{ and } (A = 1, B = 0, C = 0 \text{ and } D = 1)]$.

We can use the two don't care conditions to make two octets and leave and ignore third one x at 2nd row and third column place in the map.

TABLE 3.17 Use of don't care input combinations for determining the adjacencies

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

Now one octet is by columns 1 and 4 adjacencies and other octet is by rows 1 and 4 adjacencies.

3.5 SIMPLIFICATION OF LOGIC CIRCUIT RELATION BY MINIMIZATION USING ADJACENCIES

3.5.1 Minimization of a Karnaugh Map Using Pairs of Adjacent Cells

In Sections 3.4.1 and 3.4.4, we learnt that pair of cells has one of the variable as the complements of each other. Hence only common variable in the pair of terms needs to be retained and that variable removed after simplification of two terms as one. This follows from the application of the Boolean OR and AND rules; $X + X = 1$ and $X \cdot 1 = X$.

A pair simplifies by reducing two minterms as one in an SOP. The procedure for removing one variable and simplifying two terms in SOP into one can be understood as follows by an exemplary Karnaugh map in Table 3.18.

Table 3.18 shows the adjacent pairs of cells.

There are five places where there are 1s. Map corresponds to five minterms (Table 3.3) and following SOP expression.

$$S = \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \bar{D} + A \cdot \bar{B} \cdot \bar{C} \cdot \bar{D} + \bar{A} \cdot B \cdot C \cdot \bar{D} + A \cdot B \cdot C \cdot \bar{D} + A \cdot \bar{B} \cdot C \cdot \bar{D} \quad \dots(3.1)$$

First and second terms are for the adjacent cells (assumed wrapped map) and have ' A ' variable as complements. Hence, A is removed from first two terms and $B \cdot C \cdot \bar{D}$ is left. Third and fourth terms are for the adjacent cells at the middle of the last column. These have A variable as complements. Hence, A is removed from these two terms and $B \cdot C \cdot \bar{D}$ is left. Fifth term has a wrapping adjacency pair with $A \cdot \bar{B} \cdot \bar{C} \cdot \bar{D}$ pair. Therefore, it can't be removed but can be simplified as $A \cdot \bar{B} \cdot \bar{D}$. (a 3-variable term).

TABLE 3.18 A Karnaugh Map with two pairs of adjacent cells

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

Simplified Boolean expression becomes as follows.

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

3.5.2 Minimization of a Karnaugh Map Using Quads of Four Adjacent Cells

In Sections 3.4.2 and 3.4.4, we learnt that quad of four adjacent cells has two of the variables as the complements of each other. Hence only common variables in four terms needs to be retained and those two variables are removed for the simplification of logic circuit. Thus three minterms can be reduced from the four and only one is then needed for designing a circuit. It follows from Boolean OR and AND rules on both of them; $\bar{X} + X = 1$, $\bar{Y} + Y = 1$, $X \cdot 1 = X$ and $Y \cdot 1 = Y$.

Table 3.19 shows the adjacent quads of four adjacent cells.

This procedure can be understood as follows by an exemplary Karnaugh map in Table 3.19. There are nine places where there are 1s. Map corresponds to nine minterms and following SOP expression.

$$\begin{aligned} S = & \bar{A}\bar{B}\bar{C}.D + \bar{A}.B\bar{C}.\bar{D} + \bar{A}.B\bar{C}.D + \bar{A}.B.C.\bar{D} + A.B.\bar{C}.\bar{D} + A.B.\bar{C}.D \\ & + A.B.C.\bar{D} + A.\bar{B}\bar{C}.D + A.\bar{B}.C.\bar{D} \end{aligned} \quad \dots(3.2)$$

TABLE 3.19 A Karnaugh Map with two quads of four adjacent cells

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

Four terms (1st, 3rd, 6th and 8 in the SOP Equation) for second column adjacent cells have A and B variables as well as their complements. Hence, A and B

are removed from first four terms and $\bar{C}D$ is left. Four terms are also for the four adjacent cells (after wrapping the map into cylinder with a vertical axis). These have A and C variables as well as their complements. Hence, A and C are removed from these four terms and $B\bar{D}$ is left. Ninth term has a pair with the seventh and B can be removed.

Simplified Boolean expression becomes as follows.

$$S = \bar{C}D + B\bar{D} + A\bar{B}C\bar{D} = \bar{C}D + B\bar{D} + A\bar{C}\bar{D} \quad \dots(3.3)$$

3.5.3 Minimization of a Karnaugh Map Using Octet of Eight Adjacent Cells

In Section 3.4.3, we learnt that octet of eight adjacent cells has three variable as well as their complements present. Hence only common variable(s) in eight terms needs to be retained and those three variables are removed for the simplification of logic circuit. It follows from Boolean OR and AND rules on both of them; $X+X=1$, $Y+Y=1$, $Z+Z=1$, $X\cdot 1=X$, $Y\cdot 1=Y$ and $Z\cdot 1=Z$. The number of minterms reduces from 8 to 1 only. This procedure can be understood as follows by an exemplary Karnaugh map in Table 3.20.

Table 3.20 shows an octet of adjacent eight cells.

There are nine places where there are 1s. Map corresponds to nine minterms and following SOP expression.

$$\begin{aligned} S = & \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} \\ & + \bar{A}B\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D} \end{aligned} \quad \dots(3.4)$$

TABLE 3.20 A Karnaugh Map with an octet of eight adjacent cells

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

Eight terms (all except 2nd Equation (3.4)) for first and fourth column adjacent cells (from wrapping adjacencies) have A , B and C variables as well as their complements. Hence, A , B and C are removed from the eight terms and only D is left. Second term has a pair with third. Therefore, it can be simplified. Answer after simplification is as follows:

$$S = \bar{D} + \bar{A}\bar{B}C\bar{D} = \bar{D} + \bar{A}\bar{B}C \quad \dots(3.5)$$

3.5.4 Minimization of a Karnaugh Map Using Offset Adjacencies and Diagonal Adjacencies

Consider the following map in Table 3.21.

TABLE 3.21 A Karnaugh map with an offset adjacency and a diagonal adjacency

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

We note that rows 3 and 4 have a diagonal adjacency. \bar{C} and A are common variables between and two minterms sum will have the term $B.D$ and $\bar{B}.D$ after Boolean simplification. The operation equals an XOR operation. Hence, simplification gives $(A.C).(B.\text{XOR}.D)$.

We also note that columns 1 and 3 have an offset adjacency between the four cells. A variable is common between these. B variable has its complement \bar{B} and is removable. So answer is $A.(C.D + C.D) = A(C.\text{XNOR}.D)$. [The XNOR logic gate and truth table was given in Figure 3.2(c).]

Simplified expression from the map is as follows:

$$S = (A.\bar{C})(B.\text{XOR}.D) + \bar{A}.(C.\text{XNOR}.D). \quad \dots(3.6)$$

When XOR and XNOR form of simplifications are needed then the diagonal and offset adjacencies, respectively, are used.

3.5.5 Minimization by Finding Prime Implicants

The definitions and technique for minimizing a Boolean SOP expression or truth table (also called finding **prime implicants**) are as follows—

1. A variable in complemented as well as in un-complemented format is used in a Boolean expression. It can also be called a **literal**.
2. A product term (minterm) present in a function (Boolean expression) can also be called **implicant** and its function = 1. An implicant is implemented by the AND gates at the inputs (first level).
3. A **cover** means a set of all implicants that contains all the implicants (minterms) whose function values are 1s and that are needed to complete the map.
4. A function is implemented by ANDs in all the *implicants* at the *cover* and using the OR gates at the second level.

5. An AND (first level)-OR (second level) circuit can be converted to an NAND—NAND circuit at both first and second levels. (Refer use of DeMorgan Theorem).
6. A **prime implicant** means an implicant, which can't be further simplified into another implicant with less number of literals in it. It can't be ORed with another implicant to get less number literals (variables).
7. *Prime implicants* give a simpler minimized circuit than the circuits using *implicants*.

1. Convert a Boolean expression into standard SOP format or add more truth table rows till table has 2^n rows for n -inputs (variables to be used for making the map) with output marked as x . (x means don't care) for added rows.
2. Use x as 1 when forming pairs, quads and octets. Remove x from the other remaining cells at the map.
3. If $n = 3$, construct a three variable Karnaugh map and put the 1s at the places corresponding to the minterms present (implicants) or truth table rows corresponding to the output = 1.
4. If $n = 4$ or 5 or 6, construct a 4 or 5 or 6 variable Karnaugh map and put the 1s at the places corresponding to the minterms present (implicants) or truth table rows corresponding to the output = 1.
5. Find the adjacency cell octets of 1s in the map. Consider wrapping adjacencies also. It will simplify the eight terms for an octet into one term with three variables removed.
6. Find the adjacency cell quads of 1s in the map. Consider wrapping adjacencies also. It will simplify the four terms for a quad into one term with two variables removed.
7. Find the adjacency of cell pairs of 1s in the map. Consider wrapping adjacencies also. It will simplify the two terms for a pair into one term with two variables removed.
8. Find the diagonal adjacency(s) also in case XOR gate(s) are also to be used.
9. Find the offset adjacency(s) also in case XNOR gate(s) are also to be used.

Steps for finding the prime implicants, called minimization technique (minimizing a Boolean POS expression or truth table) using POS based is as follows Karnaugh Map:

1. Convert a Boolean expression into Standard POS format or add more rows in the truth table rows in incompletely specified till it has 2^n rows for n -inputs (variables to be used for making the map) with output marked as x . (x means don't care).
2. A sum term (maxterm) present in a function (Boolean expression) gives the output = 0 by its function = 0. A maxterm present in the map is implemented by the OR gates at the inputs (first level).

3. A cover means a set of all the maxterms whose function values are 0s and that are needed to complete the map.
4. A function is implemented by ORs in all the maxterms present and using the AND gates at the second level.
5. An OR (first level)-AND (second level) circuit can be converted to an NOR–NOR circuit at both first and second levels (Refer use of DeMorgan Theorem).
6. If $n = 3$, construct a three variable Karnaugh map and put the 0s at the places corresponding to the maxterms present or find those truth table rows that correspond to the output = 0.
7. If $n = 4$ or 5 or 6, construct a 4 or 5 or 6 variable Karnaugh map and put the 1s at the place corresponding to the maxterms present or truth table rows corresponding to the output = 0 .
8. Use x as 0 when forming pairs, quads and octets. Remove x from the other remaining cells at the map.
9. Find the adjacency cell octets of 0s in the map. Consider wrapping adjacencies also. It will simplify the eight terms for an octet into term with three variables removed.
10. Find the adjacency cell quads of 1s in the map. Consider adjacencies also. It will simplify the four terms for a quad into one term with two variables removed.
11. Find the adjacency cell pairs of 0s in the map. Consider wrapping adjacencies also. It will simplify the two terms for a pair into one term with 2 variables removed.

3.6 DRAWING OF LOGIC CIRCUIT USING AND-OR GATES, OR-AND GATES, NAND'S ONLY, NOR'S ONLY

Karnaugh map in SOP function corresponds to 1. We can use the minterm for each cell by for making a circuit from AND gates. Later on all the outputs of AND gates are ORed. For example, we can use Table 3.19 Karnaugh map for the minterms and we get an AND–OR circuit (at first and second level, respectively). This is demonstrated in Figure 3.1.

Karnaugh map in POS function corresponds to 0. We can use the maxterm for each cell by for making a circuit from OR gates. Later on all the outputs of OR gates are ANDed. For example, we can use Table 3.12 Karnaugh map maxterms and we get an OR–AND circuit (at first and second level, respectively). This is demonstrated in Figure 3.2.

Using DeMorgan theorem, an AND-OR circuit can be converted to NANDs only circuit. This is demonstrated in Figure 3.3. Using DeMorgan theorem, an OR-NAND circuit can be converted to NORs only circuit. This is demonstrated in Figure 3.4.

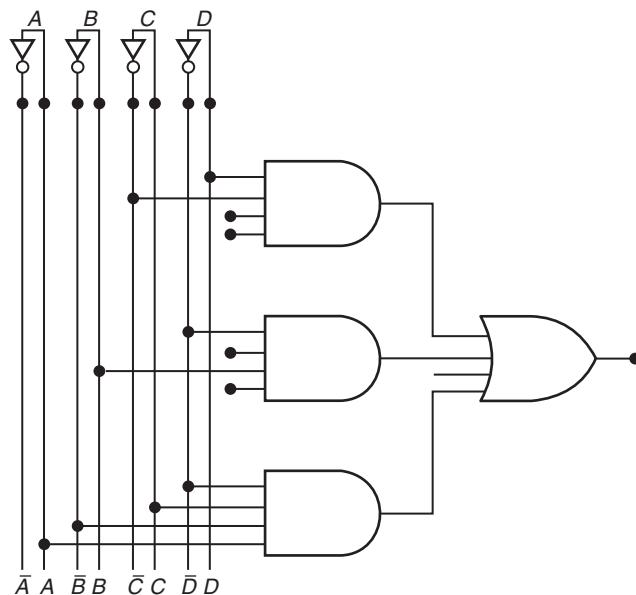


FIGURE 3.1 An AND-OR circuit Representation of Karnaugh map in Table 3.19 after simplification.

3.7 REPRESENTATIONS OF A FUNCTION (COVER) FOR A COMPUTER-AIDED MINIMIZATION FOR SIMPLIFYING THE LOGIC CIRCUITS

When simplifying and finding the prime implicants manually and up to fewer variables (six), the Karnaugh maps are suitable. For complex circuits, the computer-aided minimization is used.

3.7.1 Representation in Cube Format for Computer-aided Minimization

A cube has three axes, X -axis, Y -axis and Z -axis. Figure 3.5 shows a cube. It has 8 vertices. The origin vertex coordinates are 000, to represent $\bar{A}\bar{B}\bar{C}$. The other 3 vertices along the axes have coordinates are 100, 010, 001 (corresponding to $A\bar{B}\bar{C}$, $\bar{A}B\bar{C}$ and $\bar{A}\bar{B}C$). In three $X-Y$, $Y-Z$ and $Z-X$ planes, the other vertices are 110, 011, 101 (corresponding to $A\bar{B}\bar{C}$, $\bar{A}B\bar{C}$ and $A\bar{B}C$). One vertex along the cue diagonal is 111 (corresponding to $A.B.C$). Thus there are eight coordinates, each corresponding to a cube vertex and each corresponding to a minterm (an implicant) in SOP.

Each vertex is marked if the function value = 1 of the corresponding implicant. Two marked vertices are joined together if these are along an axis. Joined marked vertices are shown by dark or different coloured axis in the cube. Figure 3.5 shows the cubical representation for a cover with the implicants given by

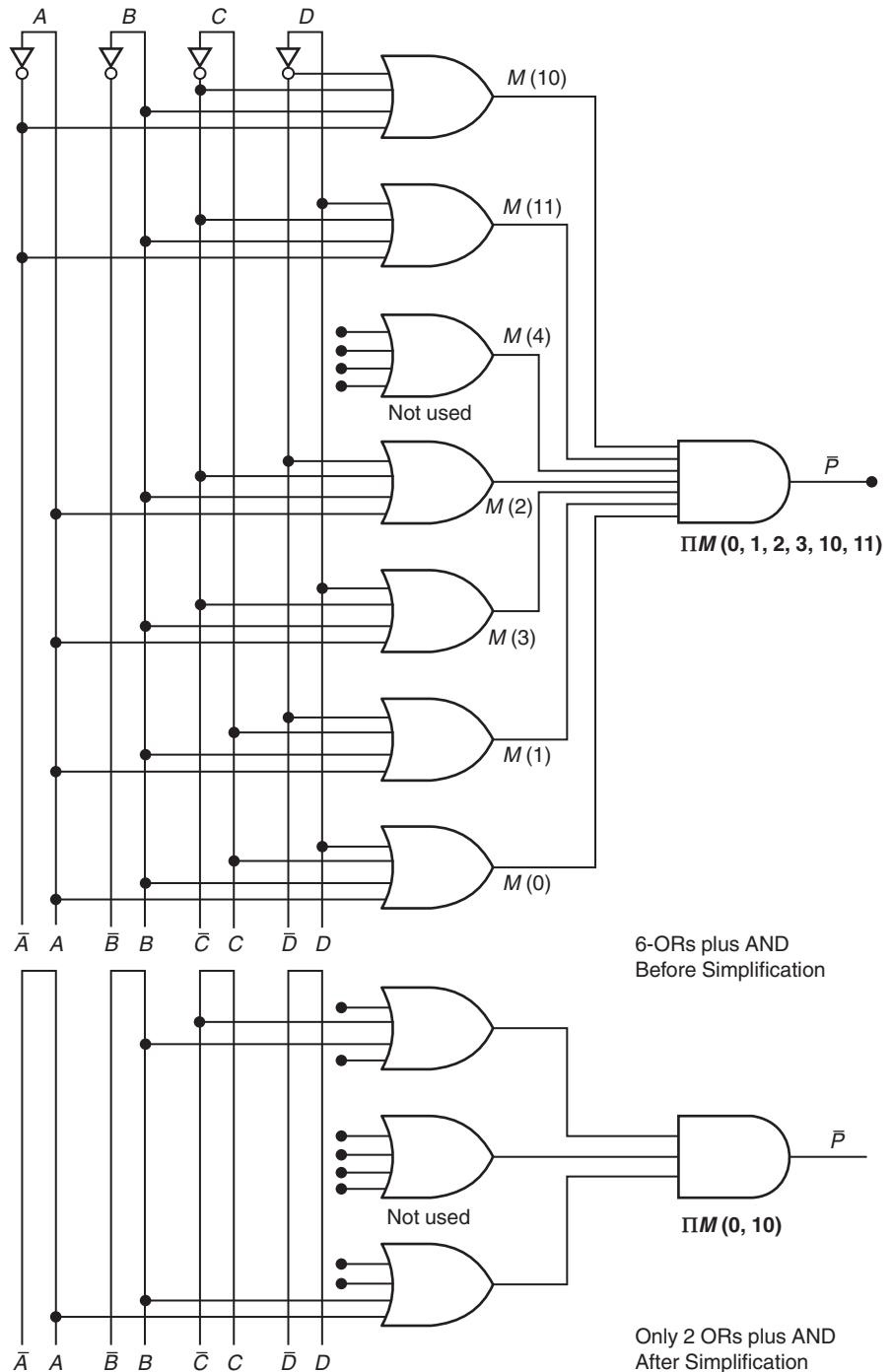


FIGURE 3.2 Using two quads, taking $(2, 3, 10, 11)$ wrapping adjacencies also in account, the OR-AND circuit a representation of Karnaugh map of Table 3.12 before and after simplification.

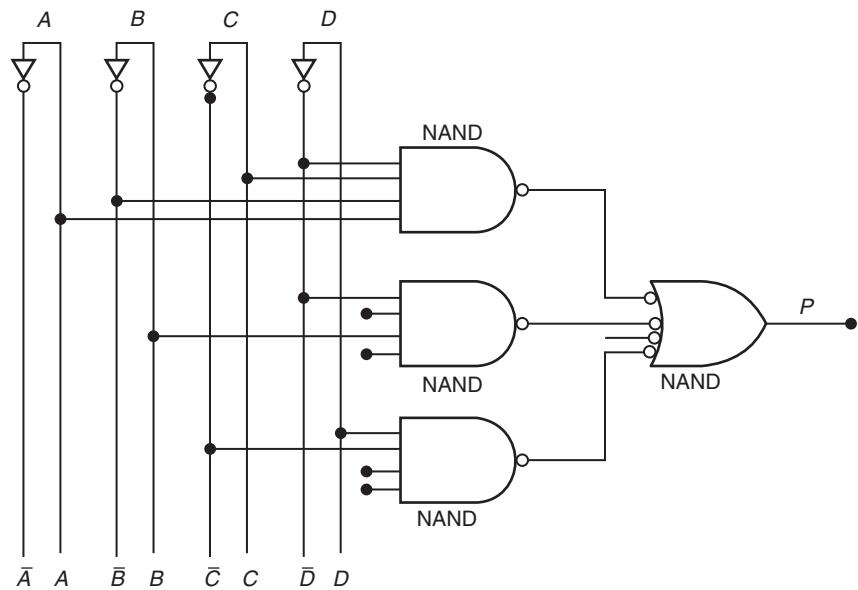


FIGURE 3.3 Representation of AND-OR circuit in Figure 3.1 into NANDs only circuit. [OR gate with all bubbled input is equivalent to NAND (deMorgan Theorem)].

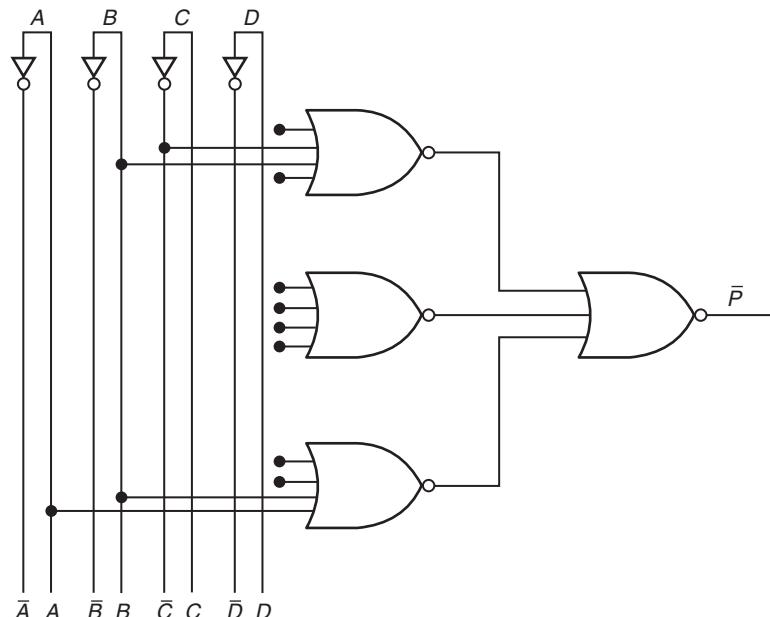


FIGURE 3.4 Representation of an OR-AND circuit in Figure 3.2 by NORs only.

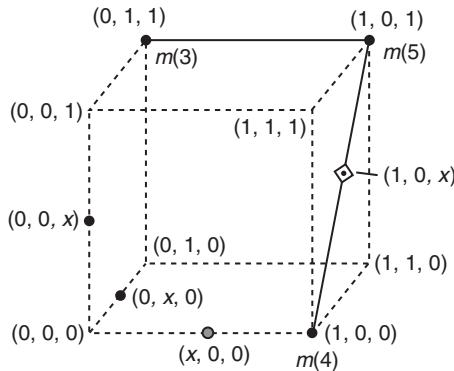


FIGURE 3.5 Representation of implicants (minterms) in a cube format. Each vertex, which is having the function value = 1 of the implicant, is marked. Marked two vertices are joined together if these are along an axis. (Dark lines in the figure.)

following Equation (3.7). It is for SOP function, S corresponding to Karnaugh map in Table 3.22.

$$\begin{aligned} S &= \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot \bar{C} + A \cdot \bar{B} \cdot C = \sum m(4,3,5) \\ &= (011,100,101) \end{aligned} \quad \dots(3.7)$$

Now, pair 100, 101 can be written as $10x$.

The coordinate of mid-point of a dark Z-axis starting from 100 is $(1, 0, 0.5)$. Let us write it as $10x$. Similarly we can write and find other dark axes mid points. For example, X-axis mid point is $x00$, when it starts from 000.

If XY plane passing through 000 has all its line as dark, its mid point can be written as $xx0$. If XY plane passing through 001 has all its line as dark, its mid point can be written as $xx1$. Similarly we can write mid point of all planes having the dark axes at the cube.

Mid point of the cube can be written as (x, x, x) when all the eight vertices are marked and all the cube axes are dark.

TABLE 3.22 Three variable Karnaugh Map

$AB \backslash C$	C	\bar{C}	C
AB			
$\bar{A} \bar{B}$	00		
$\bar{A} B$	01		1
$A B$	11		
$A \bar{B}$	10	1	1

The equation (3.7) is rewritten as

$$S = \{10x, 011\} \quad \dots(3.8)$$

(When the pair of vertices 100, 101 are written by its axis mid-point as 10x in Figure 3.5)

We can see below that finding prime implicants becomes very easy using the representation as a cube if we follow following steps and these steps can also be programmed and minimization is possible by that program.

1. Write the implicants of a function, S in terms of the coordinates of the vertices.
2. Find a pair in which X coordinates only change and are 0 and 1. Reduce it to one term with coordinate = x . Do the same with Y and Z . For example, convert the pair (1, 0, 1) and (1, 1, 1) into (1, x , 1).
3. Continue till all pairs are taken care (It is equivalent to replacing a pair of vertices by an axis mid-point).
4. Again pair the newly found coordinates with one of its coordinate marked as x also. For example, pair (0, x , 0) and (0, x , 1) into (0, x , x) (It is equivalent to replacing a pair axes mid-points by a plane mid-point).
5. Again pair the newly found coordinates with two of its coordinates marked as x also. For example, pair (x , x , 0) and (x , x , 1) into (x , x , x). (It is equivalent to replacing a pair of planes mid-points at a the cube mid-point).

Assume that a circuit is that its Boolean expression in SOP format has minterms 4, 5, 6 and 7. Therefore, S can be expressed in a cube coordinates' format as follows:

$$S = \{(111, 110, 101, 100)\} \quad \dots(3.9)$$

The steps which are implemented by a computer-based minimization, are as follows:

$$\begin{aligned} S &= (1, 1, 1), (1, 1, 0), (1, 0, 1), (1, 0, 0) \\ &= (1, 1, x), (1, 0, x) \end{aligned}$$

(Pairing 1st and 2nd and 3rd and 4th set of coordinates. It is equivalent to replacing a pair of vertex coordinates by an axial mid-point).

$$= (1, x, x) \quad \dots(3.10)$$

(Pairing 1st and 2nd.) It is equivalent to replacing a pair of axial mid-points by a plane midpoint.

The minimized function is now

$$S = A \quad \dots(3.11)$$

The A is prime implicant of the four implicants of Equation (3.9). The logic circuit is just output equal to input A . $S = A$ for minterms $\Sigma m(0, 1, 2, 3)$ case of implicants, then simplified minimized circuit is just a not operation on A .

3.7.2 Representation in Four-Dimensional Hypercube Formats for a Computer-aided Minimization

A four dimensional hyper cube has four axes, A_1 -axis, A_2 -axis, A_3 and A_4 -axis. It will have 16 vertices. The origin vertex coordinates are 0000. The other coordinates will be 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110 and 0111.

The coordinate of mid-point on a darkened axis from 1000 to 1001 is (1, 0, 0, 0.5) (An axis is darkened if there are vertices falling on it). Let us write it as 100x. Similarly, we can write, we can find other dark axes mid-points. For example, A_1 -axis mid-point is x000, when it starts from 0000 to 1000.

If A_1 - A_2 plane passing through 0000 has all its line as dark, its mid point can be written as xx00. If A_1 - A_2 plane passing through 0001 has all its line as dark, its mid point can be written as xx01. We can similarly write mid-point of all planes with the dark axes in a four dimensional cube.

Mid-point of the four dimensional cube can be written as (x, x, x, x) when all the 16 vertices are marked and all the axes are dark.

Consider the following four literal (variable) SOP equation:

$$\begin{aligned} S = & \{\bar{A}.\bar{B}.\bar{C}.\bar{D} + \bar{A}.\bar{B}.C.D + \bar{A}.B.\bar{C}.\bar{D} + \bar{A}.B.C.D + A.\bar{B}.\bar{C}.\bar{D} \\ & + A.B.\bar{C}.D\} = \Sigma m(0, 3, 4, 7, 8, 13) \end{aligned} \quad \dots(3.12)$$

Its Karnaugh map representation is same as given above in Table 3.21. This equation gives the implicants in four dimensional cube format as follows.

$$S = \{0000, 0011, 0100, 0111, 1000, 1101\} \quad \dots(3.13)$$

We can see below that finding prime implicants becomes very easy using the representation as a four dimensional cube if we follow following steps and these steps can also be programmed and minimization is possible by that program.

1. Write the implicants of a function, S in terms of the coordinates of the four dimensional cube vertices.
2. Find a pair in which one coordinate only changes from 0 to 1. Reduce the pair to one term with coordinate = x . For example, convert the pair (1, 0, 0, 1) and (1, 1, 0, 1) into (1, x , 0, 1).
3. Continue till all pairs are taken care. It is equivalent to replacing a pair of vertex coordinates by an axis mid-point.
4. Again pair the newly found coordinates with one of its coordinate being x . For example, pair (0, x , 0, 0) and (0, x , 1, 0) into (0, x , x , 0). It is equivalent to replacing a pair of axes mid-points by a plane mid-point.
5. Again pair the newly found coordinates with two of its coordinates being x . For example, pair (x , x , 0, 0) and (x , x , 1, 0) into (x , x , x , 0). It is equivalent to replacing a pair of plane mid-points by an internal cube mid-point to a four dimensional cube.

6. Again pair the newly found coordinates with three of its coordinates being x . For example, pair $(x, x, x, 0)$ and $(x, x, x, 1)$ into (x, x, x, x) . It is equivalent to replacing a pair of axes mid-points by the four dimensional cube mid-point.

Exemplary steps in computer based minimization (finding prime implicants) of Equation (3.13) based on four-dimensional cube are as follows

$$\begin{aligned} S &= \{(0, 0, 0, 0), (0, 0, 1, 1), (0, 1, 0, 0), (0, 1, 1, 1), (1, 0, 0, 0), (1, 1, 0, 1)\} \\ &= \{(0, x, 0, 0), (0, x, 1, 1), (1, 0, 0, 0), (1, 1, 0, 1)\} \end{aligned} \quad \dots(3.14)$$

(Pairing 1st and 3rd and 2nd and 4th set of coordinates).

The minimized function has four prime implicants:

$$\overline{A}.\overline{C}.\overline{D} + \overline{A}.C.D + A.\overline{B}.\overline{C}.\overline{D} + A.B.\overline{C}.D \quad \dots(3.15)$$

The AND-OR logic circuit is now minimum when using the Equation (3.15).

3.7.3 Representation in Hypercube (Multi-dimensional cube) Formats for Computer-aided Minimization

A hypercube (multidimensional dimensional cube) has n axes, A_1 axis, A_2 axis, A_3 , A_4 axis, ... up to A_n axis and is used for n literals (variables). It will have 2^n vertices that corresponds to 2^n minterms. The coordinates are easily written using 2^n sets of n -bit binary numbers between 0 and $2^n - 1$.

Let U and V are two cubes (or hypercubes) with coordinates u_i and v_i , respectively.

We can see below that finding prime implicants becomes very easy using the representation as a hypercube also if we follow following steps and these steps can also be programmed and minimization is possible by that program.

1. Write the implicants of a function, S in terms of the coordinates of the n dimensional hyper cube vertices.
2. Find a pair in which one of the coordinates change and that are 0 and 1. Reduce pair to one term with that coordinate = x . For example, convert the pair $(1, \dots, 0, 0, 1)$ and $(1, \dots, 1, 0, 1)$ into $(1, \dots, x, 0, 1)$.
3. Continue till all pairs are taken care. It is equivalent to replacing a pair of vertices coordinates by the axis mid-point.
4. Again pair the newly found coordinates with one of its coordinate as x also. For example, $(0, \dots, x, 0, 0)$ and $(0, \dots, x, 1, 0)$ into $(0, \dots, x, x, 0)$. It is equivalent to replacing a pair of axes mid-points by a plane mid-point.
5. Again pair the newly found coordinates with two of its coordinates as x also. For example, pair $(x, x, \dots, 0, 0)$ and $(x, x, \dots, 1, 0)$ into $(x, x, \dots, x, 0)$. It is equivalent to replacing a pair of axes mid-points by an internal cube mid-point to a four dimensional cube when $n = 4$.
6. Again pair the newly found coordinates with three of its coordinates as x also. For example, pair $(x, x, 1, \dots, x)$ and $(x, x, 0, \dots, x)$ into (x, x, x, \dots, x) . It is equivalent to replacing a pair of axes mid-points by the hypercube mid-point.

7. Continue searching and forming more and more pairs till we reach mid-point of $(n - 1)$ dimension cube.

We notice that computer based minimization approach is analogous, whether we use three cube (three variable) or four-cube or hypercube format. We write a cover (an SOP function) as the function of a set of coordinates on a hypercube before running the program for the steps given above.

3.8 MULTI-OUTPUT SIMPLIFICATION

Karnaugh map simplifies and gives reduced number of minterms or maxterms (product terms or sum terms) for an output of a combinational circuit. Often there are multiple outputs. For example, a binary to—segment decoder or 4-bit gray code converter. Let Y_0, Y_1, \dots, Y_{n-1} are the outputs for a set of input variable X_0, X_1, \dots, X_{m-1} . A truth table of these will have m columns for the inputs and n columns for the outputs. Number of rows will be equal to number of possible input combinations 2^m .

One method is to write Karnaugh map for the each and fabricate the separate circuit for each output. This does not minimize the cost as there are several terms that may be common and can be implemented by a common AND-OR sub-array. Other methods are as follows:

Method 1: Finding Common Set of Quads or Diads

Consider the Figures 3.6 (a) and (b). It shows implementation using the AND-ORs circuits for the two Karnaugh maps for the two outputs Y_0 and Y_1 in Tables 3.23 and 3.24, respectively. The tables show a dashed square envelope for common quad of the adjacent cells in the two maps. The Boolean expressions after Karnaugh minimization are $Y_0 = \bar{A} \cdot D$ and $Y_1 = \bar{A} \cdot D + B \cdot \bar{C} \cdot \bar{D}$.

Since $\bar{A} \cdot D$ quad (pair of four adjacent cells) is common in both, the simplified low cost circuit is as per Figure 3.6 (c). Number of gates reduces by two in the common implementation.

Method 2: Finding a Common Group of Terms

Consider the Figures 3.7 (a) to (e). It shows implementation using the AND-ORs circuits for the two Karnaugh maps for two outputs Y_0 and Y_1 in Tables 3.25 and 3.26,

TABLE 3.23 Karnaugh Map for $Y_0 = \bar{A} \cdot D$

$\bar{A} \bar{B}$	$\bar{C} \bar{D}$	$\bar{C} \bar{D}$	$\bar{C} D$	$C D$	$C \bar{D}$
$\bar{A} \bar{B}$	00	01	11	10	
$\bar{A} \bar{B}$	00		$[1]$	$[1]$	
$\bar{A} B$	01		$[1]$	$[1]$	
$A \bar{B}$	11				
$A B$	10				

TABLE 3.24 Karnaugh Map for $Y_1 = \bar{A} \cdot D + B \cdot C \cdot \bar{D}$

$\bar{A} \bar{B}$	$\bar{C} \bar{D}$	$\bar{C} \bar{D}$	$\bar{C} D$	$C D$	$C \bar{D}$
$\bar{A} \bar{B}$	00	01	11	10	
$\bar{A} \bar{B}$	00		$[1]$	$[1]$	
$\bar{A} B$	01		$[1]$	$[1]$	
$A \bar{B}$	11		$[1]$		
$A B$	10				

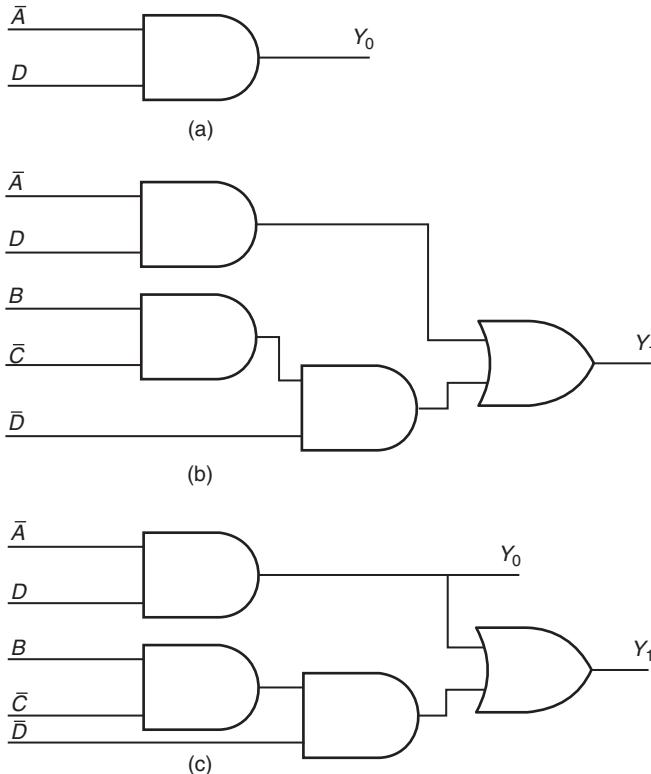


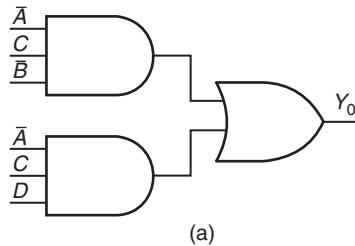
FIGURE 3.6 (a) AND-OR Array for Y_0 (b) AND-OR Array for Y_1 (c) Multi-output implementation using AND-OR sub-arrays.

respectively. The tables show a # sign marked term in each. These terms form a group with a common term in the two maps. The use of this common term in one group gives an implementation, which minimizes with lesser number of gates than when implementing by common diads or quads. The Boolean expressions after Karnaugh minimization are $Y_0 = \bar{A} \cdot B \cdot C + A \cdot C \cdot D$ and $Y_1 = \bar{A} \cdot B \cdot C + A \cdot C \cdot \bar{D}$. $Y_0 = \Sigma m(2, 3, 7)$ $Y_1 = \Sigma m(2, 6, 7)$ [from Table 3.9].

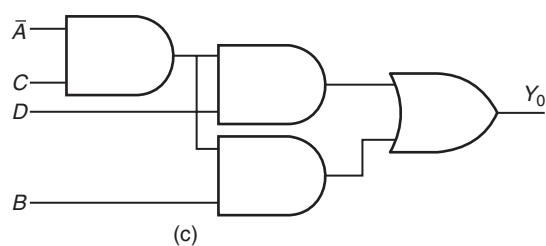
Since no diad (pair of two adjacent cells) is common in expression for Y_0 and Y_1 in both, total four ANDs of three inputs (Figures 3.7(a) and (b)) and six two-input NANDs will be needed in Figures 3.7(c) and (d). A simplified low cost circuit is as per Figure 3.7(e). Number of gates reduces by one in the common implementation. We need five two-input ANDs or alternatively one four input and two three input gates (Figure 3.7(f)).

3.8.1 Prime Implicants for Multi-Outputs Case

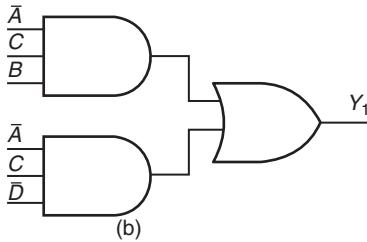
Prime implicant of a Boolean function is a product term for the function, which has no other term with lesser literals to represent the function. A prime implicants for multi-outputs case is a prime implicant of either one of the individual function



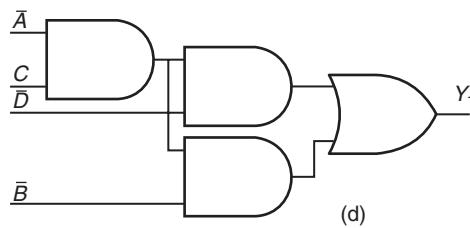
(a)



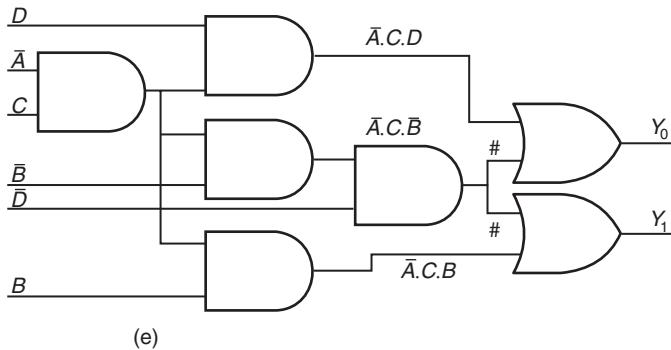
(c)



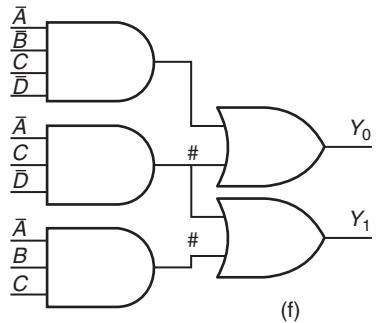
(b)



(d)



(e)



(f)

FIGURE 3.7 (a) and (b) 3-input AND-OR Array for Y_0 , Y_1 (c) and (d) Two input AND-OR Array for Y_0 , Y_1 (c) and (d) Multi-output implementation for Y_1 (e) A joint implement (f) simplification of AND-OR arrange in joint implementation.

TABLE 3.25 Karnaugh Map for $Y_0 = \bar{A}\bar{B}C + \bar{A}C\bar{D}$

$\bar{A}\bar{B}$	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$		
$\bar{A}\bar{B}$	00			<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>1</td></tr> <tr><td>1#</td></tr> </table>	1	1#
1						
1#						
$\bar{A}B$	01		<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>1</td></tr> <tr><td></td></tr> </table>	1		
1						
AB	11					
$A\bar{B}$	10					

TABLE 3.26 Karnaugh Map for $Y_1 = \bar{A}B\bar{C} + \bar{A}C\bar{D}$

AB	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$		
$\bar{A}\bar{B}$	00					
$\bar{A}B$	01			<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>1</td></tr> <tr><td>1</td></tr> </table>	1	1
1						
1						
AB	11					
$A\bar{B}$	10					

or of their products. A prime implicant for multi-outputs Y_0 and Y_1 case is a prime implicant of either Y_0 or Y_1 or $Y_0 \cdot Y_1$. A prime implicants for multi-outputs Y_0 , Y_1 and Y_2 case is a prime implicant of any of the seven functions either (a) Y_0 or Y_1 or Y_2 or (b) $Y_0 \cdot Y_1$ or $Y_1 \cdot Y_2$ or $Y_2 \cdot Y_0$ or (c) $Y_0 \cdot Y_1 \cdot Y_2$.

Cost minimization has two approaches either by reducing the number of gates or the number of inputs to the gates. Cost minimization is for a sum of the set of prime-implicants such that all the prime implicants of either that function or its product are present. That means that the cost minimization for Y_i is either done using the prime-implicants expression for Y_i or for a product function of Y_i . Reason for this is when considering Y_i the other function values do not matter. They correspond to don't care condition and don't care condition can be taken as 1 in the product.

3.9 TWO OUTPUTS SIMPLIFICATION—COMPUTER-BASED PRIME IMPLICANTS USING STAR PRODUCT AND SHARP OPERATIONS

3.9.1 Combination of Two Cubes Differing in One Variable into One Cube—A Star Product Operation

Recall the pairing operation described in sections 3.7.1 to 3.7.3 Consider a pair in which one coordinate only change and are 0 and 1 in two terms. We reduce it to one term with coordinate = x . For example, we convert the pair (1, ..., 0, 0, 1) and (1, ..., 1, 0, 1) into (1, ..., x , 0, 1).

Since conversion to a single x basically means a removal of one of the variable or reducing the dimension of the cube. Therefore, two cubes can also be combined together into one cube. Let us assume that there are two m -dimensional cubes, U and V with m -variable each. Let the variables are u_1, u_2, \dots, u_m on first hyper cube and v_1, v_2, \dots, v_m on second hyper cube. A variable, u or v is either 1 or x or 0. x means a don't care condition.

Now star-product operation U^*V is as follows:

For each pair of u_i and v_i

- (i) $u_i v_i = 0$ if both are 0.
- (ii) $u_i v_i = 0$ if $u = x$ and $v = 0$.
- (iii) $u_i v_i = 0$ if $u = 0$ and $v = x$.
- (iv) $u_i v_i = 1$ if both are 1.
- (v) $u_i v_i = 1$ if $u = x$ and $v = 1$.
- (vi) $u_i v_i = 1$ if $u = 1$ and $v = x$.
- (vii) $u_i v_i = \epsilon$ if $u = 1$ and $v = 0$.
- (viii) $u_i v_i = \epsilon$ if $u = 0$ and $v = 1$.
- (ix) $u_i v_i = x$ if $u = x$ and $v = x$.

The $i = 1, 2, \dots, m$. Star product operation means a cube W with coordinates W_i . W is obtained by a star-product operation $U * V$ using expressions (i) to (ix) for each value of i .

- (1) $W = \epsilon$, if $u_i v_i = \epsilon$ for greater than one value of i between 1 and m .
- (2) $W_i = u_i v_i$ if $u_i v_i = 0$ or 1 or x , where W_i is i^{th} coordinate of W .
- (3) $W_i = x_i$ if $u_i v_i = \epsilon$.

Points to Remember

1. If $W = \epsilon$, it means that U and V can't be combined into W .
2. Two cubes can be combined into one cube if they differ in one (not none or more) of the variable i between $i = 1$ to m .
3. W is discarded for minimization if they differ but the W is included in both U and V . For example, $U_i = \{x11\}$ and $V_i = \{1, 1, x\}$. Now $W_i = \{111\}$ from (v) and (v_i) above, which means W_i is included in U_i and V_i both.
4. W can be included for minimization if they differ but the W is not included in either U or V but not in both. For example, $U_i = \{x, 0, 1\}$ and $V_i = \{1, x, 0\}$. Now $W_i = \{1, 0, x\}$, which means a W is included in V but not included in U .

3.9.2 Finding Essential Prime Implicants Using Two Cubes—A Sharp Operation

Recall section 3.4.5. Refer inclusion of 1 at a cell in the map from a don't care condition(s) (incomplete specification case) to complete a Karnaugh map quad or pair or octet. Consider a pair of vertex on a hypercube in which one coordinate is x and other is either 0 or 1 in two vertices. We consider it as one vertex with coordinate = x replaced by either by 1 or 0 as follows. For example, convert the pair $(1, \dots, x, 0, 1)$ and $(1, \dots, 1, 0, 1)$ into $(1, \dots, 0, 0, 1)$ or convert the pair $(1, \dots, x, 0, 1)$ and $(1, \dots, 0, 0, 1)$ into $(1, \dots, 1, 0, 1)$.

Since conversion to a single x basically means a removal of one of the variable or reducing the dimension of the cube.

Let us assume that there are two m -dimensional cubes, U and V with m - variables. Let the variables are u_1, u_2, \dots, u_m on first cube and v_1, v_2, \dots, v_m on second cube. A variable, u or v is either 1 or x or 0.

Now sharp operation $U \# V$ means the following:

For each pair of u_i and v_i ,

- (i) $u_i \# v_i = \zeta$ if both are 0.
- (ii) $u_i \# v_i = 1$ if $u = x$ and $v = 0$.
- (iii) $u_i \# v_i = \zeta$ if $u = 0$ and $v = x$.
- (iv) $u_i \# v_i = \zeta$ if both are 1.
- (v) $u_i \# v_i = 0$ if $u = x$ and $v = 1$.
- (vi) $u_i \# v_i = \zeta$ if $u = 1$ and $v = x$.
- (vii) $u_i \# v_i = \epsilon$ if $u = 1$ and $v = 0$

(vii) $u_i \# v_i = \epsilon$ if $u = 0$ and $v = 1$.

(ix) $u_i \# v_i = \zeta$ if $u = x$ and $v = x$.

The $i = 1, 2, \dots, m$. Sharp operation means a cube W obtained by sharp operation $U \# V$ defined as follows:

- (1) $W = U$, if $u_i \# v_i = \epsilon$ for some value of i between 1 and m (For some i , the variable is complementary in U and V); and defined by (vii) and (viii) relationships.
- (2) $W = \epsilon$ if $u_i \# v = \zeta$ for all values of i .
- (3) $W = U(u_1, u_2, \dots, u_i, \dots, u_m)$ for cases in which $u = x$, and $v_i =$ either 0 or 1 (U is a special sign for the union. Union is for all value of i where these conditions for u and v exist).

Note the important points as following:

1. $W = U$, it means that U and V differs at least in one variable and that variable(s) is complement of each other in U and V .
2. $W = \epsilon$ means that cube U fully covers V . For all i , both u and v either 0s, or 0 and x , or x and 1, or 1 and x , or x and x .
3. $W = U$ is a new union where that part, which was not covered in U , is included now. When $u_i = x$ and is 0, and $v_i = 1$, the u did not include v_i . For a value of i , when $u_i = x$ and is 1, and $v_i = 0$, the u_i did not include v_i . It is now included in the union.
4. By using the sharp operation, all essential prime implicants can be grouped. This will result in minimum possible cover (implementation by AND-OR circuits.)

3.9.3 Computer-Based Minimization Method to Find Minimum Required Cover (SOP function implicants)

Following are the steps for a computer-based minimization to get the minimum required cover (SOP function essential prime implicants).

1. Step 1: Specify the coordinates of marked vertices, which correspond to the given set of SOP minterms (implements). Let us call it as SET A .
2. Step 2: Specify the coordinates of marked vertices, which correspond to the given set of SOP minterms (implements) after including the don't care conditions (unspecified table rows or unspecified input conditions). Let us call it as SET A' .
3. Step 3: Find prime implicants using the Star-product operation in the SET A' implicants.
4. Step 4: Find union after the Sharp operation to find essential prime implicants and thus the minimum cover for A' .
5. Step 5: Write the least cost (minimized) SOP expression.
6. Step 6: Make the AND-OR logic circuit for the terms.

3.10 COMPUTER-BASED MINIMIZATION— QUINE-McCLUSKEY METHOD

It has been observed in sections 3.1 to 3.6 that a Karnaugh map can be used to get the minimization by forming diads, quads and octets that remove the one, two and three, variables respectively. The basis for the removal is the Boolean OR operation $X \cdot \bar{Y} + X \cdot Y = X$, where Y is a single variable and X is a single variable or a multivariable product term. Therefore, two product terms $\bar{A} \cdot C \cdot D \cdot \bar{B} + \bar{A} \cdot C \cdot D \cdot B$ will reduce to $\bar{A} \cdot C \cdot D$ because they differ in one variable value B .

$X = A \cdot C \cdot D$ and $Y = B$ in above example). In Karnaugh map two adjacent cell pairs differ in one variable value. That is why a variable reduces on pairing adjacent or wrapping-adjacency showing 1s in the SOP terms of Karnaugh map.

There are two problems in Karnaugh map approach. First is how to work with the map in case of increase in the number of variables visualization becomes more and more cumbersome. Second difficulty is adoptability to a computer based minimization. To take care of these two problems, one procedure of using cube and hyper cube vertices coordinates for the minterms was described in Section 3.9. A popular procedure is the procedure known as Quine-McCluskey method. It also provides the answer to above two problems.

3.10.1 Quine-McCluskey Method of Finding Prime Implicants

Step 1: Write the expression for a Boolean function Y in terms of the product term. $Y = \sum m(1, 3, 5, 6, 7, 10, 14, 15) = \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot D + \bar{A} \cdot \bar{B} \cdot C \cdot D + \bar{A} \cdot B \cdot \bar{C} \cdot D = \bar{A} \cdot B \cdot C \cdot \bar{D} + \bar{A} \cdot B \cdot C \cdot D + A \cdot \bar{B} \cdot C \cdot \bar{D} + A \cdot B \cdot C \cdot \bar{D} + A \cdot B \cdot C \cdot D$ is an exemplary expression undertaken here for explaining the steps. Let an index represent the number of 1s in a SOP term for Y . List all the terms with index = 0 in one list, index = 1 in next list, index = 2 in next list, index = 4 in the next list and so on. For the four variable case, the maximum value of index = 4 and the indices are 0, 1, 2, 3 and 4. Separate each list by line at the end of the list. Columns 1 to 4 of Table 3.27 shows the five lists for the example under description.

There are four list-end lines in the four cells to show five separate lists. There is no term in the list with index 0. (Note: Logic of this step can be understood as follows. Basically a list having additional 1 compared to previous list is the one having the potential to have the pairing terms within the lists for the terms which can be paired to satisfy the expression $X \cdot \bar{Y} + X \cdot Y = X$).

Step 2a: Start from index 0 list and find a pair of minterm in the index 1 list in cycle 1. Continue to find pair upto lists 0 and 5 pairing. Index 0 list is empty. Therefore, no pairs. Switch to step 2b with no action.

Step 2b: Start cycl3 2 from index 1 and find a pair of minterm in the index 1 list and next list with index = 2. (The pair is the one, satisfies that $X \cdot \bar{Y} + X \cdot Y = X$). Indices 1 and 2 lists are not empty. Therefore, the action is to be for finding the pairs after comparisons. In column-5 two lists' comparison is done (between pair of minterms 1 and 3, 1 and 5, 1 and 6, and 1 and 10). Pairs 1 and 3 and 1 and 5 differ in one variable each for third and second places (taking from left), respectively. (Recall Table 3.9 Karnaugh map also, the minterms $m(1)$ and $m(3)$, and $m(1)$ and $m(5)$ are

the adjacent cell pairs.) Write up a list of SOP pairs in next column (column 6). Refer entries 00–1 and 0–01. (0001 + 0011) of column 3 gives 00–1 and (0001 + 0101) gives 0–01 in the second column. The dash sign means removed variable after using the OR relation. Successful pairs are check marked. Put check marks in column 2 sign at 1, 3 and 5 in Table 3.27. Draw a line below the list 1 in columns 5 and 8.

Step 2c: Start from index 2 and find a pair of minterms in the index 2 list and next list with index = 3. Indices 2 and 3 lists are not empty. Therefore, the action is to be for finding the pairs after comparisons. In list 2 column-5 the two lists' comparison is given after OR operations between pair of minterms (3, 7), (3, 14), (5, 7), (5, 14), (6, 7), (6, 14), (10, 7) and (10, 14). Pairs 7a(3, 7), 7b(5, 7), 7c(6, 7), 14a(6, 14), 14b(10, 14) are satisfying $X \cdot Y + X \cdot Y = X$ expression. Recall Table 3.9 Karnaugh map also, the minterms $m(6)$ and (14) , and (10) and (14) are the adjacent cell pairs, and so on. We can verify adjacency from the Table 3.9 for all these five pairs at the two lists. Put a check mark on 3, 5, 6, 10, 7 and 14 and carry remaining result of sum of product terms in the pairs to next column (column 6) second list. Successful pairs are check marked. Refer entries 0–11 and 01–1. (0011 + 0111) gives 0–11 and (0101 + 0111) gives 01–1. The dash sign means removed variable after using the OR relation described above. Draw a line below new list 2 at the last cell in column 5 to 8.

Step 2d: Start from index 3 and find a pair of minterms in the index 3 list and next list with index = 4. Indices 3 and 4 lists are not empty. Therefore, the action is to be for finding the pairs after comparisons. In column-5 two lists' comparison is given between pair of minterms (7, 15) and (14, 15). Both pairs are satisfying $X \cdot Y + X \cdot Y = X$ expression. Recall Table 3.9 Karnaugh map also, the minterms $m(14)$ and (15) , and (7) and (15) are the adjacent cell pairs. Write the result of sum of product terms in the pairs in next column (column 6). In the present step, the column 6 list 3 entries

TABLE 3.27 Finding the prime implicants using the Quine-McCluskey method

List	Minterms	Binary form and Index		Comparison cycle 2		New index and list no.		Comparison cycle 3		Comparison cycle 4
1	~ —	~ ~	0 0	3 ^a 5 ^a	00–1 0–01	1	1	7d	0—1	Not required
2	1 ^c	0001	1	7 ^a	0–11			15 ^c	—11—	
3	3%	0011	2	7 ^a	01–1	2	2			
	5%	0101	2	7 ^c	011—					
	6%	0110	2	14 ^a	—110					
	10%	1010	2	14 ^b	1—10					
4	7 [^] 14 [^]	0111 1110	3 3	15 ^a 15 ^b	—111 111—	3	2			
5	15 ^{\backslash}	1111	4							

(a) ~ means index 0 correspond to no minterm. (b) ^ means index = 1 because only at one place there is 1. (c) % means index = 2 because only at two places there are 1s. (d) ^ means index = 3 because only at three places there are 1s. (e) \ means index = 4 because only at four places there are 1s. (f) | Dash sign here means variable C removed from the minterm on simplification between $m(1)$ and $m(3)$. The term now left is $\bar{A} \cdot \bar{B} \cdot D$, represented by 00–1. (g) 3^a and 5^a means 3 and 1 and 5 and 1 paired in step 2b. Cycle 1 does not exists as no pair of list 0 with others.

are -111 and $111-$. $(0111 + 1111)$ gives -111 and $(1110 + 1111)$ gives $111-$ (15a and 15b, respectively). The dash sign means a removed variable after using the OR relation described above. Draw a line below new list at the last cells columns 5 to 8.

Carry Step further for the five variable cases, Step 2f for six variable case and so on. Since the example is for four variables case, in step 2 steps 2a, 2b, and 2c suffice. Move to next Step 3. Let us number the three lists (column 8). Marking ‘a’ or ‘b’ or ‘c’ signifies that these are the terms of the lists at the second cycle of the actions at the Steps 2a to 2d. A number specified on the left of the dashed minterms sorted out in the Step 2s corresponds to the minterm number presuming – sign as 1. For example, $00-1$ is marked as 3a. If next time at cycle 3 also if $00-1$ occurs then it is marked as 3b. Labeling of minterms is done here for better understanding of each of the steps described here.

Operations in steps 2a, 2b, 2c, ... are called cycle 2 operations.

General Instructions for Step 3 Cycle 3

1. Compare all pairs between (i) first and second lists of column 6, (i) second and third and so on. In present example, the three lists are present at column 6. Therefore, only Steps 3a and 3b are required. Compare the pairs in column 6 but in between two adjacent lists only.
2. Only those pairs in which the dash(es) are occurring at the same places needs to be considered. For example, pair 3^a and 7^a need not to be considered as dashes are at the third and second places from most bit places, in $00-1$ and $0-01$, respectively.

Step 3a: Start from index 1 as index 0 term is not present in column 6. Find a pair of minterms in the corresponding list and next list with index = 2. (Index is now counted after ignoring the dash(es)). Pairs should also have the dashes at the same place. Indices 1 and 2 lists in the column are not empty. Therefore, the action is to be for finding the pairs after comparisons. In column-9 two lists’ comparison is given between the pair of minterms (3a, 7b) and (5a, 7a). Pair (3a, 7b) only is satisfying $X.Y + X.Y = X$ expression. Put the check marks on 3a, 5a, 7b and 7a terms. Carry remaining result of sum of product terms in the pairs to next cycle first list. We find that both pairs give same term $0--1$. Refer third cycle entry $0--1$. Entries $0--1$ is because $(00-1 + 01-1)$ gives $0--1$ and $(0-01 + 0-11)$ gives $0--1$. Two dash signs means two removed variables after using the OR relation described above in cycles 2 and 3. Draw a line below new list at the last cell in columns 9 and 10.

Step 3b: Start from index 2. Find a pair of minterms in the new index 2 list and next list with index = 3 (index is now counted after ignoring the dash(es)). The pair is the one, which satisfies the $X.\bar{Y} + X.Y = X$. Pairs should also have the dashes at the same place. Indices 2 and 3 lists 2 and 3 are not empty. Therefore, the action is to be for finding the pairs after the comparisons. In column-9 two lists’ comparison is done between pair of minterms (7c, 15b) and (14a, 15a), which only are satisfying $X.\bar{Y} + X.Y = X$ expression. Put the check marks on 7c, 15b, 14a and 15a terms. Carry remaining result of sum of product terms in the pairs to next cycle list. We find that both pairs give the same term $-11-$. Refer third cycle entry $-11-$. Entries $-11-$ is because $(011- + 111-)$ gives $-11-$ and $(-110 + -111)$ gives $-11-$.

Two dash signs means two removed variables after using the OR relation described above, operated in cycle 1 and cycle 2. Draw a line below new list at the last cell in columns 9 and 10.

Out of ninth entries in column 6, the eighth are marked (paired in cycle 3 for generating column 10). Continuation of the further cycle 4 steps are not required as both the lists in column 10 have one entry each and the pair between cannot be formed because the dashes are at the different places (0 – – 1, – 11 –).

Collect all the unmarked entries from (unpaired entry) columns 3, 6 and 10. Column 6 has left one uncheck marked entry $m(10)[1-10]$ and column 10 has two entries $m(1)$ and $m(6)$ [–11 – and 0 – – 1] unpaired. An unchecked entry corresponds to a prime implicant.

Three prime implicants are therefore as follows:

$$Y = \Sigma m (1, 3, 5, 6, 7, 10, 14, 15) = \Sigma m (1, 6, 10) \bar{A} \bar{C} D = \bar{A} D + B C$$

The above process looks tedious, but easily implements using a computer program. For multiple variables, this will be the best option.

3.10.2 Finding Minimal Sum from the Prime Implicants for an Output

Exemplary prime implicants table was shown in Table 3.28. We assume don't care variable as 1. Minimal sum is sum of prime implicants, which are absolutely necessary, else the Boolean function itself modifies.

First check whether each row has at least one term present and check that whether removal of a row still leaves label y in at least one column. If removal of a row is possible then remove provided a row still leaves label y in at least one column (for example, assume there is y at $m7$ also. Then middle row can be removed because $m6 + m7 = m6$ and it removes the redundant term).

We therefore get on summing all the prime implicants at the table the minimal sum expression. In the following expression, there is no redundancies. Hence, the answer is follows:

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

It is also called *irredundant disjoint* expression. None of the row of Table 3.26 can be removed; otherwise the function will be modified.

TABLE 3.28 Prime implicants table from the Quine-McCluskey method for finding redundancy and then obtaining minimal sum expression (irredundant disjoint expression)

	m_0	m_1	m_2-m_5	m_6	m_7	m_8	m_9	m_{10}	m_{14}	m_{15}
BC				y						
$\bar{A} D$		y								
$AC\bar{D}$								y		

y shows the prime implicants (1–10, 0––1, – 11 –) in the Table 3.27 example.

3.10.3 Finding Minimal Sum for the Multi-Output Case Using Quine-McCluskey Method

Quine-McCluskey method can be applied to multiple outputs case as follows. We have to find three sets of prime implicant table for two-outputs case, for Y_0 , Y_1 and $Y_0 Y_1$.

Let Y_0, Y_1, \dots, Y_{n-1} are the outputs for a set of input variable X_0, X_1, \dots, X_{m-1} . A truth table of these will have m columns for the inputs and n columns for the outputs. Number of rows will be number of possible input combinations 2^m . Now suppose $n = 2$ and $m = 3$. Therefore, the Boolean functions are Y_0 and Y_1 , which depends on X_0, X_1 and X_2 only.

Let us consider a *labeled product* term. Let us understand a labeled product term by an example. Suppose in the truth table, for $X_0 X_1 X_2 = 001$, the $Y_0 = 1$ and $Y_1 = 0$, then labeled product term is $001 Y_0 -$. It means that Y_0 term exists for 001 input and Y_1 does not exist for 001 input. For $X_0 X_1 X_2 = 000$, if $Y_0 = 1$ and $Y_1 = 1$, then labeled product term is $000 Y_0 Y_1$. For $X_0 X_1 X_2 = 010$, if $Y_0 = 0$ and $Y_1 = 1$, then labeled product term is $001 - Y_1$.

Now find the prime implicants using Quine-McCluskey method for the five-variable case in place of four input variables A, B, C and D case considered in section 3.10.1 (Note: a general case will be $(m + n)$ variable case).

After finding prime implicants, make three prime implicants table in place of one table. One table is for the variable $X_0 X_1 X_2 Y_0 -$, other for $X_0 X_1 X_2 - Y_1$ and another for $X_0 X_1 X_2 Y_0 Y_1$. Prime implicants in third table will correspond to the common prime implicants so that the terms can be used in Y_0 and Y_1 both. Prime implicants in first table will correspond to the terms that can be used in Y_0 . Prime implicants in second table will correspond to the Y_1 .

■ EXAMPLES

Example 3.1

From the given three inputs, A, B , and C truth table (Table 3.29), construct a Karnaugh map and then construct the SOP functions based on map for output S . Simplify the result and find a Boolean expression.

TABLE 3.29 Truth table

A	B	C	Output S
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Solution

The 0th, 2nd, 4th and 6th row of truth table have the output = 1. We put the 1s at the 000, 010, 100 and 110 cells in the Karnaugh map (Table 3.1). (Four minterms (implicants) are the ones for which $S = 1$). We can use Table 3.3 to get the SOP—

$$S = \Sigma m(0, 2, 4, 6)$$

Table 3.30 shows the map.

TABLE 3.30 Map from truth table and corresponding minterms

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

AB	C	0	
00		$m(0)$	
01		$m(2)$	
11		$m(6)$	
10		$m(4)$	

There is a quad containing four adjacent columns in first column. Hence two variables can be removed out of three. Only \bar{C} is left and

$$S = \bar{C}.$$

Example 3.2

From the given four inputs, A , B , C and D truth table (Table 3.31) construct a Karnaugh map, construct the SOP functions based on the Karnaugh map for an SOP output S . There is no specifications given for $A = 1$, $B = 0$, $C = 1$, and $D = 1$.

TABLE 3.31 Truth table

A	B	C	D	S
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Solution

If we count from 0th row, the 3rd, 5th, 7th, 9th and 13th row of truth table have the output = 1. Now use the map in Table 3.7 as template to fill 1s from the truth table. We put the 1s at the 0011, 0101, 0111, 1001 and 1101 cells in the Karnaugh map (Table 3.32). Eleventh row is not present and we assume it as don't care condition cell. We put the x in the 1011 cell. Five minterms (implicants) are the ones for which $S = 1$. Using Table 3.9, we find 1s at $m_3, m_5, m_7, m_9, m_{13}$.

$$S = \Sigma m(3, 5, 7, 9, 13)$$

TABLE 3.32 Four variable Karnaugh Map and adjacencies

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

Wrapping Adjacency

We have two pairs of the adjacent cells (m_5, m_7) and (m_9, m_{13}). One additional-pair exists assuming x (don't care) as 1 at last row. This additional-pair exists on considering a wrapping adjacency. We can remove one variable each. Hence answer is as follows:

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

Example 3.3 From the given Karnaugh map in Table 3.33, find the minterms present at the SOP.

TABLE 3.33 Three variable Karnaugh Map and minterms

$AB \backslash C$	\bar{C} 0	\bar{C} 1	$AB \backslash C$	0	1
00		1	00	—	$m(1)$
01		1	01	—	$m(3)$
11	1		11	$m(6)$	—
10			10	—	—

Solution

There are 1s at the cells, 001, 011, and 110. Hence 1st, 3rd and 6th minterm is present (using Table 3.3). Hence $m(1), m(3)$ and $m(6)$ are present in the SOP—

$$S = \Sigma m(1, 3, 6)$$

Example 3.4

From the given maxterms in Table 3.34, find the Karnaugh map. Conditions for $A = 1$ are not specified and are to be taken as don't care. Find the POS expression also.

TABLE 3.34 Maxterms and POS output function

A	B	C	D	Maxterm	POS Output function
0	0	0	0	$Mx0 \ A + B + C + D$	1
0	0	0	1	$Mx1 \ A + D + C + \bar{D}$	1
0	0	1	0	$Mx2 \ A + B + \bar{C} + D$	0
0	0	1	1	$Mx3 \ A + B + \bar{C} + \bar{D}$	1
0	1	0	0	$Mx4 \ A + \bar{B} + C + D$	0
0	1	0	1	$Mx5 \ A + \bar{B} + C + \bar{D}$	1
0	1	1	0	$Mx5 \ A + \bar{B} + \bar{C} + D$	0
0	1	1	1	$Mx7 \ A + \bar{B} + \bar{C} + \bar{D}$	1

Solution

There are 0s at $M(2)$, $M(4)$ and $M(6)$. Hence, we put the 0s in the cells at 0010, 0100, 0110 as alone in filling Table 3.12 (Section 3.2.3). We also put x for the cell positions with $A = 1$. Therefore, the map is as per Table 3.35. We have

$\bar{P} = \prod M(2, 4, 6)$ (from the Table 3.11) if we do not consider don't care conditions. Else $\bar{P} = \prod M(2, 4, 6, 8, 9, 10, 11, 12, 13, 14, 15)$ by assuming 0s at the places of x .

TABLE 3.35 Four variables Karnaugh Map for maximum 16 Maxterms in a POS expression— $(A + B + \bar{C} + D).(A + \bar{B} + \bar{C} + D).(A + \bar{B} + C + D)$ when not considering don't care terms

$A + B$	$C + D$	$C + D$ 00	$C + \bar{D}$ 01	$\bar{C} + \bar{D}$ 11	$\bar{C} + D$ 10
$A + B$	00				0
$A + \bar{B}$	01	0			0
$\bar{A} + \bar{B}$	11	x	x	x	x
$\bar{A} + B$	10	x	x	x	x

Note: $\bar{P} = \bar{A}.(A + \bar{C} + D). (A + \bar{B} + D)$ on placing 0s for x .

Example 3.5

Show the octets, quads and pair of adjacent cells in Table 3.36 below and hence simplify the given implicants (minterms) for function $S=1$ and obtain prime implicants.

TABLE 3.36 Use of don't care input combinations for determining the adjacencies

<i>AB</i>	<i>CD</i>	00	01	11	10
-	00	1	1	1	1
-	01	x		x	1
-	11	1			1
-	10	1	x	1	1

<i>AB</i>	<i>CD</i>	<i>C + D</i> 00	<i>C + D̄</i> 01	<i>C̄ + D̄</i> 11	<i>C̄ + D</i> 10
-	00	1	1	1	1
-	01	x		x	1
-	11	1			1
-	10	1	x	1	1

Solution

The map shows two octets consisting of (1st and 4th rows) and (1st and 4th columns) considering wrapping adjacencies and taking don't care conditions as 1. The map also shows a quad in first and in second rows.

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

But the second term $\bar{A} \cdot \bar{C}$ is not essential, because x can as well be taken as 0. Therefore, the essential, (prime) implicant is only one term in S , and

$$S = B$$
 is the least cost AND-OR circuit.

One AND has all inputs = B .

Example 3.6 Find Karnaugh map of $X + \bar{Y} \cdot Z$ in POS standard format (POS cover) and verify the answer by minimizing the map.

Solution

Recall Boolean rule; $X + \bar{Y} \cdot Z = (X + \bar{Y}) \cdot (X + Z)$ (Equation 2.15b). We have to first convert it into standard POS format.

This can be written as $P = (X + \bar{Y} + Z \cdot \bar{Z}) \cdot (X + Y \cdot \bar{Y} + Z)$. Using ANDing rule that AND with the complement is 0 and ORing rule that OR with 0 has no effect in a Boolean operation.

$$\bar{P} = (X + \bar{Y} + \bar{Z}) \cdot (X + \bar{Y} + Z) \cdot (X + Y + Z) \cdot (X + \bar{Y} + Z)$$

(After expanding by using the rule $X + Y \cdot Z = (X + Y) \cdot (X + Z)$ for both the ANDing terms). Using ANDing rule that $X \cdot X = X$ for the second and fourth term in P :

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

The POS form therefore $\Pi M(0, 3, 2)$ (First term in $M(3)$, second term is $M(2)$ and third is $M(0)$). We have to put 0s in cells 000, 011 and 010 for forming the POS Karnaugh map.

Now recall Table 3.5 and put 0s at appropriate Maxterm positions and we get map as shown in Table 3.37.

TABLE 3.37 Three variable Karnaugh Map

$XY \backslash Z$	Z	$Z = 0$	$Z = 1$
$X + Y$	00	0	
$X + \bar{Y}$	01	0	0

Table 3.37 is the desired map. We can verify that the map is correct as follows. There are two adjacent cell pairs. From pair in first column, Y is removable. So $(X + Z)$, which is common is left. From pair in second row, Z is removable. So $(X + \bar{Y})$, which is common is left. Using distribution rule $X + Y.Z = (X + Y).(X + Z)$, we get the simplest circuit $X + \bar{Y}.Z$. This is from where we started.

Example 3.7

Give Karnaugh map of $X + Y.Z$ in a SOP cover (standard format) and verify the answer by minimizing the map.

Solution

Recall ORing rule; $Y + \bar{Y} = 1$. (Equation 2.10d) We have the expression $X + Y.Z$. We first convert it into standard SOP format using this rule. Therefore, SOP S is as follows:

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

(Remove fourth term $X.Y.Z$ occurring twice using rule $X + X = X$).

$= \Sigma m(3, 4, 5, 6, 7)$. (Above expression first term is $m(4)$, second is $m(6)$, third $m(5)$, fourth $m(3)$ and fifth $m(7)$).

We have to put 1s in cells 011, 100, 101, 110 and 111 for forming the SOP Karnaugh map.

Now recall Table 3.5 and put 1s at appropriate minterm positions (cell positions 011, 100, 101, 110 and 111) and get map as shown in Table 3.36.

Table 3.38 is desired the Map. We can verify that map is correct as follows. There are two adjacencies; one quad of 4 cells and one pair of cells. From quad in last two rows, Y and Z are removable. So X , which is common is left. From pair in 2nd column, X is removable. So $(Y.Z)$, which is common is left. We get simplest circuit as $X + Y.Z$. This is from where we started.

TABLE 3.38 Three variable Karnaugh Map

$XY \backslash Z$	Z	\bar{Z}	Z
$XY \backslash \bar{Z}$	0	1	
00			
01		1	1
11	1	1	
10	1	1	

Example 3.8

From the given Karnaugh map in Table 3.39, find the simplified expression in terms of XOR and XNOR logic circuits.

TABLE 3.39 Diagonal and offset adjacencies Karnaugh Map

$AB \backslash CD$	CD	$\bar{C}D$	$\bar{C}\bar{D}$	CD	$\bar{C}\bar{D}$
$AB \backslash \bar{C}D$	00				
-	00				
XOR	01	1			
-	11		1	1	1
-	10	1			

Solution

We find that between first row last two columns and third row last two columns, there are 4-cell with offset of one row. Therefore, we can write the simplified term as XNOR gates. $(A \text{ XNOR } B)C.D + (A \cdot \text{XNOR } C.D)$. This is equal to $(A \text{ XNOR } B)C$. (From ORing rule $D + \bar{D} = 1$).

We also find that between 2nd and 3rd rows first two columns, there is a 2-cell pair having diagonal adjacent. $B \cdot \bar{C}$ is common. We can write remaining simplified term as XOR gates, $(A \text{ XOR } D)B \cdot \bar{C}$.

We also find that between 3rd and 4th rows first two columns, there is another 2-cell with diagonal adjacent. $A \cdot \bar{C}$ is common. Therefore, we can write the simplified term as XOR gates. $(B \text{ XOR } D)A \cdot \bar{C}$.

$$\text{Therefore, } S = (A \text{ XNOR } B).C + (A \text{ XOR } D)B \cdot \bar{C} + (B \text{ XOR } D)A \cdot \bar{C}.$$

Example 3.9

Represent $S = \bar{A}.\bar{B}.C + \bar{A}.B.C + A.\bar{B}.C$ in a cube form.

Solution

To write in terms of minterms, we put A , B and C as 1. From $\bar{1}\bar{1}1$, $\bar{1}11$ and $1\bar{1}1$ in the expression and evaluate the binary value in decimal. We find $S = \Sigma m(1, 3, 5)$ and the cube coordinates are (001, 011, 101) because $\bar{1} = 0$.

Answer is a cube with vertices at the coordinates (0, 0, 1), (0, 1, 1) and (1, 0, 1) marked. Also we show an axis joining (0, 0, 1), (0, 1, 1) as dark with mid point labeled as (0 x 1). This is because only Y middle coordinate is differing.

Also show another axis joining (0, 0, 1), (1, 0, 1) as dark with mid point labeled as (x, 0, 1).

Now two axis with mid points are (0 x 1) and (x, 0, 1). This will give simplified logic circuit as $(\bar{A} + \bar{B}).C$ from [$\bar{A}C$ is term from (0 x 1) and $\bar{B}C$ from (x 0 1)].

Example 3.10

Verify the simplification of $S = \bar{A}.\bar{B}.C + \bar{A}.B.C + A.\bar{B}.C$ as $(\bar{A} + \bar{B}).C$ from cube form in Example 3.9 by using Karnaugh map approach.

Solution

To write in terms of minterms, we put A , B and C as 1. From 111, 111, 111 and 111 in the expression and evaluate the binary value in decimal. $S = \Sigma m(1, 3, 5)$. Karnaugh map cells, which have 1s, are 001, 011 010. We get the map as per Table 3.40 using the Table 3.3.

TABLE 3.40

$AB \backslash C$	0	1	
\bar{C}	0	1	
00		1	Adjacency
01		1	
11			Wrapping Adjacency
10		1	

We have two pair of adjacent cells (one is with wrapping adjacency). From each pair we remove one variable and take the common two variables each.

$$\text{We get } S = \bar{A}.C + B.C = (\bar{A} + B).C$$

We get the same answer as by the cube form representation in Example 3.9.

EXERCISES

- From the given three inputs, A , B , and C truth table in Table 3.41, construct the SOP functions based Karnaugh map for an output S .

TABLE 3.41 Three variable truth table

A	B	C	Output S
0	0	0	x
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

2. From the given four inputs, A, B, C and D truth table in Table 3.42, construct the SOP functions based Karnaugh map for output S (i) for SOP (ii) POS standard forms.

TABLE 3.42 Four variable truth table

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>S</i>
0	0	0	0	1
0	0	0	1	0
0	0	1	0	
0	0	1	1	1
0	1	0	0	
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	
1	0	1	0	
1	0	1	1	x
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	x

3. From the given Karnaugh map in Table 3.43, find the minterms present. Also minimize by two adjacent pairs and another adjacent pair after wrapping adjacency.

TABLE 3.43 Three variable Karnaugh Map

<i>AB</i>	<i>C</i>	\bar{C}	<i>C</i>
	0	1	
00			1
01		1	
11			1
10	1		1

4. From the given Boolean expression $A \cdot B + \bar{A} \cdot C + B + A \cdot D + \bar{A} \cdot \bar{E} + D$, find the Karnaugh maps after converting the expression in both SOP and POS standard forms.
5. From the given Boolean expression $A \cdot B \cdot D + A \cdot C \cdot D + C \cdot \bar{D} + A \cdot D + D$ find the POS and SOP Karnaugh maps.
6. From the given Boolean expression $A \cdot B \cdot C + A \cdot B \cdot \bar{C} + B \cdot \bar{C}$ find the Karnaugh maps after converting the expression in both SOP and POS standard forms.

7. Consider the following Table 3.44 for conversion of a decimal number to Gray code (a code in which next number have only one bit place changing). Assuming each code bit as a S_1 , S_2 , S_3 and S_4 , draw four three-variable Karnaugh maps and simplify as much as possible (find prime implicants).

TABLE 3.44

Decimal value	4-bit binary representation	Gray code representation
0	0000_2	0 0 0 0
1	0001_2	0 0 0 1
2	0010_2	0 0 1 1
3	0011_2	0 0 1 0
4	0100_2	0 1 1 0
5	0101_2	0 1 1 1
6	0110_2	0 1 0 1
7	0111_2	0 1 0 0
8	1000_2	1 1 0 0
9	1001_2	1 1 0 1
10	1010_2	1 1 1 1
11	1011_2	1 1 1 0
12	1100_2	1 0 1 0
13	1101_2	1 0 1 1
14	1110_2	1 0 0 1
15	1111_2	1 0 0 0

8. Consider the following Table 3.45 for conversion of a decimal number to excess-3 code (a code in which next number in binary form is three more than the binary value for the decimal value). Assuming each code bit as a S_1 , S_2 , S_3 and S_4 , draw four three-variable Karnaugh maps and simplify as much as possible (find prime implicants).

Table 3.45

Decimal value	4-bit binary representation	Excess-3 code representation
0	0000_2	0 0 1 1
1	0001_2	0 1 0 0
2	0010_2	0 1 0 1
3	0011_2	0 1 1 0
4	0100_2	0 1 1 1
5	0101_2	1 0 0 0
6	0110_2	1 0 0 1
7	0111_2	1 0 1 0
8	1000_2	1 0 1 1
9	1001_2	1 1 0 0

9. From the given minterms $S = \sum m(1, 7, 8, 9, 10)$ find the Karnaugh map-for four and five variable forms.

10. From the given maxterms $\bar{P} = \prod M(1, 2, 7, 9)$ find the Karnaugh map for four variable forms.
11. From the given minterms $S = \Sigma m(1, 6, 11, 18)$ find the Karnaugh map for five variables forms.
12. From the given maxterms $\bar{P} = \sum M(2, 7, 9, 19, 42, 47)$ find the Karnaugh map six variable forms.
13. From the given maxterms $P = \prod M(2, 7, 11, 9)$ find the Karnaugh map and draw the only NORs based circuit.
14. From the given minterms $S = \Sigma m(2, 7, 11, 9)$ find the Karnaugh map and draw the only NANDs based circuit.
15. Draw a Karnaugh map of four variable showing only wrapping adjacencies and having an octet, a quad and a pair of adjacent cells.
16. Simplify after first converting $(A \text{ XOR } B) \text{ XOR } C$ to standard SOP form and then using Karnaugh map make a circuit with the NANDs only.
17. Simplify after first converting $(A \cdot \bar{B} \cdot C) + (\bar{A} \cdot C \cdot D)$ to standard POS form and then using Karnaugh map to make a circuit with the NORs only.
18. Show the octets, quad and the pairs of adjacent cells in Karnaugh map in Table 3.46 and hence simplify the given implicants (minterms) for function $S = 1$.

TABLE 3.46 Four variable Karnaugh Map

AB		CD			
		00	01	11	10
AB	00	1	1	1	1
	01				
	11	1			1
	10	1	x	1	1

19. Give Karnaugh map of $A + B.C.D$ in POS standard format (POS cover) and verify the answer by minimizing the map.
20. Give Karnaugh map of $A + B.C.D$ in SOP standard format (SOP cover) and verify the answer by minimizing the map.
21. From the given Karnaugh map in Table 3.47, find the simplified expression in terms of XOR and XNOR logic circuits.
22. Show simplification of $S = \bar{A} \cdot B \cdot C \cdot \bar{D} + A \cdot B \cdot C \cdot \bar{D} + \bar{A} \cdot \bar{B} \cdot C$ from Karnaugh map and by a 4-dimensional cube representation approach and-draw logic circuit using AND-OR gates at first and second levels.
23. Draw $S = A \cdot B \cdot \bar{C} \cdot \bar{D} + A \cdot B \cdot C \cdot D \cdot E$ Karnaugh map and hypercube representation approach and draw logic circuit using AND-OR gates at first and second levels.
24. Simplification of $S = A \cdot B \cdot \bar{C} \cdot D + A \cdot B \cdot C \cdot D + A \cdot B \cdot \bar{C}$ from Karnaugh map and by cube representation approach and draw logic circuit using NANDs.

TABLE 3.47 A Karnaugh Map, which includes wrapping and diagonal adjacencies

$AB \backslash CD$	CD 00	CD 01	CD 11	CD 10
$\bar{A}\bar{B}$	00		1	
$\bar{A}B$	01	1		1
$A\bar{B}$	11			
AB	10	1		1

25. Simplification of $S = \bar{A}.B.C.\bar{D} + \bar{A}.B.C.\bar{D} + A.B.C$ from Karnaugh map and by cube representation approach and draw logic circuit using NORs.
26. Using Karnaugh maps for multiple- outputs given in Tables 3.48 and 3.49 draw the AND-OR arrays minimize and find the minimal expressions for multiple outputs Y_0 and Y_1 .

TABLE 3.48 Karnaugh Map for Y_0

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

TABLE 3.49 Karnaugh Map for Y_1

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

27. Using Karnaugh maps for multiple-outputs given in Tables 3.50 and 3.51 draw the AND-OR arrays minimize and find the minimal expressions for multiple outputs Y_0 and Y_1 .

TABLE 3.50 Karnaugh Map for Y_0

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

TABLE 3.51 Karnaugh Map for Y_1

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

28. Minimize using $Y = \Sigma m(2, 3, 6, 7, 10, 14, 15)$ using Quine-McCluskey method.
29. Minimize and find minimal expression for $Y = \Sigma m(2, 4, 8, 9, 10, 11)$ using Quine-McCluskey method.
30. Using minimize and find the minimal expressions for multiple outputs $Y_0 = \Sigma m(2, 3, 6, 7, 10, 14, 15)$ and $Y_1 = \Sigma m(2, 4, 6, 7, 10, 11, 14)$ using Quine-McCluskey method.
31. From a prime implicants in Table 3.52, use the Quine-McCluskey method for finding redundancy and then obtains minimal sum expression (irredundant disjoint expression).

TABLE 3.52

	m_0	m_1	m_2-m_5	m_6	m_7	m_8	m_9	m_{10}	m_{13}	m_{15}
BC				Y						
$\bar{A}D$		Y								
$AC\bar{D}$								Y		
$\bar{A}BCD$					Y					
$ABC\bar{D}$									Y	

■ QUESTIONS

1. How will you construct a three-variable Karnaugh map from a given three inputs A, B , and C truth table?
2. How will you construct a four-variable Karnaugh map from a given three inputs A, B , and C truth table?
3. Explain with two examples each, method of construction a Karnaugh map for output S (i) for SOP (ii) POS standard forms.
4. How will you find the minterms present from a Karnaugh map?
5. How will you draw logic circuit from the Karnaugh maps after converting the expression in (i) SOP and (ii) POS standard forms?
6. How will you find the wrapping adjacencies between first and last rows of a Karnaugh map?
7. How will you find the wrapping adjacencies between first and last columns of a Karnaugh map?
8. Show use of Karnaugh map for a Gray code converter.
9. How do you find the octets, quads and pairs in a Karnaugh map?
10. How do you minimize a Karnaugh map from the adjacencies?
11. How will you make a logic circuit after minimizing a SOP based Karnaugh map?

12. How will you make a logic circuit after minimizing a POS based Karnaugh map?
13. How do you from a Karnaugh map draw the only NORs based circuit?
14. How do you from a Karnaugh map draw the only NANDs based circuit?
15. How do you from a Karnaugh map draw the XNORs and XORs based circuits?
16. How do find the hypercube vertex coordinates from the five variable minterms?
17. Explain the star-product operation on the terms of a cube when minimizing using a computer program.
18. List the steps for a computer based minimizing method.
19. Describe Quine-McCluskey method to find prime implicants in five variable case.
20. Describe Quine-McCluskey method to find prime implicants in four input variables for the three multiple output expressions.

This page is intentionally left blank.

CHAPTER 4

Binary Arithmetic and Decoding and Mux Logic Units

OBJECTIVE

How do we formulate and implement the design of combinational circuits for the 4-bit and 8-bit arithmetic functions? We shall learn the answers of these questions in this chapter.

We shall learn that the binary arithmetic circuits fast addition by carry lookahead circuit; decoders and multiplexers can also be used as the bigger building blocks for the logic design.

4.1 BINARY ARITHMETIC UNITS

4.1.1 Binary Addition of Two Bits

4.1.1.1 Half Adder

Let us assume that Cy'_0 is a carry bit obtained at an output in an adder circuit, and S is a sum bit obtained in another output of the adder. Then, the circuit is called a half adder (H.A.) if we have

$$A \text{ add } B = S \text{ plus } Cy'_0, \quad \dots(4.1)$$

where the left hand side and right hand side values are as per Table 4.1. This equation is in fact representing four sub-equations, one for each row of the table.

TABLE 4.1 Truth table for half adder

Inputs left hand side		H.A. Outputs right hand side		Equation number
A	B	S	Cy'_0	
0	0	0	0	...(4.1a)
0	1	1	0	...(4.1b)
1	0	1	0	...(4.1c)
1	1	0	1	...(4.1d)

A circuit of the logic gates as shown in Figure 4.1(a) implements it. Whenever an XOR is required in a circuit the adder can be used as a building block. Examples 5.5 and 5.6 in next chapter will show use of the adder as building block.

4.1.1.2 Full Adder

When we add two digits in decimal system, we also add any previous carry. For example, let us consider a decimal addition of $59 + 07$. When lower digits $9 + 7$ are first added, we get 6 and a carry 1 to the left. We do not take into account any previous carry for the lower digits. We add just like shown in equations (4.1a to 4.1d) above for a half adder. When the upper digits 5 and 0 are added, we get 5 only. Result of the decimal addition will be 56, which is wrong. This is due to an error. We are not adding previous lower digit addition's carry while adding 5 and 0. Actually, we will get the correct answer 66 the previous stage carry also adds. This example shows that any carry from previous stage must also be added. A full adder takes care of it. Its working can be understood by truth table in Table 4.2.

Figure 4.1(b) shows the logic circuit of a full adder and also shows a shorter representation of full adder as FA . If Cy , a carry at an input of a FA , is 0 then the FA is equivalent to a half adder. Cy' is a new carry which is taken as previous (input) carry, if we use it in the next stage. FA is represented by equation

$$Cy \text{ add } A \text{ add } B = S \text{ plus } Cy', \quad \dots(4.2)$$

where the left hand side and right hand side values are as per Table 4.2. This equation is in fact representing six sub-equations, one for each row of the Table 4.2.

4.1.2 Addition of Two Arithmetic Numbers Each of 4 Bits

Let us assume that two arithmetic number **A** and **B** each of 4 bits, which are to be added. Let us assume bits in **A** are A_3, A_2, A_1 and A_0 and bits in **B** are B_3, B_2, B_1 and B_0 . Figure 4.2(a) shows four FAs processing the addition operation the four bits of **A** and four of **B**. Previous stage carry at right most stage is reset at 0 so that at least significant bit position FA works as a half adder. Figure 4.3 shows how to a subtractor builds from FA .

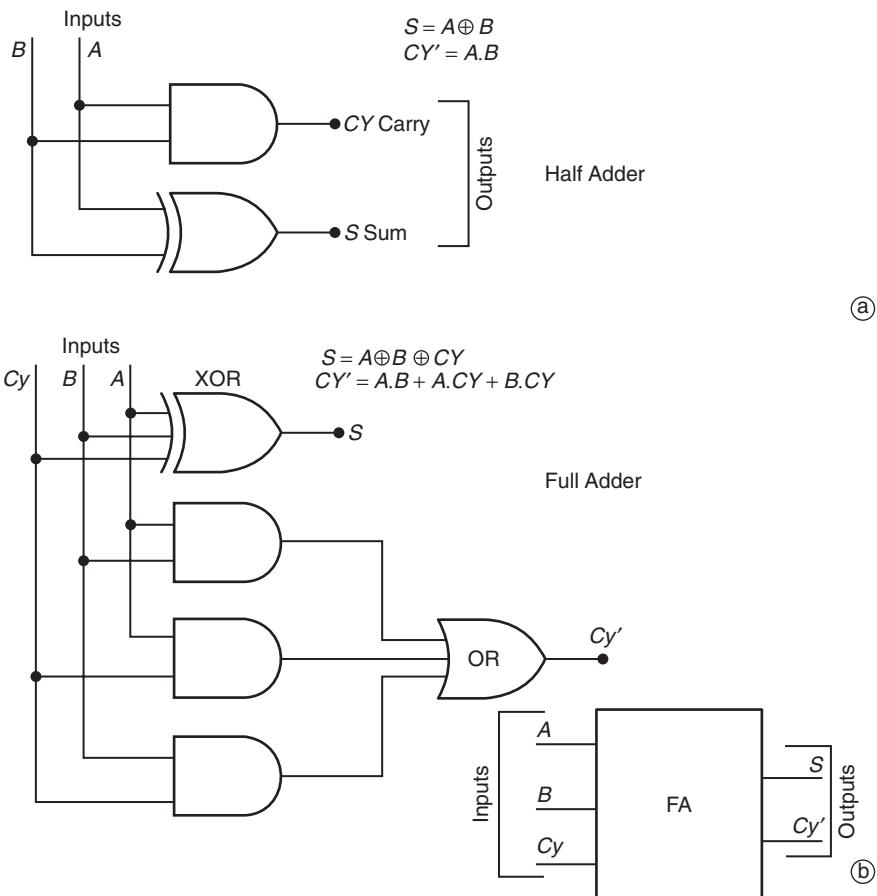


FIGURE 4.1 (a) Half Adder (addition of two bits without previous stage carry) (b) Full Adder (addition of two bits with previous stage carry) logic circuit and its simpler representation by an FA block.

TABLE 4.2 Truth table for full adder

Inputs left hand side			FA Outputs right hand side		Equation number
A	B	Cy	S	Cy'	...(4.2a)
0	0	0	0	0	...(4.2b)
0	0	1	1	0	...(4.2c)
0	1	0	1	0	...(4.2d)
0	1	1	0	1	...(4.2e)
1	0	0	1	0	...(4.2f)
1	0	1	0	1	...(4.2g)
1	1	0	0	1	...(4.2h)
1	1	1	1	1	

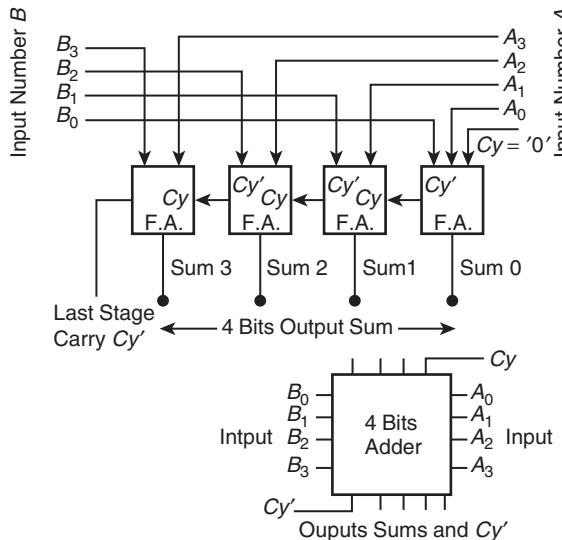


FIGURE 4.2 Four FAs processing the addition operation on four bits of A and four bits of B (Previous stage carry at right most stage is reset at 0).

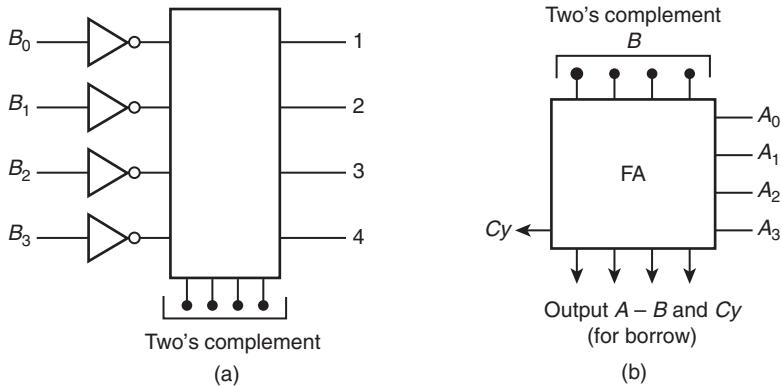


FIGURE 4.3 (a) Circuit for two's complement of B. (b) Implementation method subtraction by the addition of A with two's complement of B.

Karnaugh map from the truth table for an implementation of logic circuit for S (Sum) operations of Table 4.2 is as follows.

TABLE 4.3 Three variable Karnaugh Map for S

$B.Cy$	A	\bar{A}	A
$\bar{B}.Cy$	00	0	1
$\bar{B}.Cy$	01	1	0
$B.Cy$	11	0	1
$B.\bar{C}y$	10	1	0

Recall section 3.5.4 for a method to minimize the logic circuit from the Karnaugh map offset and diagonal adjacencies. Therefore, $S = Cy \cdot XOR(A \cdot XOR \cdot B)$. (There are two diagonal adjacencies with offset to each other.)

Karnaugh map from the truth table for a logic circuit implementation for Cy' (next stage carry) operations of Table 4.2 is as follows:

TABLE 4.4 Three variable Karnaugh Map for Cy'

$B \cdot Cy$	\bar{A}	A
	0	1
$\bar{B} \cdot \bar{C}y$	00	0
$\bar{B} \cdot Cy$	01	0
$B \cdot Cy$	11	1
$B \cdot \bar{C}y$	10	1

Minimizing the logic circuit from the Karnaugh map showing three quads, we get $Cy' = A \cdot B + A \cdot Cy + B \cdot Cy = A \cdot B + (A + B) \cdot Cy$.

Point to Remember

Full adder circuit is a basic building block in the adders and subtractors. Next stage carry $Cy' = A \cdot B + A \cdot Cy + B \cdot Cy$ and sum = $A \oplus B \oplus Cy$ in a full adder. It finds wide application in the arithmetic operation circuits.

4.1.2.2 Adder and Subtractor Implementation Using MSI ICs

For laboratory experiments, we can use an MSI IC of CMOS family 74HC82 or of TTL family 7482. This IC is a two-bit FA. Other ICs, which are usable, are CMOS based 74HC 83 or 74 HC 283 or TTL based 7483 OR 74283. These ICs are the four bits FAs. TTL based 74283 has a faster processing of the Cy at the higher stages than the 7483 TTL. (It also differs in pin numbers). In a computer, there is no single 4 bits adder unit, as in Seventies, but nowadays the 32 bit and 64 bit adder units are there. In the computer circuits, the FAs are implemented with NANDs or NORs only.

4.1.3 Subtraction of Two Arithmetic Numbers Each of 4 Bits

Let us assume two arithmetic numbers of A of 4 bits A_3, A_2, A_1 and A_0 and B of 4 bits B_3, B_2, B_1 and B_0 . Figure 4.3(a) showed implementation of two's complement of B . Figure 4.3(b) showed an implementation method for subtraction by the addition of A with two's complement of B .

Figure 4.4 shows a detailed circuit made for subtracting using the FAs as well as the XORs. Figure shows an adder cum subtractor circuit and implements two's complement of B as well as subtraction by an adder circuit. It uses a property of

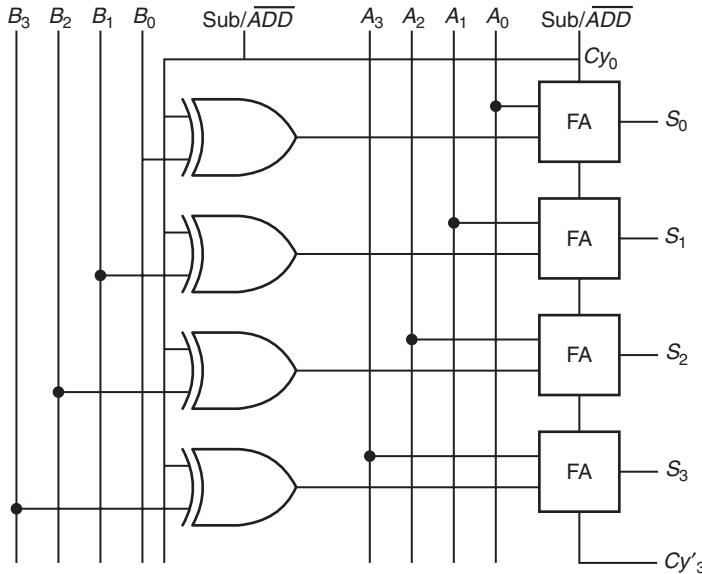


FIGURE 4.4 Adder cum subtractor circuit and implement two's complement of **B**, as well as subtraction by an adder circuit using the property of XOR gate that a two-input XOR gate if given input = 1, then its output is complement of the other input.

XOR gate that a two-input XOR gate if given input = 1, then its output is complement of the other input and if given input = 0, then its output is same as that of complement of the other input. (XOR is working like a controlled inverter. We get NOT operation if control input = 1 else output = input when control input = 0).

Case 1: For active 0 at Cy input ($\text{Sub}/\overline{\text{ADD}}$ line = 0) of $F4$, which is also one of two inputs of all the four XORs, obtain an arithmetic addition only by this circuit. Case 2: For active 1 at Cy input ($\text{Sub}/\overline{\text{ADD}}$ line = 1) of first $F4$, we get an increment by one circuit and we simultaneously get the one's complement of a bit of **B** at one of three inputs of each FAs . In this way, an active 1 generates 2's complement of **B**. The 2's complement of **B** is like a negative number corresponding to **B**. Therefore; by adding it to **A** we get an arithmetic subtraction.

Figure 4.4 circuit is useful both as a subtractor and an adder. It also work as a two's complement generator. It works as a two's complement generator when we connect all the four inputs of number **A** to A_3, A_2, A_1 and A_0 to the logic state 0s as the two's complementation is simply a subtraction from (called negation 0).

Point to Remember

A two-input XOR gate if given input = 1, then its output is complement of the other input. Two's complement of a binary number is first generated using an XOR left side circuit. Subtraction is then done with full adder circuit as a basic building block in the subtractor. A two-input XOR gate if given input = 0, then its output is same as the other input. The same circuit will then work as an adder.

4.2 DECODER

A decoder is a circuit that converts the binary information from one form to another. A decoder selects a unique combination of inputs and according to that information generates a unique output(s) at one-line (or at multiple lines).

4.2.1 Decoder (Line Decoder)

Let us assume that we have eight circuits to implement the different functions—One is for addition, other is for subtraction, other for incrementing, and so on. We have to select only one by giving appropriate instruction input. A decoder will let us select only one.

For example, we want to decode and get a selected output when the input is 000 (like an instruction) and activate the addition circuit, when 001 activate the subtraction, when 010 activate the increment circuit, and so on. Decoder can generate an output $Y_0 = 0$ or $Y_1 = 0$, or $Y_2 = 0$ or ... $Y_6 = 0$ or $Y_7 = 0$, respectively, for the next stage circuit so that either addition or subtraction or increment circuit or any one of the eight circuits can activate as per the inputs; 000, 001, 010, ..., respectively. Such a decoder circuit is called ‘line decoder’. It is also called 3-line to 8-line decoder or 1 of 8 decoder. Its truth table will be as per Table 4.5. Example 4.5 will describe an exemplary combination circuit for the decoder for implementing Table 4.5.

TABLE 4.5 A 3 to 8 line decoder for selecting 1 of the circuit out of 8 from the given 3 binary inputs

Inputs			Outputs							
A_2	A_1	A_0	\bar{Y}_0	\bar{Y}_1	\bar{Y}_2	\bar{Y}_3	\bar{Y}_4	\bar{Y}_5	\bar{Y}_6	\bar{Y}_7
0	0	0	0	z						
0	0	1	z	0	z	z	z	z	z	z
0	1	0	z	z	0	z	z	z	z	z
0	1	1	z	z	z	0	z	z	z	z
1	0	0	z	z	z	z	0	z	z	z
1	0	1	z	z	z	z	z	0	z	z
1	1	0	z	z	z	z	z	z	0	z
1	1	1	z	0						

Note: The z is either 1 or tristate. An appropriate next stage circuit enables by appropriate Y becoming = 0. Bar over Y s shows that line output is active 0. It represents active state for the next stage when at 0. When $z = 1$, an output Y reflects a maxterm. We can implement the Boolean function(s) by ANDing(s) the various maxterm outputs.

4.2.1.1 Decoder with Outputs Enabling Control (Gate) Input(s)

Certain decoders ICs (MSIs-Medium Scale Integrated Circuits) have in addition either a single control gate input or two or three control gate inputs. All control gate pins are activated then only the any of the output of the decoder can be activated else all the outputs will be in the tristate. Figure 4.5(a) shows a 3-line to 8-line (1 of 8) decoder with three control inputs \bar{G}_0 , \bar{G}_1 , and G_2 . \bar{G}_0 , and \bar{G}_1 control input

are activated by 0 and G_2 input is activated by 1. Table 4.6 gives the truth table of 3-line to 8-line decoder with 3 control gate input. The sign * means an output condition *tristate* instead of 1 or 0. A MSI chip 74138 has the pin corresponding to the decoder shown in the Figure 4.5(a).

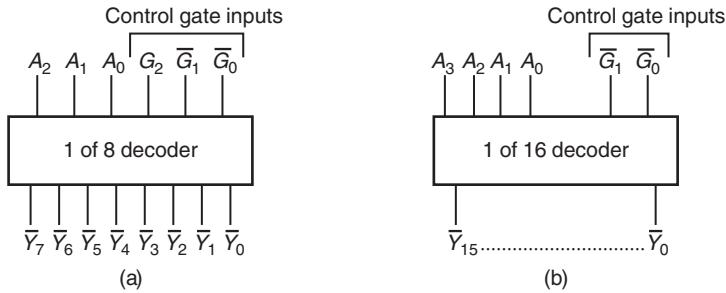


FIGURE 4.5 (a) 3-line (A, B, C) to 8-line decoder for selecting 1 of 8 with two active 0 and one active 1 control gate inputs $\bar{G}_0, \bar{G}_1, \bar{G}_2$ (b) 4-line to 16-line(A_0 to A_3) decoder for selecting 1 of 16 with two active 0 control gate inputs G_0, G_1 .

TABLE 4.6 Truth table of 3-line to 8-line decoder with 3 control gate inputs

Control (Gate) Inputs		A_0	A_1	A_2	\bar{F}_0	\bar{F}_1	\bar{F}_2	\bar{F}_3	\bar{F}_4	\bar{F}_5	\bar{F}_6	\bar{F}_7
' \bar{G}_1	' \bar{G}_2	\bar{G}_3	0	0	0	0	z	z	z	z	z	z
0	0	1	1	0	0	z	0	z	z	z	z	z
0	0	1	0	1	0	z	z	0	z	z	z	z
0	0	1	1	1	0	z	z	z	0	z	z	z
0	0	1	0	0	1	z	z	z	z	0	z	z
0	0	1	1	0	1	z	z	z	z	0	z	z
0	0	1	1	0	1	z	z	z	z	0	z	z
0	0	1	0	1	1	z	z	z	z	z	0	z
0	0	1	1	1	1	z	z	z	z	z	z	0
Any combination other than above		*	*	*	*	*	*	*	*	*	*	*

* All F outputs = z tristate or 1.

An output-enabling pin (input) helps in placing more decoders in parallel. For example, suppose, we need (4 of 16) decoder from two 3 to 8 decoders. (1) A_0, A_1 and A_2 are given to first (3 to 8) decoder at A_0, A_1, A_2 pins. A_3 is the input at G_2 . (2) A_0, A_1 and A_2 are given to second (3 to 8) decoder at A_0, A_1, A_2 pins. A_3 is the input at G_2 after a NOT operation. (3) \bar{G}_0 and \bar{G}_1 are made common in both the decoders, and are enabled by giving 0s as inputs to them.

Point to Remember

1. Decoder(s) finds wide application in selecting a memory chip or port in a computer systems from a given address of it at the input lines and/or at control lines.
2. A number of decoders can be arranged in a parallel or a tree topology to obtain a bigger number of input-bits decoder by using the control gate (enable) input(s) defining the specific decoder and applying the inputs to the enable inputs of all the decoders at the tree also in different combinations so that only one decoder activates at an instant.

4.2.2 The 1 of 2 and 1 of 4 Line Decoders

Figures 4.6(a) and (b) show the 1-line to 2-line and 1 of 4 decoders. Figure shows both active 1 and active 0 decoders. Truth tables are at the insets.

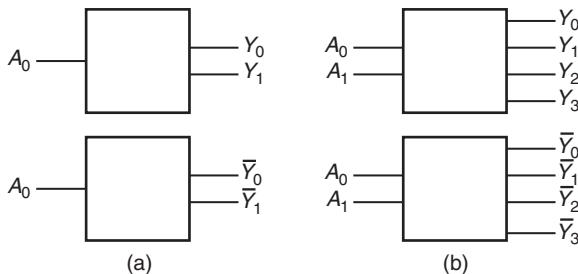


FIGURE 4.6 (a) 1-line to 2-line decoder (b) 1 of 4 decoders (truth tables at the inset).

4.2.3 The Four-line to 16-line Decoder

Figure 4.5(b) showed a 4 to 16 decoder with two active 0 control gate inputs. A MSI chip 74154 has the pins corresponding to the decoder shown in the figure. It has two gate enabling pins; both are active 0. When are then decoding action (active 0) takes place, else all the output remains 1. (MSI IC 74159 is open collector outputs version of 74154).

4.2.4 Function Specific Decoders

There can be function specific decoder. Consider the following type of decoder.

4.2.4.1 BCD to Decimal Line Decoder

Let a BCD to decimal decoder has 12 outputs, each one activating corresponding to the given BCD input. Truth table is per Table 4.7. Example 4.6 will describe an exemplary combination circuit for the decoder for implementing Table 4.7. MSI IO 7442 is a BCD to decimal decoder. It has 11 output pins, Y_0 to Y_{10} . It has four pins A_0, A_1, A_2 and A_3 for the BCD inputs.

TABLE 4.7 A BCD to decimal line decoder for selecting 1 of the circuit out of 10 from the given BCD inputs

Inputs					Outputs											
A_4	A_3	A_2	A_1	A_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9	Y_{10}	Y_{11}
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

4.2.4.2 Multi-line Decoder

The line decoders described above when selected activated only one output. It is possible to activate multi-line outputs for a specific combination of inputs. For example, gives three line outputs as 011 when input is 0000 and as 100 when input is 0001 (three more than the input).

4.2.4.3 Four Binary Input Seven Line-Decoder for the Seven LED Segments

The line decoders described above when selected activated only one output. It is possible to activate multi-line outputs for a specific combination of inputs.

For example, consider the truth table of a binary to LED one to seven segment display outputs. An LED digit has the seven segments. Assume that a segment lights up if its input logic is 1 (active 1) and light off if 0. When inputs are 0000, the outputs should be a, b, c, d, e and $f = 1$ and $g = 0$. Table 4.8 shows the outputs needed by different segments of LED. MSI IC 7447 is a BCD to 7-segment decoder cum driver. All outputs are active 0 (\bar{y}_0, \dots). It has a control gate pin for blanking input and ripple blanking output RBI/RBO when 0. It has a control gate pin for ripple blanking input RBI when 0. It has a control gate pin for LED test LT, which makes all outputs 0, when 0. Ripple means a (\bar{y}_6) carry from a previous stage (carry means blank the previous stage digit and light up the new one). These three pins enable all the multiple 7447s connected together to display digits with a set of multiple seven segments.

Example 4.7 will show an exemplary combination circuit with truth table as per Table 4.8. The circuit gives the seven line decoded outputs for the LED segments for displaying the as per the BCD input. The example will also show the requirements of an eight segment LED with a figure there.

TABLE 4.8 Seven segment (7-line) LED decoder for selecting one set of the circuit out of 10 different BCD inputs

Inputs					Outputs					
A_3	A_2	A_1	A_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6
				a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

4.2.4.4 Logic Design and Boolean Function Implementation Using the Decoders

Assume a Boolean function F is $\Sigma m(3, 7, 9, 10)$. Consider a decoder outputs with active 1 output and the n binary inputs (number of binary input combinations = 2^n , where n is the number of literals in F). Its outputs reflect the minterms with each minterm at each of the output. Therefore, if the OR operations are done on Y_3 , Y_7 , Y_9 and Y_{10} outputs of the four active 1 decoders with four binary inputs and 16 line outputs from Y_0 to Y_{15} , then the F implements by $Y_3 + Y_7 + Y_9 + Y_{10}$. A decoder circuit can be used to implement AND-OR circuit SOP Boolean expression when active output is 1. Example 4.8 will give a logic design and implementation using a decoder for $F_1 = \Sigma m(3, 7, 9, 10)$ and F_2 is $\Sigma m(2, 7, 12, 15)$.

Assume a Boolean function F' is $\Pi M(1, 7, 9, 13)$. Consider a decoder outputs with active 0 output and the n binary inputs (number of binary input possible combinations = 2^n , where n is the number of literals in F). Its outputs reflect the maxterms with one term each at each of the output. Therefore, if the AND operations are done on Y_1 , Y_7 , Y_9 and Y_{13} outputs on the active 0 decoders with four binary inputs and 16 line outputs from Y_0 to Y_{15} , then the F implements by $F = Y_1 \cdot Y_7 \cdot Y_9 \cdot Y_{13}$. A decoder circuit can be used to implement OR-AND arrays based circuit for the POS Boolean expression when active output is 0. Example 4.9 will give a logic design and implementation using a decoder for $F' = \Pi M(1, 7, 9, 13)$.

Point to Remember

A decoder with one-line output decoder is also a minterm or maxterm generator. Therefore, a decoder can be used to implement a Boolean expression(s).

4.3 ENCODER

1. An encoder is a circuit that converts the binary information from one form to another.
2. An encoder gives a unique combination of outputs according to the information at a unique input at one-line (or at multiple lines).
3. Action of a one active line input encoder is opposite of that of a one active line output decoder.
4. An encoder, which has multi-lines as the active inputs, is also called ‘priority encoder’.
5. Encoder can be differentiated from a decoder by greater number of inputs than the outputs when compared to the decoder.

A widely used application of an encoder is keypad (or keyboard) encoder. (A keypad has limited number of keys as in a telephone or a mobile phone. Keyboard has many more keys). At an instant, when a key presses, the input after an appropriate de-bouncing circuit applies the input to an encoder. The encoder generates an 8-bit code for a given active input corresponding to the pressed-key.

4.3.1 Encoder (Line Encoder)

Let us assume that we have eight inputs from the different functions. One is from completion of addition, other is for completion of subtraction, other on completing increment, and so on. We have to find which of the operation has been completed. An encoder will let us find one.

For example, we want to encode and get an output 000 when addition completes and activates an input A_0 , get an output 001; when a subtraction completes it activates another input A_1 , get an output 010, and so on. Encoder can generate an outputs $Y_2Y_1Y_0 = 000$ or 001 or 010 or 011 and so on depending on whether $A_0 = 1$ or $A_1 = 1$ or $A_2 = 1$ or $A_3 = 1$ and so on, respectively. Such an encoder circuit is called ‘ 2^n to n line encoder’ also. It is also called 8-line to 3-line encoder or 8 of 1 encoder when $n = 3$. Table 4.9 gives a part of the truth table as a functional table showing nine rows only in place of 256 rows needed for the 8-input case. Example 4.13 will describe an exemplary combination circuit for the encoder for implementing Table 4.9.

MSI IC74148 is an 8 to 3 encoder with an active 0 gate-enable pin for the enabling the inputs and another active 0 gate-enable pin for enabling the outputs.

Point to Remember

An application of an encoder is to generate at an instant set of outputs, which can be a code or an address corresponding to a given input which is active at that instant. If there are N inputs and only one of it is active at a time, then the encoder outputs at m pins where $2^m = N$ will provide us the means to find which of the N inputs is active at present with the remaining of the inputs (z and z' in Table 4.9) are in either the tristate or inactive states.

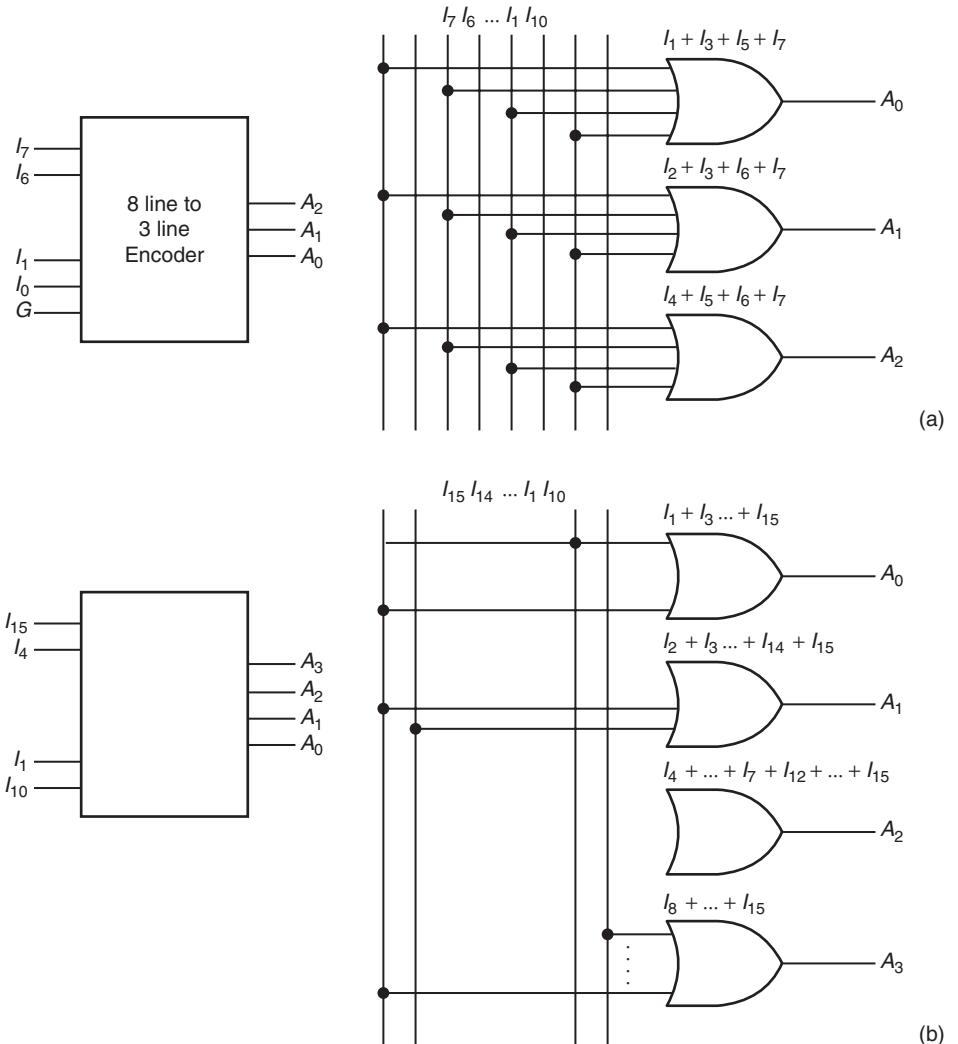


FIGURE 4.7 (a) 8-line to 3-line encoder for finding which 8 out of 1 (b) 16-line to 4-line encoder for finding 16 of 1.

4.3.2 Encoder (Priority Encoder)

Suppose multiple inputs can activate simultaneously. Assume that A_6 and A_1 activate simultaneously. When input $A_1 = 1$, the output = 001 and $A_6 = 1$ then output = 110. A priority encoder will resolve the simultaneous activations of multiple inputs, by giving the output as per the highest priority one. It will give the output = 110 for A_6 . Such an encoder is called a priority encoder. Table 4.9 is also a truth table for a priority encoder when an encoder design is implemented for $z = X$ (don't care) and only $z' =$ either 1 or 0 depending upon whether the line encoder design is for input active 1 or active 0, respectively.

TABLE 4.9 Functional Table 8 to 3 line encoder for finding I at the input circuit out of 8 inputs and give the 3-bit binary output

Inputs								Output			
A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	S	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	0	0
1	z'	1	0	0	0						
z	1	z'	z'	z'	z'	z'	z'	1	0	0	1
z	z	1	z'	z'	z'	z'	z'	1	0	1	0
z	z	z	1	z'	z'	z'	z'	1	0	1	1
z	z	z	z	1	z'	z'	z'	1	1	0	0
z	z	z	z	z	1	z'	z'	1	1	0	1
z	z	z	z	z	z	1	z'	1	1	1	0
z	z	z	z	z	z	z	1	1	1	1	1

Note: (1) $S = 0$ when none of the input is active. $S = 1$ when at least one input is active. (2) Both z and z' are either 0 or 1 depending upon whether the line encoder design is for I active 1 or active 0, respectively. (3) When an encoder design is for a priority encoder, then the design is implemented for $z = X$ (don't care) and only z' either 0 or 1 depending upon whether the line encoder design is for input active 1 or active 0, respectively.

4.3.3 BCD 10 of 1 Four-bit Encoder

Table 4.10 gives the truth table of the combinational circuit for BCD priority encoder. X means don't care. p means active. $q = \bar{p}$. The input G means gate pin to control the output. If G is active then only all the outputs S , B_5 , B_4 , B_3 , B_2 and B_1 are not in idle state (not in tristate). Figure 4.8(a) shows a decimal to BCD output encoder [Also called (10 of 1) four bits encoder (BCD priority encoder)] MSI IC 74147 is a decimal (9-active 0 inputs) to BCD (four outputs active 0) encoder. BCD output = 1111 when none of the input is 0.

4.3.4 Octal 8 of 1 Three-bit Encoder and Hexadecimal Encoder

Figure 4.8(b) shows an octal to binary (8 of 1) three-bit encoder. Table 4.11 gives the truth table of the combinational circuit of a hexadecimal to binary (14 of 1) four-bit encoder.

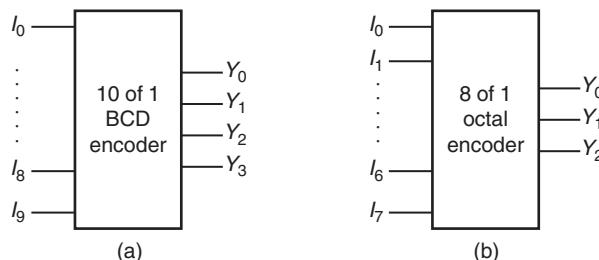


FIGURE 4.8 (a) A decimal to (BCD) output encoder [Also called (10 of 1) four bits encoder (BCD priority encoder)] (b) an octal to binary (8 of 1) three bits encoder.

TABLE 4.10 Decimal to BCD Encoder

Gate	Only One out of 10 is an Active input										Encoded BCD outputs					
	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	S	B_4	B_3	B_2	B_1	B_0
G	p	q	1	0	0	0	0	0								
G	X	p	q	1	0	0	0	0	1							
G	X	X	p	q	1	0	0	0	1	0						
G	X	X	X	p	q	q	q	q	q	q	1	0	0	0	1	1
G	X	X	X	X	p	q	q	q	q	q	1	0	0	1	0	0
G	X	X	X	X	X	p	q	q	q	q	1	0	0	1	0	1
G	X	X	X	X	X	X	p	q	q	q	1	0	0	1	1	0
G	X	X	X	X	X	X	X	p	q	q	1	0	0	1	1	1
G	X	X	X	X	X	X	X	X	p	q	1	0	1	0	0	0
G	X	X	X	X	X	X	X	X	X	p	1	0	1	0	0	1
G	X	X	X	X	X	X	X	X	X	X	1	1	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: X don't care in priority encoder, else equals q.

TABLE 4.11 Hexadecimal (16 bits) to Nibble (4 bits) Encoder

(Input) Control pin	One out 16 active inputs active Input	Address outputs if an (output) control pin active			
Active	I_0 , active, rest inactive	B_3	B_2	B_1	B_0
Active	I_1 active, rest inactive	0	0	0	0
Active	I_2 active, rest inactive	0	0	1	0
.....
.....
Active	I_{13} active, rest inactive	1	1	0	1
Active	I_{14} active, rest inactive	1	1	1	0
Active	I_{15} active, rest inactive	1	1	1	1
Inactive	I_0 $I_{15} = x$	*			

The x means don't care. The sign * means tristate.

4.4 MULTIPLEXER

A multiplexer is a circuit that selects a input line among the input lines as per the channel-selector logic-inputs and gives that at the output. A multiplexer selects a unique input line according to the channel selector inputs to it.

4.4.1 Multiplexer (Line Selector)

Let us assume that we have two logic circuits that provide the outputs. One I_1 is for a logic function F_1 and other I_2 is for F_2 . We have to select only one by giving

appropriate instruction. A multiplexer will select for the output only one. Figure 4.9(a) shows a 2-channel input-selector 2 to 1 multiplexer with one channel selector pin A (0 for channel F_0 and 1 for channel F_1). Its functional table is given in the inset at the figure.

Figure 4.9(b) shows 4-channels input selector; 4 to 1 multiplexer with two channel-selector pins A_0 and A_1 (when 00 the channel F_0 ; when 01 the channel F_1 ; when 10 the channel F_2 and when 11 the channel F_3). Its functional table is given in the inset at the figure. MSI IC74156 is a four-channel multiplexer. It selects the inputs I_0, I_1, I_2 and I_3 as per the channel selector lines A_0 and A_1 . When A_0 and A_1 are 00, $F = I_0$, 10 then $F = X_1$, and 01 then $F = X_2$, 11 then $F = I_3$, respectively.

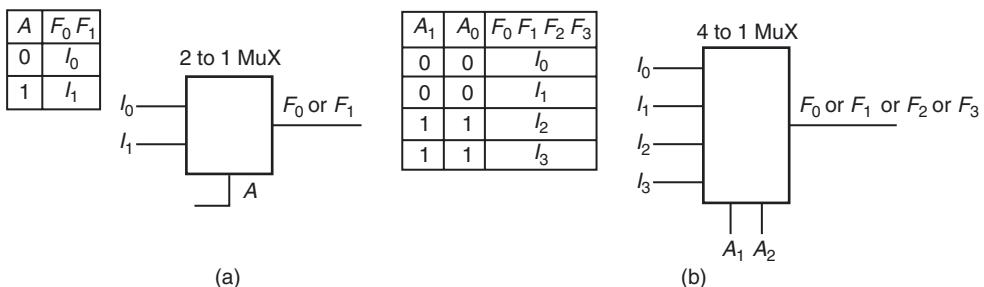


FIGURE 4.9 (a) A 2-channel input-selector using a 2 to 1 multiplexer with one channel selector pin A = (0 for channel F_0 and 1 for channel F_1) (b) 4-channel input selector (4 to 1 multiplexer) with two channel selector pins A_0 and A_1 , when 00 the channel F_0 , when 01 the channel F_1 , when 10 the channel F_2 when 11 channel F_3 is selected.

Point to Remember

A multiplexer provides one of the output path (channel) for the one of the channel data from a number of channels' data present at a given instant. Its important application is in sharing the circuits, ports, devices and resources.

Examples 4.15 and 4.16 will describe an exemplary combination circuits for the multiplexers for implementing tables in insets of Figures 4.9(a) and 4.9(b), respectively.

4.4.2 Multiplexer with Outputs Enabling Control (gate) Pin(s)

Certain multiplexers ICs [MSIs (Medium Scale Integrated circuits)] have in addition either a single control gate pin or two or three control gate pins. A control gate pin(s) when activated then only the output of the multiplexer activate else all the outputs will be in the tristate. Figure 4.10 shows a 16-line to 1-line (16 of 1) multiplexer with one control pins \bar{G} control pin activates the output when it is at 0. A 16 to 1 line multiplexer for selecting 1 of the circuit out of 16 from the given 4 binary channel selection inputs. There is a control gate pin with active 0. The sign * means an output condition *tristate* instead of 1 or 0.

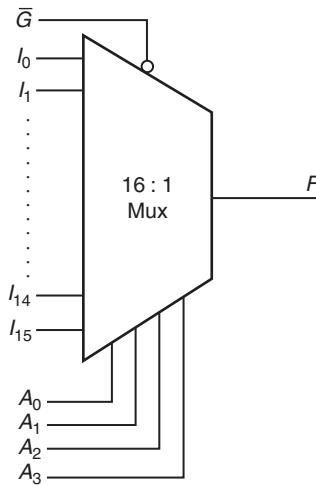


FIGURE 4.10 A 16-line to 1-line (16 of 1) multiplexer with one control pins ' \bar{G} ' control pin activates the output when it is at 0.

TABLE 4.12 Truth table of 4-line to 26-line multiplexer with one control gate pin

Gate \bar{G}_1	A_0	A_1	A_2	A_3	Input I	Output F
0	0	0	0	0	I_0 to I_{15}	I_0
0	1	0	0	0	I_0 to I_{15}	I_1
0	0	1	0	0	I_0 to I_{15}	I_2
0	1	1	0	0	I_0 to I_{15}	I_3
0	0	0	1	0	I_0 to I_{15}	I_4
0	1	0	1	0	I_0 to I_{15}	I_5
0	0	1	1	0	I_0 to I_{15}	I_6
0	1	1	1	0	I_0 to I_{15}	I_7
0	0	0	0	1	I_0 to I_{15}	I_8
0	1	0	0	1	I_0 to I_{15}	I_9
0	0	1	0	1	I_0 to I_{15}	I_{10}
0	1	1	0	1	I_0 to I_{15}	I_{11}
0	0	0	1	1	I_0 to I_{15}	I_{12}
0	1	0	1	1	I_0 to I_{15}	I_{13}
0	0	1	1	1	I_0 to I_{15}	I_{14}
0	1	1	1	1	I_0 to I_{15}	I_{15}
1	X	X	X	X	X	*

X means 0 or 1 (don't care condition). I_0 to I_{15} means 16 input lines. * means tristate.

4.4.2.1 Multiplexers Arranged as a Tree

How do we get the (m of 1) multiplexing from i numbers of the (m' of 1) multiplexers? Here $m = i \cdot m'$ where i is an integer and $m' = 2^n$ where n is the number of channel selector lines at each of the i multiplexers. Multiplexers arranged as a tree facilitates this. Figure 4.11 shows an exemplary tree type arrangement of the five multiplexers. Four are at the leaves and one at the root. We use the multiplexers of truth table shown in Figure 4.9(b). $m' = 4$, $n = 2$ and design circuit for $m = 16$ and $i = 4$ in this example. We get the (16 of 1) multiplexing from five numbers (4 of 1) multiplexers. Function table is as per Table 4.13. (GS are control bits for the MUXs.)

Channel selector inputs at the root multiplexer R is selecting a leaf multiplexer (L_0 or L_1 or L_2 or L_3) among the four multiplexers at leaves L_0 , L_1 , L_2 and L_3 . The channel selector inputs A_0 and A_1 select the one out of the four inputs among I_0 , I_1 , I_2 and I_3 at each of the multiplexer.

Point to Remember

A number of multiplexers can be arranged in a tree topology to obtain a bigger numbers of channels multiplexer.

TABLE 4.13 An exemplary tree type arrangement for (16 of 1) multiplexing from our multiplexers at the leaves

Inputs at Leaf A			Inputs at Leaf B			Inputs at Leaf C			Inputs at Leaf d			Inputs at Root R			Y	
GA	I	A_0	A_1	GB	I	A_0	A_1	GC	I	A_0	A_1	GD	I	A_0	A_1	F
1	IA	0	0	1	IA	0	0	1	IA	0	0	1	IA	0	0	I_0A
1	IA	1	0	1	IA	1	0	1	IA	1	0	1	IA	1	0	I_1A
1	IA	0	1	1	IA	0	1	1	IA	0	1	1	IA	0	1	I_2A
1	IA	1	1	1	IA	1	1	1	IA	1	1	1	IA	1	1	I_3A
1	IB	0	0	1	IB	0	0	1	IB	0	0	1	IB	0	0	I_0B
1	IB	1	0	1	IB	1	0	1	IB	1	0	1	IB	1	0	I_1B
1	IB	0	1	1	IB	0	1	1	IB	0	1	1	IB	0	1	I_2B
1	IB	1	1	1	IB	1	1	1	IB	1	1	1	IB	1	1	I_3B
1	IC	0	0	1	IC	0	0	1	IC	0	0	1	IC	0	0	I_0C
1	IC	1	0	1	IC	1	0	1	IC	1	0	1	IC	1	0	I_1C
1	IC	0	1	1	IC	0	1	1	IC	0	1	1	IC	0	1	I_2C
1	IC	1	1	1	IC	1	1	1	IC	1	1	1	IC	1	1	I_3C
1	ID	0	0	1	ID	0	0	1	ID	0	0	1	ID	0	0	I_0D
1	ID	1	0	1	ID	1	0	1	ID	1	0	1	ID	1	0	I_1D
1	ID	0	1	1	ID	0	1	1	ID	0	1	1	ID	0	1	I_2D
1	ID	1	1	1	ID	1	1	1	ID	1	1	1	ID	1	1	I_3D
0	X	X	X	0	X	X	X	0	X	X	X	0	X	X	X	*

/A means I_0 to I_3 at leaf A, /B means at B, /C means at C and /D means at D. * means tristate. I_0A means F = Input I_0 at leaf A multiplexer. Y is the output

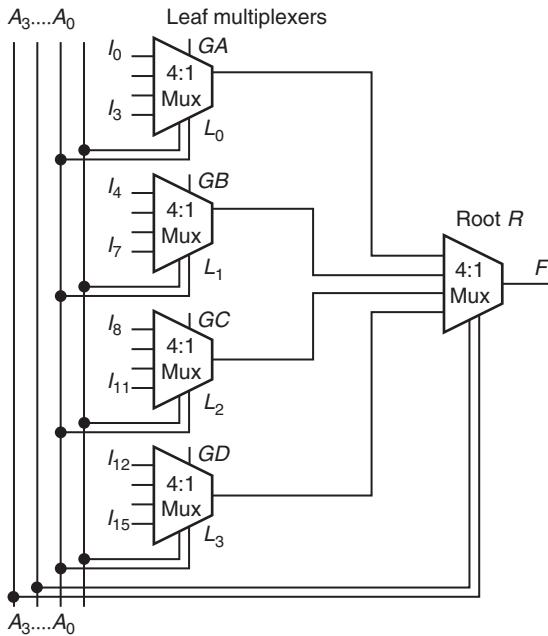


FIGURE 4.11 An exemplary tree type arrangement of the five multiplexers: four at the leaves and one at the root. We use the multiplexers of truth table shown in Figure 10.9(b) and $m' = 4$, $n = 2$, $m = 16$ and $i = 4$.

4.4.2.2 Logic Design and Boolean Function Implementation Using a Multiplexer

Example 4.18 will show a logic design and implementation using a multiplexer for $F_1 = \Sigma m(3, 7, 9, 10)$ and F_2 is $\Sigma m(2, 7, 12, 15)$. Example 11.5 will show a use of mux as building block for Gray code converter. Assume a Boolean function F is $\Sigma m(3, 7)$ for a three variable expression. Assume channel selector inputs are A_0 , A_1 and A_2 . Let the inputs to an 8 of 1 multiplexer are X_0, X_1, \dots, X_6 and X_7 . If a Karnaugh map is drawn for a three variable input, then at $m(3)$ position and at $m(7)$ position there is 1. (Refer Figure 4.12a).

The output F' is as follows:

$$\begin{aligned} F' = & X_0 \cdot A_0 \cdot A_1 \cdot A_2 + X_1 \cdot A_0 \cdot A_1 \cdot A_2 + X_2 \cdot A_0 \cdot A_1 \cdot A_2 + X_3 \cdot A_0 \cdot A_1 \cdot A_2 \\ & + X_4 \cdot A_0 \cdot A_1 \cdot A_2 + X_5 \cdot A_0 \cdot A_1 \cdot A_2 + X_6 \cdot A_0 \cdot A_1 \cdot A_2 + X_7 \cdot A_0 \cdot A_1 \cdot A_2. \end{aligned}$$

Now if $F = \Sigma m(3, 7)$ is to be implemented, if X_3 and X_7 are given as 1 (at the positions of 1s at the Karnaugh map) and remaining inputs = 0 then Boolean function F is $\Sigma m(3, 7)$ is implemented by the multiplexer. Figure 4.12(b) shows implementation by 8 of 1 Mux.

Suppose we are using a gated output multiplexer. $F' = F'^* \text{ if gate is disabled and } F = F'. G \text{ if gate is enabled.}$

Example 4.19 will give a logic design of 8 of 1 Mux using three 4 of 1 Muxes. Example 4.20 will give a logic design and implementation using a multiplexer for $F' = \Pi M(1, 7, 9, 13)$ (Implementation of POS Boolean expression).

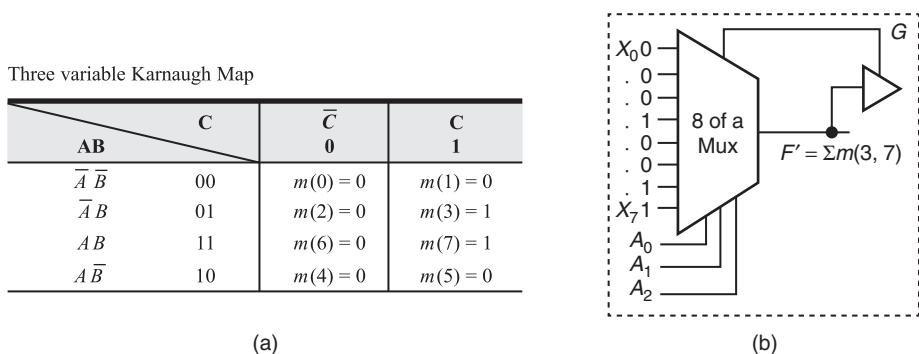


FIGURE 4.12 (a) Karnaugh map for $F = \Sigma m(3, 7)$ for a three variable expression with 1 at the cells corresponding to $m(3)$ and $m(7)$. Therefore the X_3 and X_7 are given as 1 (at the positions of 1s at the Karnaugh map) and remaining inputs = 0. (b) Boolean variables are given as the inputs at A_0 , A_1 and A_2 when implementing the circuit with the help of a multiplexer.

Point to Remember

A multiplexer with one-line output is also a Boolean expression implementer and implements of a SOP (or POS cells) Karnaugh map. The literal variables are the inputs to the channel selector lines and minterm (or mux-term) selectors are given at the input lines of a multiplexer.

A three-variable expression and its Karnaugh map can also be implemented by a 4 to 1 multiplexer with two variables at the channel selector lines and third variable, its complement or 1 or 0 at the channel input lines (Example 4.19).

4.5 DEMULTIPLEXER

4.5.1 Demultiplexer Definition

A demultiplexer is a circuit that takes the binary information and sends it to select appropriate channel. A demultiplexer gives an output at a unique channel (one-line or multiple line) among according to a unique combination of the channel selector inputs at the input at one-line (or at multiple lines).

MSI IC74155 is a dual four-channel demultiplexer. It gives two output Y as well as its complement at the selected channel. It gives output Y_0 , Y_1 , Y_2 and Y_3 as per the channel selector lines A_0 and A_1 . When A_0 and A_1 are 00, $Y_0 = X$, 10 then $Y_1 = X$, and 01 then $Y_2 = X$, 11 then $Y_3 = X$, respectively. X is the input to be sent to an addressed channel.

MSI IC 74156 has open collector outputs. It has dual inputs, input and its complement.

Point to Remember

Action of a one active line input demultiplexer is opposite of that of a one active line output multiplexer. Demultiplexer can be differentiated from multiplexer by a greater number of outputs than inputs compared to a multiplexer.

4.5.1.1 Demultiplexer (Line Demultiplexer)

Let us assume that we have eight line-outputs for activating the different functions at the different channels. One is for initiating an addition; other is for initiating a subtraction; other initiating an increment and so on. We select one operation from the channel selection inputs. A demultiplexer will let us find one. An input 1 is sent to the addressed channel by the demultiplexer.

For example, we want to get output at address 0, $Y_0 = 1$ when the three channel selector (address selector) pins are at 000, output at address 1, $Y_1 = 1$ when selector pins are at 001, and so on. Output at address 7, $Y_7 = 1$ when selector pins are at 001. Remaining outputs remain at 0s. (Active 1 output case).

TABLE 4.14 Truth table of a demultiplexer for sending 1 at output using 8-line F_0 to F_7 selector from 3 channel select bits

Channel (address) select inputs			Data		Outputs at the 8 channels							
A_0	A_1	A_2	I	G	\bar{F}_0	\bar{F}_1	\bar{F}_2	\bar{F}_3	\bar{F}_4	\bar{F}_5	\bar{F}_6	\bar{F}_7
0	0	0	/	1	1	z	z	z	z	z	z	z
1	0	0	/	1	z	1	z	z	z	z	z	z
0	1	0	/	1	z	z	1	z	z	z	z	z
1	1	0	/	1	z	z	z	1	z	z	z	z
0	0	1	/	1	z	z	z	z	1	z	z	z
1	0	1	/	1	z	z	z	z	z	1	z	z
0	1	1	/	1	z	z	z	z	z	z	1	z
1	1	1	/	1	z	z	z	z	z	z	z	1
Any combination of inputs			x	0	*	*	*	*	*	*	*	*

'/ is the input at pin I to be sent to the addressed channel. The $z = 0$ or 1 or $*$ as per the demultiplexer design. x means don't care condition.

Alternatively, $Y_0 = 0$ at one line when the three channel selector (address selector) pins are at 000, output $Y_1 = 0$ when selector pins are at 001, and so on. Output $Y_7 = 0$ when selector pins are at 001. Remaining outputs remain at 0s. (Active 0 output case).

Example 4.21 will describe an exemplary combination circuit for the demultiplexer for implementing Table 4.14.

4.5.1.2 Demultiplexer as a Decoder

Let us compare the functional truth tables in Table 4.6 and 4.14. We note that in a demultiplexer if the encoded inputs are given at the channel address lines and a data is applied at control enable pin, then demultiplexer also functions as a line decoder. Therefore MSI IC74154 is a four-channel demultiplexer, which also functions as 2 to 4 decoder. (MSI IC 74159 is open collector outputs version of 74154).

Point to Remember

Like a decoder, a demultiplexer can also be used as a minterm or maxterm generator. Therefore, a demultiplexer can be also used to implement a Boolean expression(s). It is when the channel select pins are used to give Boolean inputs. If $I = 0$ and $z = 1$, the circuit is maxterm selector. If $I = 1$ and $z = 0$, the circuit is minterm selector.

4.5.1.4 Demultiplexers Arranged in Tree Topology

A set of output-enabling pin helps in placing more demultiplexers in parallel. For example, consider a circuit for four pins for the addressed selector for the 16 outputs (channels). [1] A_0, A_1 and A_2 are given to first 3 to 8 demultiplexer and data to I pin. Enable pin $E = 1$ (active 1 case). [2] A_0, A_1 and A_2 are given to second demultiplexer also. I' is the input at I pin. E is connected to E of the first demultiplexer after a NOT operation. [3] I and I' are made common in both the demultiplexers. Now when $E = 1$, the first demultiplexer gives the output, else the second.

Points to Remember

A number of demultiplexers can be arranged in tree topology to obtain a bigger number of input-bits decoder by using the control gate (enable) pin(s) and applying inputs to the enable pins also in different combinations.

■ EXAMPLES

Example 4.1

Write a general formula for an i -th stage carry to next stage using AND operation as a new term and OR operation as a propagating term at the i -th stage.

Solution

Using the formula; $Cy = A.B + A.Cy' + B.Cy'$, the carry at stages 0 to $(m-1)$ th for an m -bit adder can be written as follows:

$$Cy'_0 = 0 = Cy_{-1}; Cy_{-1} = \text{Previous stage carry at the } 0\text{-th stage} = 0.$$

$Cy_0 = A_0.B_0 = n_0 + Cy_{-1}$; (Let n_0 is a new term product of the adding elements at stage 0).

$Cy_1 = A_1.B_1 + (A_1 + B_1), n_0 = n_1 + p_1.n_0$; (Let p_1 is the propagating term, which is equal to OR operation of the adding elements at stage 1. The n_1 is the new term at the stage 1).

$Cy_2 = A_2.B_2 + (A_2 + B_2)(n_1 + p_1.n_0) = n_2 + p_2.(n_1 + p_1.n_0)$; (Let p_2 is the propagating term, which is equal to OR operation of the adding elements at stage 2. The n_2 is the new term at the stage 2, equal of product of the adding elements).

Therefore, a general formula for next stage carry from an i -th stage is as follows:

$Cy_i = n_i + p_i.n_{i-1} + p_i.p_{i-1}.n_{i-2} + p_i.p_{i-1}.p_{i-2}.n_{i-3} + \dots + p_i.p_{i-1}.p_{i-2} \dots p_0.n_0 + p_i.p_{i-1}.p_{i-2} \dots p_{-1}.Cy_{-1}$, where $p_{-1} = 1$, Cy_{-1} is the input carry to the 0th stage, p_0 is 0th stage OR operation of adding elements, ..., p_i is the i -th stage OR operation and n_i is the i -th stage product (AND) operation between the adding elements.

An adder circuit of m bits and 0^{th} to $(m - 1)^{\text{th}}$ stages is based upon the four stage exemplifying circuit of Figure 4.2 and C_y of last stage is given by the above equation. Each calculation of ORs among the propagating terms, second, third, fourth takes longer time than the previous stage.

Example 4.2

Calculate the delay at an i -th stage in finding C_y , assuming that each stage of FA in Figure 4.2 takes propagation time = t_s .

Solution

Every stage gives output carry after t_s to the next stage. Note that inputs A and B are readily available but the carry is available after the previous stage result is obtained. Time taken at the i -th stage = $i \cdot t_s$.

Example 4.3

Implement a carry lookahead circuit for each stage using the general formula derived in Example 4.1 for an i -th stage carry to next stage and generate the next stage carry using AND operation as a new term and OR operation as a propagating term at the i -th stage.

Solution

Look ahead carry generator is a logic circuit to implement at each i -th stage, n_i , p_i and s_i . It calculates the sum term in advance so that carry input at each stage is readily available. This creates a fast adder circuit. Figure 4.13(a) shows a summation element at each stage, which has two outputs n_i and p_i . Figure 4.13(b) shows a fast addition circuit using the carry lookahead general formula derived in Example 4.1.

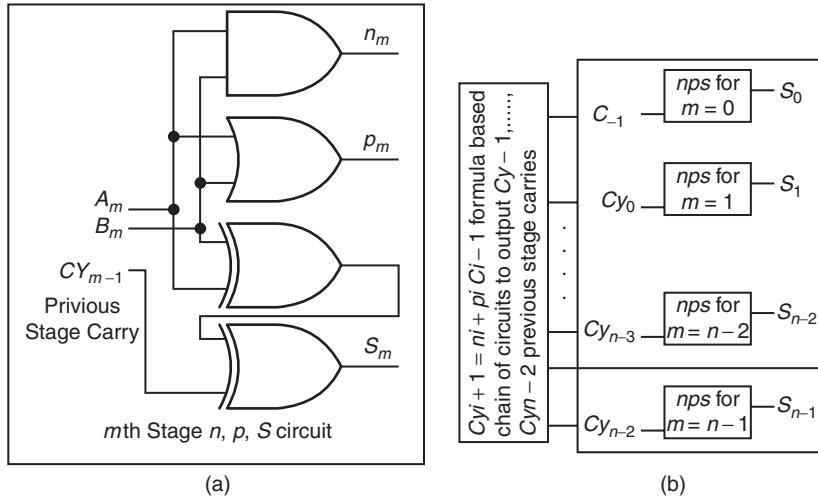


FIGURE 4.13 (a) Summation element at each stage, which has two outputs n_i and p_i (b) Fast addition circuit using the carry lookahead general formula derived in Example 4.1.

Example 4.4

Calculate the delay at an i -th stage using the lookahead carry generator circuit. Assume that summation unit in Figure 4.3 takes time = t_s . Time taken by lookahead carry generator = t_c .

Solution

Time taken at the i -th stage = $t_c + t_s$. (Every stage gives output carry after t_s to the next stage).

Example 4.5

Formulate the problem and how will you implement a 3-line to 8-line decoder of truth table as per Table 4.5. Output is active at 0 and inactive at = 1 (output when line not selected).

Solution

Assume that outputs are Y_0 to Y_7 . Since 0 occurs only at one output and remaining outputs remains 1, therefore, it will easier to implement the logic design by POS form of three variable (literals). From the Table 10.5, the outputs in terms of the max-terms are as follows: $\bar{Y}_0 = M(0)$; $\bar{Y}_1 = M(1)$; $\bar{Y}_2 = M(2)$; $\bar{Y}_3 = M(3)$; $\bar{Y}_4 = M(4)$; $\bar{Y}_5 = M(5)$; $\bar{Y}_6 = M(6)$; $\bar{Y}_7 = M(7)$.

$$\begin{aligned}\bar{Y}_0 &= (A + B + C); \bar{Y}_1 = (A + B + \bar{C}); \bar{Y}_2 = (A + \bar{B} + C); \bar{Y}_3 = (A + \bar{B} + \bar{C}); \bar{Y}_4 = (\bar{A} + \\ &B + C); \bar{Y}_5 = (\bar{A} + B + \bar{C}); \bar{Y}_6 = (\bar{A} + \bar{B} + C); \bar{Y}_7 = (\bar{A} + \bar{B} + \bar{C}); A = A_2, B = A_1 \text{ and} \\ &C = A_0.\end{aligned}$$

Figures 4.14(a) and (b) show the logic circuit using OR-AND gates and NAND gates.

Example 4.6

Formulate the problem and how will you implement a BCD- to 12 -line decimal decoder on defined in truth table (Table 4.7). Output is active at 1 and inactive at = 0 (output when line not selected).

Solution

From the Table 4.7, the outputs in terms of the minterms are as follows: $Y_0 = m(0)$; $Y_1 = m(1)$; $Y_2 = m(2)$; $Y_3 = m(3)$; $Y_4 = m(4)$; $Y_5 = m(5)$; $Y_6 = m(6)$; $Y_7 = m(7)$; $Y_8 = m(8)$; $Y_9 = m(9)$; $Y_{10} = m(16)$; $Y_{11} = m(17)$

The combinational circuit is as per Figure 4.15.

Example 4.7

Formulate the problem and how will you implement a BCD- to seven segments LED decoder with segments as per Figure 4.16(a) and truth table as per Table 4.8. Output is active at 1 and inactive at = 0 (output when line not selected).

Solution

Recall columns of Table 4.8 for a to g . We observe that the set of outputs for the segments should be as follows.

$$\begin{aligned}a &= \sum m(0, 2, 3, 5, 6, 7, 8, 9); \\ b &= \sum m(0, 1, 2, 3, 4, 7, 8, 9); \\ c &= \sum m(0, 1, 3, 4, 5, 7, 8, 9); \\ d &= \sum m(0, 2, 3, 5, 6, 8, 9); \\ e &= \sum m(0, 2, 6, 8); \\ f &= \sum m(0, 4, 5, 6, 8, 9); \\ g &= \sum m(2, 3, 4, 5, 6, 8, 9);\end{aligned}$$

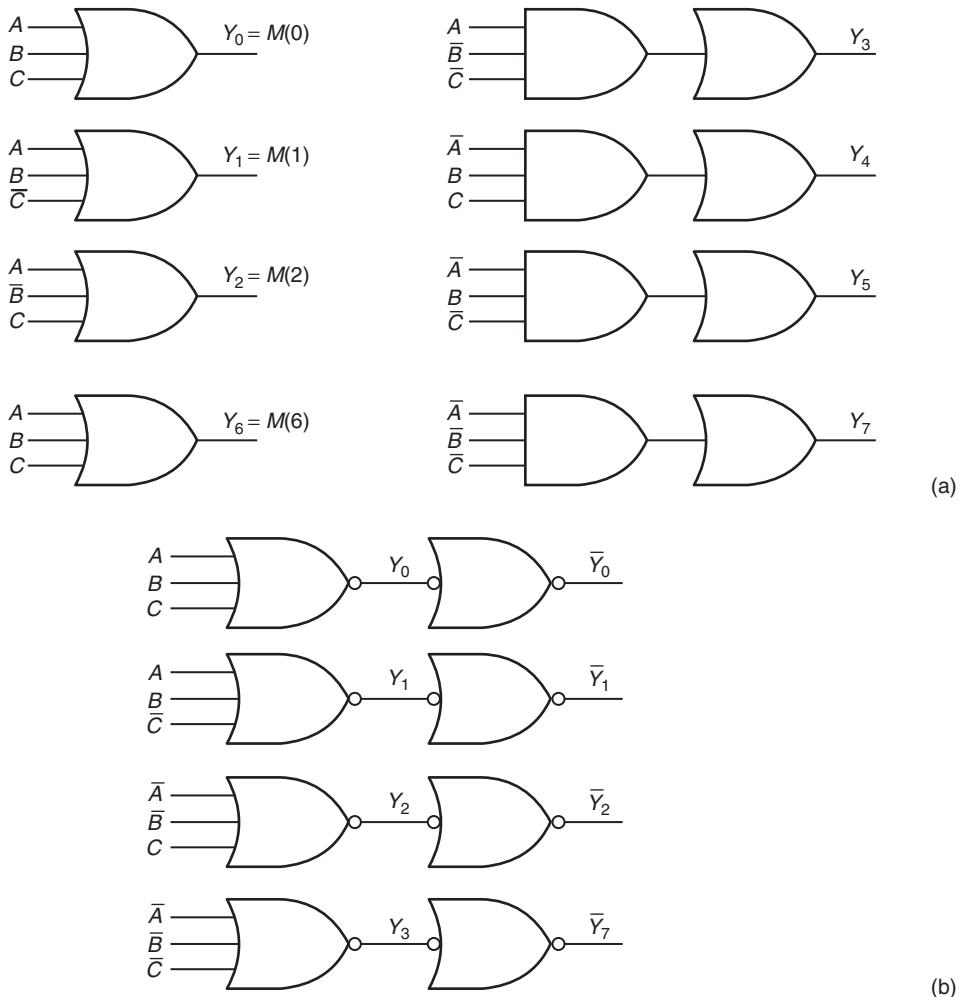


FIGURE 4.14 (a) Logic circuit using OR-AND gray of gates (b) Logic circuit using NOR gates for a decoder truth table as per Table 4.5.

One of the ways of implementing above functions are using OR gates to the outputs of a decoder Y_0 to Y_9 as inputs and get the outputs a, b, c, d, e, f and g . Figure 4.16(b) shows the combinational circuit made from a decoder.

Example 4.8

Logic design and implement the Boolean functions $F_1 = \Sigma m(3,7,9,10)$ and $F_2 = \Sigma m(2,7,12,15)$ by using a decoder.

Solution

Since highest minterm is $m(15)$, a four bit input based decoder is needed. Since the minterms implement both the Boolean functions, we need a decoder with active 1 (inactive state = 0) output. Figure 4.17 shows an implementation.

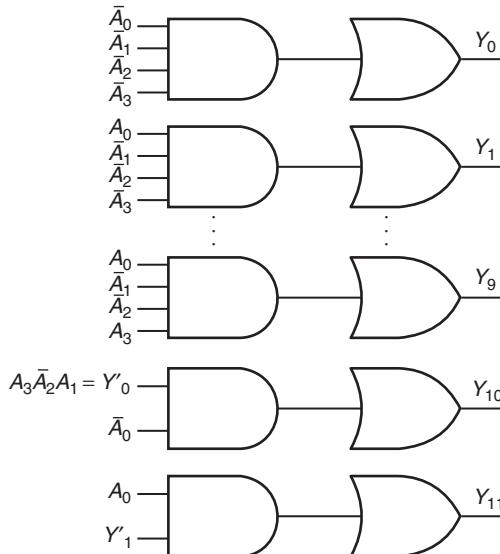


FIGURE 4.15 Implement a BCD to 12-line Decimal decoder of truth table as per Table 4.7.

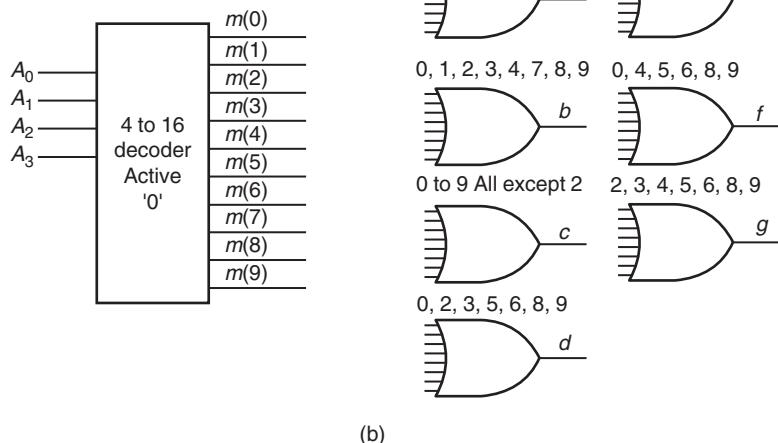
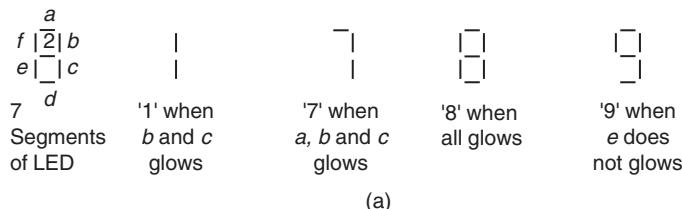


FIGURE 4.16 (a) Seven segments of an LED and how these display 0, 1, 2, up to 9 decimal digits. (b) Combinational circuit made from a decoder and the OR at the outputs of the decoder.

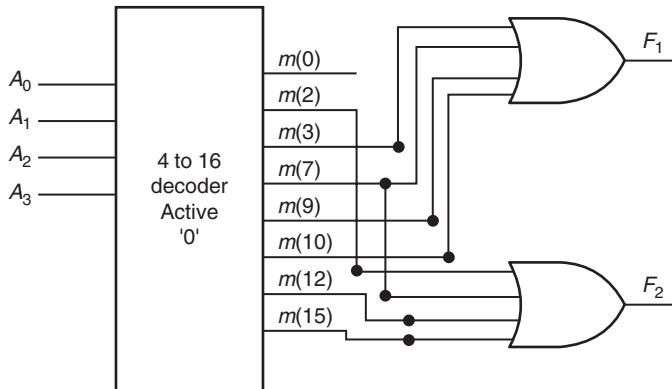


FIGURE 4.17 Implement the Boolean functions $F_1 = \Sigma m(3,7,9,10)$ and $F_2 = \Sigma m(2,7,12,15)$ by using a decoder.

Example 4.9

Logic design and implement a Boolean function $F' = \Pi M(1,7,9,13)$ by using a decoder.

Solution

Since highest minterm is $M(13)$, a four bit input based decoder is needed. Since the maxterms implement the Boolean function, we need a decoder with active 0 (inactive state = 1) output. Figure 4.18 shows the implementation.

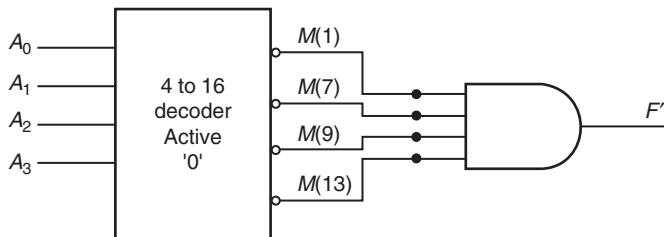


FIGURE 4.18 An implementation of a Boolean function $F' = \Pi M(1,7,9,13)$ by using a decoder.

Example 4.10

Design a circuit for a decoder with three inputs and active 0 and 1 outputs and implement the decoder circuit by NANDs alone. How will you extend the circuit for four inputs.

Solution

Since a decoder with three inputs and active 0 gives the outputs and after NOTs gives the minterms, we can implement the decoder circuit by AND-OR arrays first. Figure 4.19(a) shows this circuit. We convert the circuit into NANDs based circuit using DeMorgan theorem Figure 4.19(b) shows the circuit. We can generate in identical manner the terms $m(0)$ to $m(15)$ in case of 4 inputs.

Example 4.11

Logic design and implement a Boolean function, which Karnaugh map in Table 4.15 represents. Use a decoder.

Solution

A five-variable Karnaugh-map is given in Table 4.15

TABLE 4.15 Five variable Karnaugh Map set with 32 minterms of the SOP Expression and upper layer for $A_4 = 0$ on the left side and lower layer for $A_4 = 1$ on the right side

		$\longleftrightarrow A_4 = 0 \longrightarrow$				$\longleftrightarrow A_4 = 1 \longrightarrow$				
$\backslash A_1 A_0$		$\bar{A}_1 \bar{A}_0$ 00	$A_1 \bar{A}_0$ 01	$A_1 A_0$ 11	$A_1 \bar{A}_0$ 10	$\bar{A}_1 \bar{A}_0$ $A_3 A_2$	$\bar{A}_1 \bar{A}_0$ 00	$\bar{A}_1 A_0$ 01	$A_1 A_0$ 11	$A_1 \bar{A}_0$ 10
$\bar{A}_3 \bar{A}_2$	00	1	1	1	1	$\bar{A}_3 \bar{A}_2$ 00	1	1	1	1
$\bar{A}_3 \bar{A}_2$	01	1	1	1	1	$\bar{A}_3 \bar{A}_2$ 01	1	1		
$A_3 \bar{A}_2$	11					$A_3 \bar{A}_2$ 11				
$A_3 \bar{A}_2$	10	1	1			$A_3 \bar{A}_2$ 10	1	1	1	1

The map shows on the left side one octet with \bar{A}_4, \bar{A}_3 common. Left hand side also shows one quad $\bar{A}_4, \bar{A}_2, \bar{A}_1$. On right hand side, there are, we can redesign the map as four variable map with 2 minterms a quad for $A_4, \bar{A}_3, \bar{A}_1$ and one octet (wrapping adjacencies).

Result is $Y = \bar{A}_4 \cdot \bar{A}_3 + \bar{A}_4 \cdot \bar{A}_2 \cdot \bar{A}_1 + A_4 \cdot \bar{A}_3 \cdot \bar{A}_1 + A_4 \cdot \bar{A}_2$.

Problem is now formulated for implementation of Boolean expression for Y from prime implicants. Since a decoder with five binary inputs and output active 1 gives the outputs, which are also the implementation of the minterms, we can design logic circuit using the minterm outputs only by placing the OR gate. Figure 4.20 shows the implementation.

Example 4.12

Formulate the problem that how will you implement an 8-line to 3-line encoder of truth table as per Table 4.9. Input is active at 1 when that input line selected and is to be encoded (input when that line not selected).

Solution

Assume that output is 000 when input is $A_0 = 1$ and A_1 to $A_7 = 0$. Since there are three outputs, three Boolean Expressions Y'_0 , Y'_1 and Y'_2 are as follows.

$$Y'_0 = A_1 \cdot \bar{A}_3 \cdot \bar{A}_5 \cdot \bar{A}_7 + \bar{A}_1 \cdot A_3 \cdot \bar{A}_5 \cdot \bar{A}_7 + \bar{A}_1 \cdot \bar{A}_3 \cdot A_5 \cdot \bar{A}_7 + (\bar{A}_1 \cdot \bar{A}_3 \cdot \bar{A}_5 \cdot A_7)$$

$$Y'_1 = A_2 \cdot \bar{A}_3 \cdot \bar{A}_6 \cdot \bar{A}_7 + \bar{A}_2 \cdot A_3 \cdot \bar{A}_6 \cdot \bar{A}_7 + \bar{A}_2 \cdot \bar{A}_3 \cdot A_6 \cdot \bar{A}_7 + (\bar{A}_2 \cdot \bar{A}_3 \cdot \bar{A}_6 \cdot A_7)$$

$$Y'_2 = A_4 \cdot \bar{A}_5 \cdot \bar{A}_6 \cdot \bar{A}_7 + \bar{A}_4 \cdot A_5 \cdot \bar{A}_6 \cdot \bar{A}_7 + \bar{A}_4 \cdot \bar{A}_5 \cdot A_6 \cdot \bar{A}_7 + (\bar{A}_4 \cdot \bar{A}_5 \cdot \bar{A}_6 \cdot A_7)$$

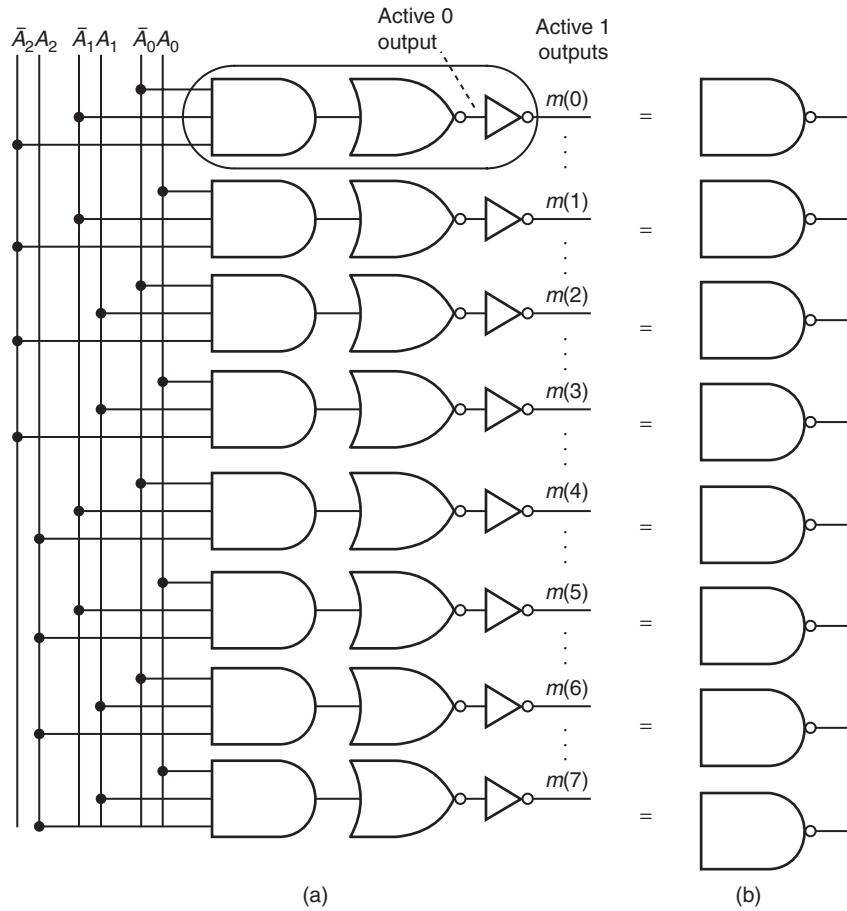


FIGURE 4.19 (a) A decoder-circuit, which implements the minterms at the outputs using AND-OR arrays (b) Circuit with NANDs alone.

Let S is 1 when at least one input is 1. It means all the inputs are not 0 or input is not invalid, it is a valid condition. Therefore,

$$S = A_0 + A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 \text{ and } Y_0 = (S.Y_0').\bar{A}_0 \\ Y_1 = (S.Y_1').\bar{A}_0 \text{ and } Y_2 = (S.Y_2').\bar{A}_0.$$

Figure 4.21 shows the logic circuit for Y_0 and S .

Example 4.13

Formulate the problem that how will you implement a 16-line to 4-line encoder of truth table as per Table 4.11. Input is active at 1 Input when that line selected and is to be encoded and active at = 0 (Input when that line not selected).

Solution

Assume that output is 0000 when input is $A_0 = 1$ and A_1 to $A_{15} = 0$. Since there are four outputs, three Boolean expressions Y_0' , Y_1' , Y_2' and Y_3' are as follows.

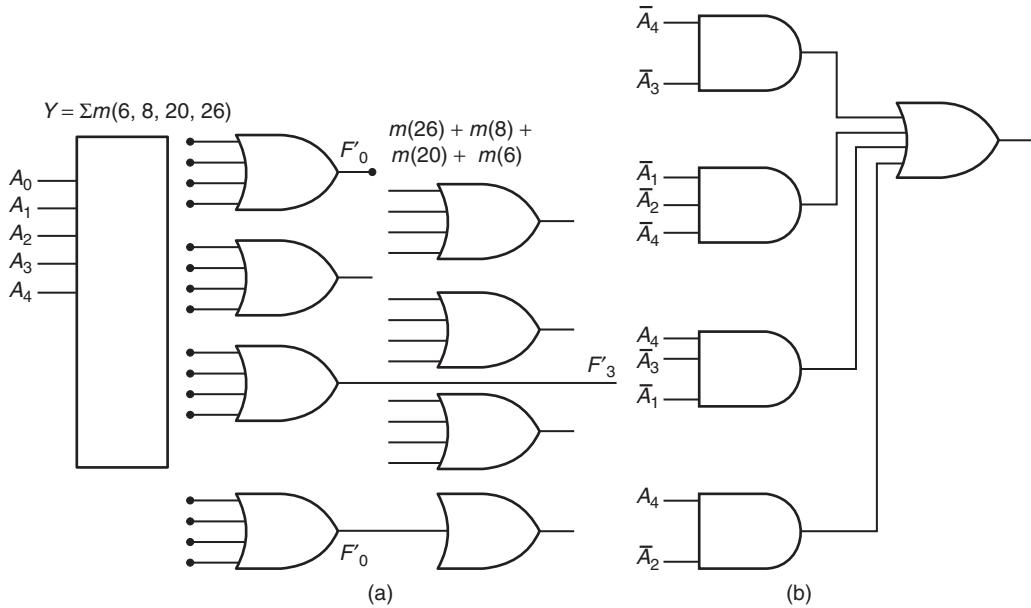


FIGURE 4.20 (a) An implementation of a Boolean function, which Karnaugh map in Table 4.14 represents using a decoder as a basic building block (b) Using three input AND-OR array.

$$\begin{aligned}
 Y'_0 &= \Sigma m(1) + m(3) + m(5) + m(7) + m(9) + m(11) + m(13) + m(15) \\
 Y'_1 &= \Sigma m(2) + m(3) + m(6) + m(7) + m(10) + m(11) + m(14) + m(15) \\
 Y'_2 &= \Sigma m(4) + m(5) + m(6) + m(7) + m(12) + m(13) + m(14) + m(15) \\
 Y'_3 &= \Sigma m(8) + m(9) + m(10) + m(11) + m(12) + m(13) + m(14) + m(15)
 \end{aligned}$$

Let \$S\$ is 1 when at least one input is 1. It means all the inputs are not 0 or input is not invalid, it is valid condition. Therefore,

$$\begin{aligned}
 S &= A_0 + A_1 + \dots + A_{14} + A_{15} \text{ and } Y_0 = (S.Y'_0).\bar{A}_0; \\
 Y_1 &= (S.Y'_1).\bar{A}_0; Y_2 = (S.Y'_2).\bar{A}_0 \text{ and } Y_3 = (S.Y'_3).\bar{A}_0;
 \end{aligned}$$

Example 4.14

Implement by logic design combinational logic circuit for truth table in inset of Figure 4.9(a).

Solution

Boolean expressions are as follows:

$F_0 = A.I_0; F_1 = A.I_1$. This circuit easily implements by two ANDs and one NOT gate.

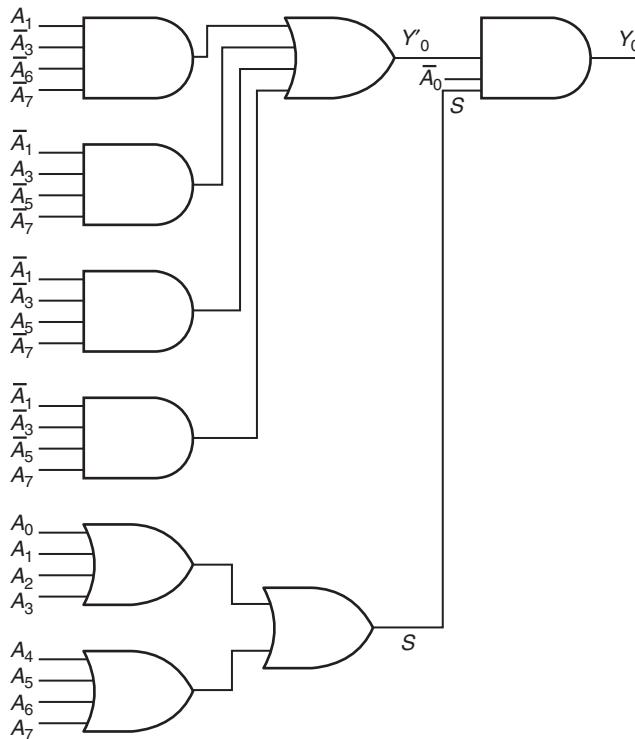


FIGURE 4.21 Implementation of an 8-line to 3-line encoder of truth table as per Table 4.9 (Only Y_0 output shown. We can draw in similar way for Y_1 and Y_2).

Example 4.15

Implement by logic design combinational logic circuit for truth table in inset of Figure 4.9(b).

Solution

Boolean expressions are as follows:

$$F_0 = \bar{A}_1 \cdot \bar{A}_2 \cdot I_0;$$

$$F_1 = A_1 \cdot \bar{A}_2 \cdot I_1;$$

$$F_2 = \bar{A}_1 \cdot A_2 \cdot I_2;$$

$$F_3 = A_1 \cdot A_2 \cdot I_3;$$

This circuit easily implements by four ANDs and two NOT gates. F is a common line in case only one is active at an instant.

Example 4.16

Draw logic design for combinational logic circuit for truth table in inset of Figure 4.9(b) with two control pins one with active 1 and other with active 0.

Solution

Figure 4.22 shows a logic design.

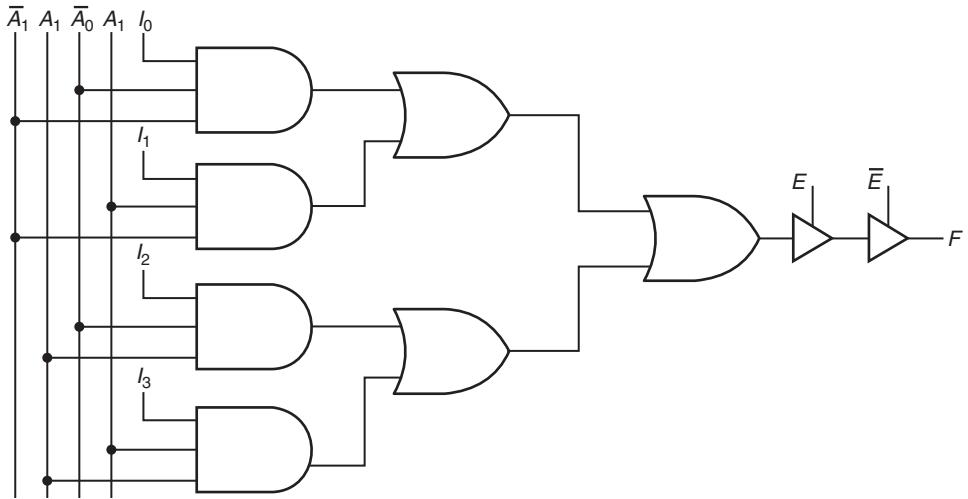


FIGURE 4.22

Example 4.17

How will we arrange in tree topology for (32 of 1) multiplexing using the nine numbers of (4 of 1) multiplexers?

Solution

Refer to the Figure 4.23, which shows the tree topology. Eight Muxs are given inputs between $I_{31}-I_0$, four each one. Then their output are given as inputs to two Muxs. At third stage, the 2 Muxs output are given to a Mux as inputs. The output of this Mux is one among I_0-I_{31} de pending on $A_4A_3A_2A_1A_0$.

Example 4.18

Give a logic design and implementation using the multiplexers for $F_1 = \Sigma m(3,7,9,10)$ and $F_2 = \Sigma m(2,7,12,15)$.

Solution

Multiplexer is Boolean expression implementer when the inputs are given at the channel selector pins and the inputs are used also as a minterm generator for F_1 give inputs X_3, X_7, X_9 and $X_{10} = 1$, other inputs as 0s. Apply the four inputs at the channel select input pins. For F_2 , make X_2, X_7, X_{12} and X_{15} as 1 and apply inputs at the channel select pins. Figure 4.24(a) shows an implementation for F_1 and F_2 .

Example 4.19

Give a logic design and implementation using the multiplexers for $F_1 = \Sigma m(3,7)$ using a 4 of 1 multiplexer.

Solution

The output F' is as follows:

$$\begin{aligned} F' = & X_0 \cdot \bar{A}_0 \cdot \bar{A}_1 \cdot \bar{A}_2 + X_1 \cdot A_0 \cdot \bar{A}_1 \cdot \bar{A}_2 + X_2 \cdot \bar{A}_0 \cdot A_1 \cdot \bar{A}_2 + X_3 \cdot A_0 \cdot A_1 \cdot \bar{A}_2 \\ & + X_4 \cdot \bar{A}_0 \cdot \bar{A}_1 \cdot A_2 + X_5 \cdot A_0 \cdot \bar{A}_1 \cdot A_2 + X_6 \cdot \bar{A}_0 \cdot A_1 \cdot A_2 + X_7 \cdot A_0 \cdot A_1 \cdot A_2; \end{aligned}$$

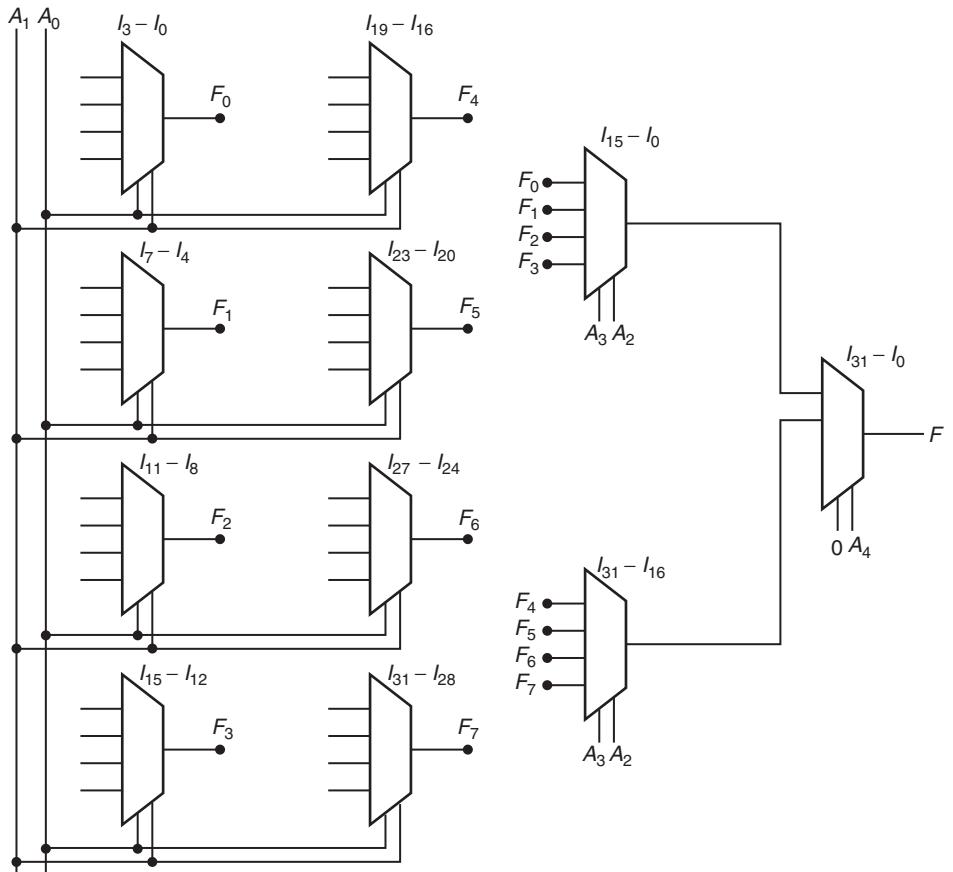


FIGURE 4.23

$$F' = \bar{A}_2 \cdot \bar{A}_1 (X_0 \cdot \bar{A}_0 + X_1 \cdot A_0) + \bar{A}_2 \cdot A_1 (X_2 \cdot \bar{A}_0 + X_3 \cdot A_0) \\ + A_2 \cdot \bar{A}_1 (X_4 \cdot \bar{A}_0 + X_5 \cdot A_0) + A_2 \cdot A_1 (X_6 \cdot \bar{A}_0 + X_7 \cdot A_0)$$

Let $F' = A_0 \cdot A_1 \cdot (I_0) + A_0 \cdot A_1 \cdot (I_1) + A_0 \cdot A_1 \cdot (I_2) + A_0 \cdot A_1 \cdot (I_3)$ in terms of inputs to a 4 of 1 multiplexer.

I_0, I_1, I_2 and I_3 are either 0 or 1 for A_2 or $\bar{A}_2 = 1$. F' can be implemented by using A_0 and A_1 channel selector inputs and result of the sum terms in the above four terms as the inputs X'_0, X'_1, X'_2 and X'_3 . When $F_1 = \Sigma m(3, 7)$, we have $X_0 = X_1 = 0$ and therefore $I_0 = 0.11$ and $I_2 = 0$ and $I_3 = 1$, and for other mux also $I_3 = 1$. Figure 4.24(b) gives the implementation circuit. I_3 and I'_3 are given inputs = $\bar{A}_2 \cdot I_3$ and $A_2 \cdot I'_3$.

Example 4.20

Give a logic design and implementation using the multiplexers for $\bar{F}_1 = \Pi M(1, 7, 9, 15)$ using a 4 of 1 multiplexer

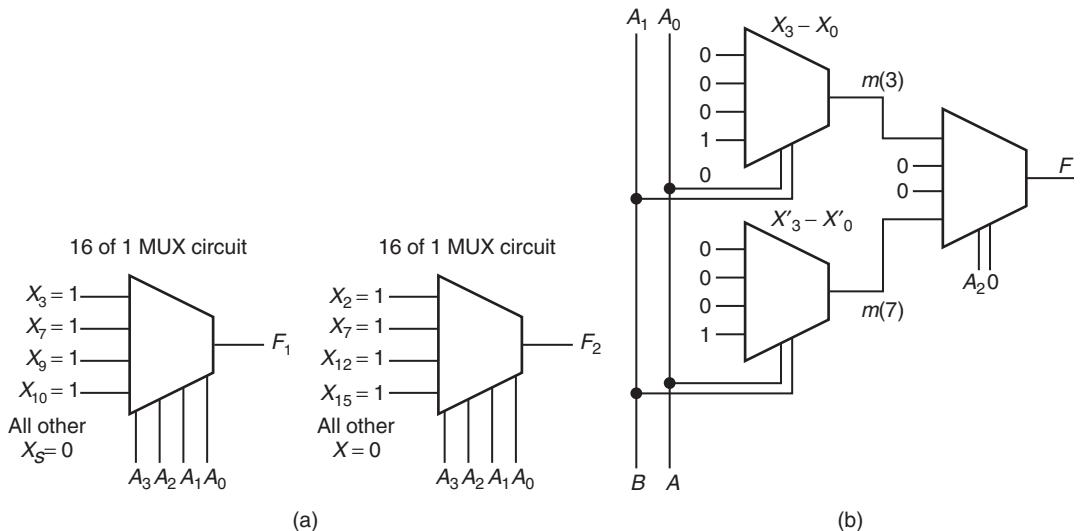


FIGURE 4.24 (a) Implementation of $F = \sum m(3,7,9,10)$ and $\sum m(2,7,12,15)$ (b) Implementation of $F_1 = \sum m(3,7)$ using a 4 of 1 multiplexer.

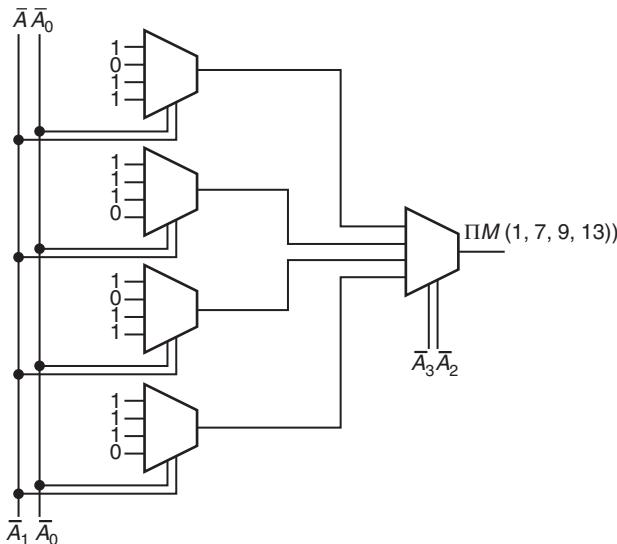


FIGURE 4.25 Implementation of Boolean expression $\Pi M(1, 7, 9, 13)$ using multiplexers.

Solution

Assume a Boolean function \bar{F}' is $\Pi M(1, 7, 9, 13)$. It means there are four literals in Boolean expression. We can implement this by a (16 of 1) multiplexer. It is equivalent to $\bar{F}' = \sum m(0, 2, 3, 4, 5, 6, 8, 10, 11, 12, 13, 14)$. We give complements of the four variables as the inputs at the four channel selector lines, A_0, A_1 and A_2 . Also give inputs 1 at the $X_0, X_2, X_3, X_4, X_5, X_6, X_8, X_{11}, X_{12}, X_{13}$, and X_{14} . Figure 4.25 gives the implementation circuit.

Example 4.21

How will you implement the design of the circuit as for logic as per truth table of a demultiplexer for sending output 1 using a 8-line F_0 to F_7 selector from three channel select bits, Assume inactive output state $z = 0$.

Solution

Consider Table 4.14. First let us ignore I (4th column). The outputs in terms of the minterms are as follows: $Y'_0 = m(0)$; $Y'_1 = m(1)$; $Y'_2 = m(2)$; $Y'_3 = m(3)$; $Y'_4 = m(4)$; $Y'_5 = m(5)$; $Y'_6 = m(6)$; $Y'_7 = m(7)$ if we assume that A_0 , A_1 and A_2 are the inputs (in place of the channel or address select bits).

Now consider I. $Y_0 = Y'_0 \cdot I$; $Y_1 = Y'_1 \cdot I$; $Y_2 = Y'_2 \cdot I$; $Y_3 = Y'_3 \cdot I$; $Y_4 = Y'_4 \cdot I$; $Y_5 = Y'_5 \cdot I$; $Y_6 = Y'_6 \cdot I$. The combinational circuit is as per Figure 4.26.

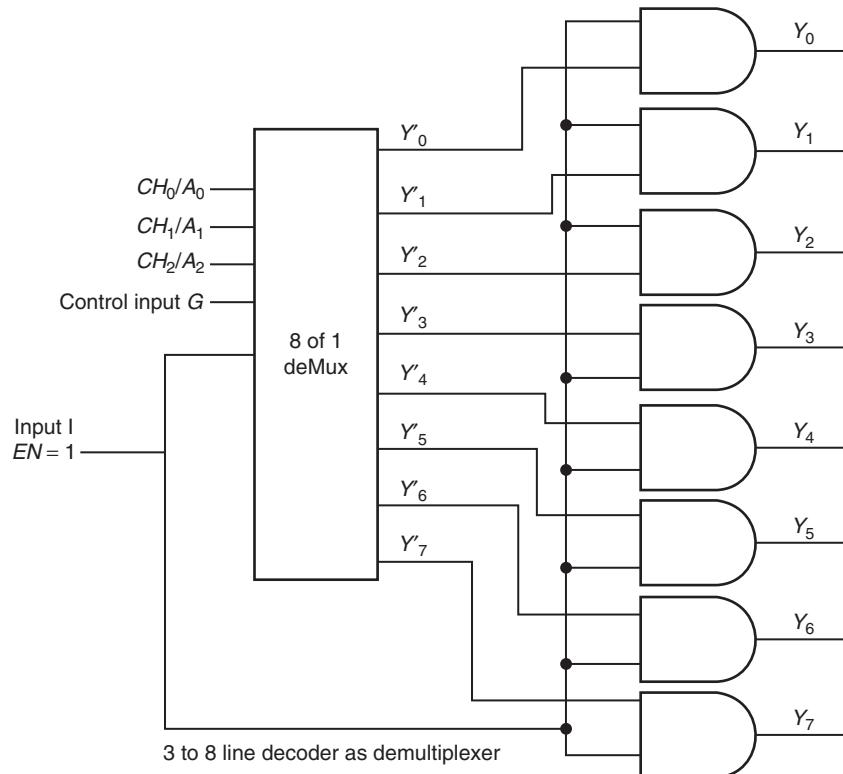


FIGURE 4.26 Combinational circuit for an 8 output demultiplexer with addressed output = input as per the channel selection bits A_0 , A_1 and A_2 . Except the selected output channel, all outputs $z = 0$. The addressed output = 1.

■ EXERCISES

1. Draw on a drawing sheet an 8-bit adder circuit using full adders as the building blocks.
2. Make the circuits for the following Boolean Expressions for a 4-stage FA circuit shown in Figure 4.2.

$$Cy_i = n_i + p_i \cdot n_{i-1} + p_i \cdot p_{i-1} \cdot n_{i-2} + p_i \cdot p_{i-1} \cdot p_{i-2} \cdot n_{i-3} + \dots + p_i \cdot p_{i-1} \cdot p_{i-2} \dots \dots \\ p_0 \cdot n_0 + p_i \cdot p_{i-1} \cdot p_{i-2} \dots \dots p_{-1} \cdot Cy_{-1}, \text{ Assume } i = 4.$$

3. Formulate the problem and how will you implement a 3-line to 8-line decoder of truth table as per Table 4.6. Output is active at 0 and inactive state is at * (tristate output when line not selected). The outputs are available only when two control gate inputs are at 0 (low) and one control gate pin is at 1 (high).
 4. How will you implement (6 of 64) decoder from 74138, a 3 to 8 decoder with three control gate pins? [Hint: Use eight number (3 to 8) decoders. Give three inputs A_0, A_1 and A_2 to A_0, A_1, A_2 of each decoder. Give remaining three inputs A_3, A_4 and A_5 at G_0, G_1 , and G_2 to each of the 8 decoders in 8 possible combinations. Only one decoder activates at a given inputs A_0, A_1, A_2, A_3, A_4 and A_5 .]
 5. How will you implement (6 of 64) decoder from 74154, a 4 to 16 decoder with two control gate pins, both active 0 inputs? [Hint: Use four number (4 to 16) decoders and control gate pins also for the additional two inputs A_4 and A_5 .]
 6. Formulate the problem and implement an 11-line to 5-line encoder (decimal to BCD priority encoder truth table as per Table 4.10. Input is active at 1 (Input when that line selected and is to be encoded) and inactive = 0 (input when that line not selected).
 7. Formulate a problem for implementing a 16 key keypad encoder.
 8. Formulate the problem that how will you implement an 8-line to 3-line encoder of truth table as per Figure 4.8(b). Input is active at 1 input when that line selected and is to be encoded) and inactive = 0 (input when that line not selected).
- [Hint: Use maxterms in Example 4.14.]
9. Using a multiplexer implement a four variable Karnaugh map of Table 4.16 circuit.

TABLE 4.16 Four variable Karnaugh Map

AB	CD	00	01	11	10
00				1	
01			1	1	
11			1		
10			1	x	

10. Using a decoder tree implement (8 to 64) decoder.
11. Design a multiplexer tree circuit with seven (4 to 1) multiplexers. How many channels are multiplexed at the root multiplexer?
12. Design a demultiplexer tree circuit with seven (1 to 4) demultiplexers. How many channels are demultiplexed at the root demultiplexer output?

13. Show how will you use a 4 to 16 channel multiplexer for a four-variable Boolean-expression.
14. Design a decoder based logic circuit for a Karnaugh map of three variables in which the even minterms are 0s.
15. Design a multiplexer based logic circuit for a Karnaugh map of three variables in which the even maxterms are 0s.

■ QUESTIONS

1. Explain a four-bit binary adder circuit.
2. Explain a 4-bit adder cum subtractor circuit, which uses the XORs as a controlled inverter.
3. Draw a circuit for a two's complement implementer using the 4-bit adder cum subtractor circuit.
4. Why is a four-bit adder circuit implemented with full adders like Figure 4.2 also called a ripple adder?
5. Why does a carry look-a-head generator give a fast adder? How much is the speed up for an 8-stage circuit?
6. Give four exemplary applications of a decoder.
7. What is a difference between a decoder and a digital demultiplexer? Explain their truth table differences by taking an example.
8. Show that decoder circuit in Figure 4.8 is also useful for generating minterms and maxterms.
9. What is the purpose of a control gate pin in a decoder?
10. What is the purpose of multiple control gate pins in a decoder?
11. What is a difference between an encoder and a decoder? Explain with an example.
12. A digital multiplexer (MUX) cannot be used as DMUX (digital demultiplexer). Why? (Hint: A digital gate is not bilateral; an analog gate can be).
13. What is a difference between an encoder and a digital multiplexer?
14. Give four exemplary applications of a multiplexer.
15. Explain the truth table differences of an encoder and a DMUX by taking an example of each.
16. What is the difference between a digital multiplexer and a digital demultiplexer? Explain with an example?
17. Implement a 4 to 1 digital multiplexer using a decoder and four tristate buffers.
18. Show a decoder circuit can be is a minterms generator.
19. Show a decoder circuit can be is a maxterms generator
20. Show a multiplexer is also a Boolean expression implementer.

This page is intentionally left blank.

CHAPTER 5

Code Converters, Comparators and Other Logic Processing Circuits

OBJECTIVE

In Chapter 4, we learnt that full adder circuits, decoders, encoders and multiplexers can also be used as the bigger building blocks for the logic design.

We will learn some important logic processing circuits, namely, code converters, digital comparator for magnitude and equality, parity generators and checkers, and bit-wise AND, OR, NOT logic processing circuits.

5.1 CODE CONVERTERS

5.1.1 Codes for Decimal Numbers

5.1.1.1 Common BCD Code

Recall the Equation (2.6) of Chapter 2.

A general formula to get the total N is again as under with only w_0 changed

$$N = y_{p \max-1} \times w_0^{p \max-1} + y_{p \max-2} \times w_0^{p \max-2} + y_{p \max-3} \times w_0^{p \max-3} + \dots + y_2 \times w_0^2 + y_1 \times w_0^1 = y_0 \times w_0^0, \quad \dots(5.1)$$

where $p = p \max-1, p \max-2, p \max-3, \dots, 2, 1$ and 0 , and $p \max$ is maximum, the number of places used in the representation and $w_0 = 2$ for the binary numbers. The y_0, y_1, y_2, \dots are the digit = 0 or 1 at the right-most, left first from that, left second from that and so on.

A general formula to get the total N for the common BCD code (called 8421 code) is again as under:

$$N = y_{p \max-1} \times w'_7 + y_{p \max-2} \times w'_6 + y_{p \max-3} \times w'_5 + \dots + y_2 \times w'_2 + y_1 \times w'_1 + y_0 \times w'_0, \quad \dots(5.2)$$

where $p = p \max-1, p \max-2, p \max-3, \dots, 2, 1$ and 0 , and $p \max$ is 8 (maximum) the number of places used in the BCD representation and $w_0 = 1; w_1 = 2; w_2 = 4; w_3 = 8; w_4 = 10 w_0; w_5 = 10 w_1; w_6 = 10 w_2; w_7 = 10 w_3$ for the binary numbers when representing BCD code. The y_0, y_1, y_2, \dots are the bit = 0 or 1 at the right-most, left first from that, left second from that and so on. The BCD code represented by above equation is also called 8421-code. At lowest nibble weights are 1, 2, 4 and 8. At next to lowest nibble weights are 10, 20, 40 and 80, and so on. Example 5.17 will show a design of 8421 circuit.

5.1.1.2 Excess-3 (XS-3) BCD Code

If instead of starting from 0000, we start from 0011, then the XS-3 code will be 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011 and 1100 for 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 for ten decimal numbers, respectively. A specialty of this code is that for each decimal number is self-complementing. Complement of 0 is 9. Similarly XS-3 code for 0 is 0011 and for 9 it is 1100. Complement of 1 is 8. Similarly XS-3 code for 0 is 0100 and for 8 it is 1011.

An adder for adding the BCD code and 0011 implements a logic circuit for the excess-3 code. Example 5.16 will show the design.

5.1.1.3 7536, 2421, and 5421 Format BCD Codes

(1) Formula to get the total N for the 7536 BCD code is as under:

$$N = y_{p \max-1} \times w'_7 + y_{p \max-2} \times w'_6 + y_{p \max-3} \times w'_5 + \dots + y_2 \times w'_2 + y_1 \times w'_1 + y_0 \times w'_0, \quad \dots(5.3)$$

where $p = p \max-1, p \max-2, p \max-3, \dots, 2, 1$ and 0 , and $p \max$ is 8 (maximum) the number of places used in the BCD representation but $w_0 = -6; w_1 = 3; w_2 = 5; w_3 = 7; w_4 = 10 w_0; w_5 = 10 w_1; w_6 = 10 w_2; w_7 = 10 w_3$ for the binary numbers when representing 7436 BCD code. The y_0, y_1, y_2, \dots are the bit = 0 or 1 at the right-most, left first from that, left second from that and so on. At lowest nibble weights are $-6, 3, 5$ and 7 . At next to lowest nibble weights are $-60, 30, 50$ and 70 , and so on.

(2) Formula to get the total N for the 2421 BCD code is again as under:

$$N = y_{p \max-1} \times w'_7 + y_{p \max-2} \times w'_6 + y_{p \max-3} \times w'_5 + \dots + y_2 \times w'_2 + y_1 \times w'_1 + y_0 \times w'_0, \quad \dots(5.4)$$

where $p = p \max-1, p \max-2, p \max-3, \dots, 2, 1$ and 0 , and $p \max$ is 8 (maximum) the number of places used in the BCD representation but $w_0 = 1; w_1 = 2; w_2 = 4; w_3 = 2; w_4 = 10 w_0; w_5 = 10 w_1; w_6 = 10 w_2; w_7 = 10 w_3$ for the binary numbers when representing 2421 BCD code. The y_0, y_1, y_2, \dots are the bit = 0 or 1 at the right-most, left first from that, left second from that and so on. The 2421-BCD code represented by above equation. At lowest nibble weights are 1, 2, 4 and 2. At next to lowest nibble weights are 10, 20, 40 and 20, and so on.

(3) Formula to get the total N for the 5421 BCD code is again as under:

$$N = y_{p \max -1} \times w'_7 + y_{p \max -2} \times w'_6 + y_{p \max -3} \times w'_5 + \dots + y_2 \times w'_2 + y_1 \times w'_1 + y_0 \times w'_0 \quad \dots(5.5)$$

where $p = p \max - 1, p \max - 2, p \max - 3, \dots, 2, 1$ and 0 , and $p \max$ is 8 (maximum) the number of places used in the BCD representation and $w_0 = 1; w_1 = 2; w_2 = 4, w_3 = 5, w_4 = 10 w_0, w_5 = 10 w_1, w_6 = 10 w_2, w_7 = 10 w_3$ for the binary numbers when representing 5421 BCD code. The y_0, y_1, y_2, \dots are the bit = 0 or 1 at the right-most, left first from that, left second from that and so on. At lowest nibble weights are 1, 2, 4 and 5. At next to lowest nibble weights are 10, 20, 40 and 50, and so on.

5.1.2 Unit Distance Code Converter

In the Equation (5.1) described above, suppose 3 increments to 4, then 8421 BCD code changes will be from 0011 to 0100. The digits are changing at $p = 0, 1$ as well as 3 (0 to 1, 1 to 0 and 0 to 1).

Assume that there are circular slots in a shaft at $0^\circ, 3^\circ, 6^\circ, 9^\circ, 12^\circ, \dots$, up to 357° . When shaft is rotating, its 8-bit position will be found from the 00000000, 00000001, 00000010, 00000011, and so on. The numbers of places where the changes occur either from 0 to 1 or from 1 to 0 are variable; 1, 2, 1 and so on.

When we convert analog signal to the digital bits, it is more reliable if for each increment or decrement step, the bit change is only at one place. Effect of some misalignment in the sensors can then be detected from the sequence of signals that is received from the rotating shaft and accounted for.

Unit distance code is a code in which next increment or decrement causes the bit-transition only at one place.

5.1.2.1 Gray Code Converter

Gray code is a unit distance code. The code is given in Table 5.1.

TABLE 5.1

Decimal value representation	Four bit binary representation variables				Four bit Gray code	Cell and minterm at the Karnaugh map of four	
	A	B	C	D	$Y_3 Y_2 Y_1 Y_0$	Corresponding Cell number	Minterm at the Karnaugh
0	0	0	0	0	0000 _{gray}	0	$m(0)$
1	0	0	0	1	0001 _{gray}	1	$m(1)$
2	0	0	1	0	0011 _{gray}	2	$m(3)$
3	0	0	1	1	0010 _{gray}	3	$m(2)$
4	0	1	0	0	0110 _{gray}	4	$m(6)$
5	0	1	0	1	0111 _{gray}	5	$m(7)$
6	0	1	1	0	0101 _{gray}	6	$m(5)$
7	0	1	1	1	0100 _{gray}	7	$m(4)$
8	1	0	0	0	1100 _{gray}	8	$m(12)$

Table 5.1 Contd.

9	1	0	0	1	1101_{gray}	9	$m(13)$
10	1	0	1	0	1111_{gray}	10	$m(15)$
11	1	0	1	1	1110_{gray}	11	$m(14)$
12	1	1	0	0	1010_{gray}	12	$m(10)$
13	1	1	0	1	1011_{gray}	13	$m(11)$
14	1	1	1	0	1001_{gray}	14	$m(9)$
15	1	1	1	1	1000_{gray}	15	$m(8)$

Four-variable Karnaugh Map with Gray codes and cell numbers shown inside the cells is as follows:

\overline{AB}	CD	\overline{CD} 00	\overline{CD} 01	CD 11	CD 10
\overline{AB}	00	0000_{Gray} 1	0001_{Gray} 2	0011_{Gray} 3	0010_{Gray} 4
$\overline{A}B$	01	0100_{Gray} 7	0101_{Gray} 6	0111_{Gray} 5	0110_{Gray} 5
AB	11	1100_{Gray} 8	1101_{Gray} 9	1111_{Gray} 10	1110_{Gray} 11
$A\overline{B}$	10	1000_{Gray} 15	1001_{Gray} 14	1011_{Gray} 13	1010_{Gray} 12

Examples 5.4 to 5.8 will explain the gray code logic circuits.

5.1.3 ASCII (American Standard Code for Information Interchange) for the Alphanumeric Characters

Examine your computer keyboard. Alphanumeric characters (Table 5.2) are *a* to *z*, *A* to *Z*, 0 to 9 and signs *, !, @, #, \$, % and so on. Examine your telephone keypad. Alphanumeric characters are 0 to 9 and signs * and #. ASCII codes are now universally used. Maximum significant bit (msb) Bit 7 is used as parity bit to check transmission error at the receiver when an ASCII code transfers as a byte on a line or network. (For international characters representation, an extension of ASCII code is nowadays used. It is called 16-bit unicode.) Example 5.18 explain the code with an example.

TABLE 5.2 Seven bits of ASCII Codes for the alphanumeric characters

ASCII Code lower nibble				Character for ASCII Code upper nibble lower 3 bits							
Y_3 0	Y_2 0	Y_1 0	Y_0 0	000	001	010	011	100	101	110	111
0	0	0	1	z	z	SP	0	@	P	'	p
0	0	1	0	z	z	!	1	A	Q	a	q
0	0	1	1	z	z	"	2	B	R	b	r
0	1	0	0	z	z	#	3	C	S	c	s
						\$	4	D	T	d	t

0	1	0	1	z	z	%	5	E	U	e	u
0	1	1	0	z	z	&	6	F	V	f	v
0	1	1	1	z	z	'	7	G	W	g	w
1	0	0	0	z	z	(8	H	X	h	x
1	0	0	1	z	z)	9	I	Y	i	y
1	0	1	0	z	z	*	:	J	Z	J	z
1	0	1	1	z	z	+	:	K	[k	{
1	1	0	0	z	z	'	<	L	\	l	
1	1	0	1	z	z	-	=	M]	m	}
1	1	1	0	z	z	.	>	N	^	n	~
1	1	1	1	z	z	/	?	O	_	o	DEL

Note: z means that it is one of the keyboard specific special codes—for example, 0001101 is ASCII code for CR (carriage return). SP in third ASCII code column means code generated when the space key of keyboard presses. ~ sign mean keyboard specific Escape character. DEL means Delete character.

5.2 EQUALITY AND MAGNITUDE COMPARATORS BETWEEN TWO FOUR-BIT NUMBERS

An important operation in the computations requires a comparison of two binary words of 8 or 16 bits and to find whether these are equal or one greater than other or vice versa. A digital comparator has many applications like during executions of while ... do... repeat ... until if ... then ... else type of computer statements. It is one of the important logical units.

A digital comparator (Examples 5.9 and 5.10) differs from a analog voltage comparator in the sense that (i) the former compare only the logic levels of one number's binary bits with that of another number's binary bits while the later compares the two potential differences each with respect to a common ground potential, and (ii) the former is made from the digital logic gates while the latter is made from an operation amplifier.

A comparator, which is not a magnitude comparator and just an equality comparator has only output terminal for $\mathbf{A} = \mathbf{B}$. It will be set to, say 1 if condition is satisfied and show complementary output if the equality condition is not satisfied. (Alternatively, it will be set to, say, 0 if condition is satisfied and show complementary output 1 if the equality condition is not satisfied). A circuit for the comparator of \mathbf{A} and \mathbf{B} four bit digital words has only one terminal to show equality condition. Figure 5.1 shows such a circuit of a digital (equality) comparator.

Another four bit digital comparator, called a magnitude comparator, means as following: Let \mathbf{A} is binary word 1101 of four bits; A_3, A_2, A_1 and A_0 , i.e. $A_0 = 1, A_1 = 0, A_2 = 1$, and $A_3 = 1$, the corresponding decimal number for is 13 (thirteen). Let \mathbf{B} the another binary word of four bits B_3, B_2, B_1 and B_0 . Let $B_0 = 0, B_1 = 1, B_2 = 1$ and $B_3 = 0$ —the corresponding decimal number for 0110, 6 (six). The comparator will compare and find if $\mathbf{A} > \mathbf{B}$. It then exhibits a logic state 1 (or alternatively, a state 0) at first output terminal designated as $\mathbf{A} > \mathbf{B}$ output. This output is complement of other two outputs.

At second output terminal designed as $\mathbf{A} = \mathbf{B}$, if $\mathbf{A} = \mathbf{B}$ then it exhibits a state 1 (or alternatively, a state 0) which is also the zero flag or equality bit.

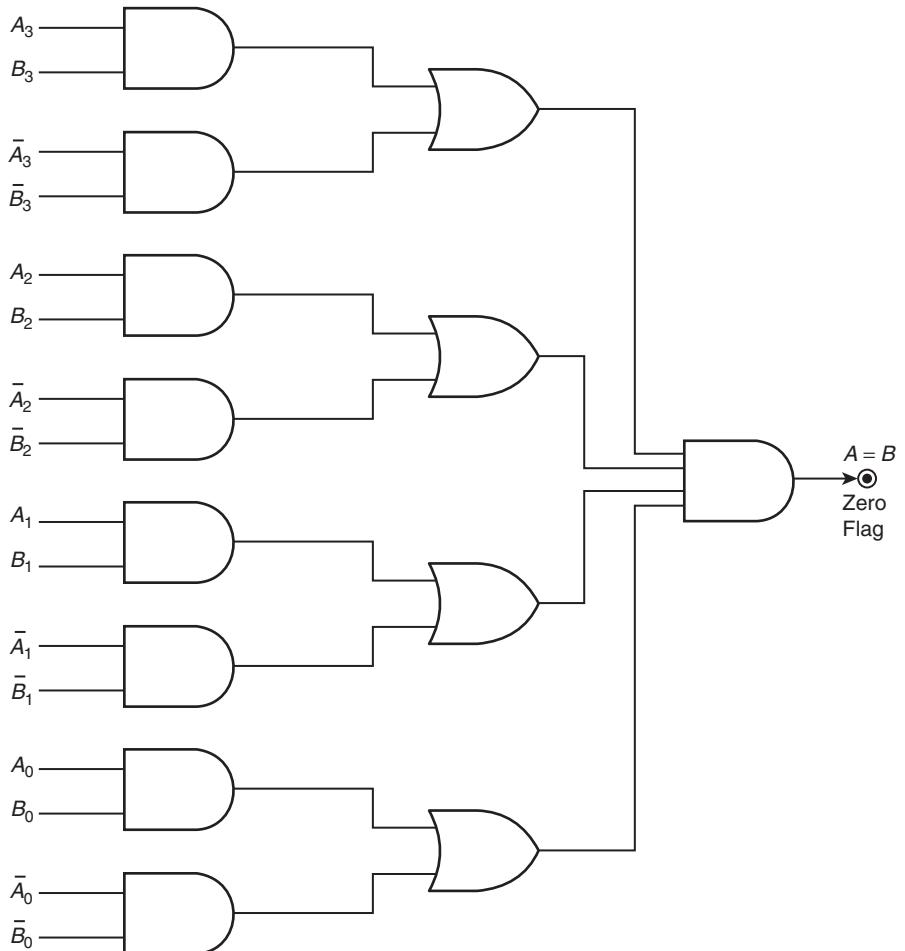


FIGURE 5.1 Digital (equality) comparator.

Complement of the first as well as third output terminals: At third $A < B$ output terminal, if $A < B$ then it exhibits a logic state 1 (or alternatively, a state 0) which is also the complement to that at the other two terminals.

MSI IC 7485 (or its HCMOS version 74HC85) is a 4-bit magnitude comparator. It also as well incorporates all the requirements shown in Figure 5.1. Its block diagram is described in Figure 5.2. An implementation example is given for the comparator in Example 5.10. It shows how to implement a four bit digital comparator by AND-OR arrays.

Let us consider a bit as a flag. If we assume that a flag sets if a condition is satisfied then we can define three flags from a digital magnitude comparator. Three results of an imaginary subtraction of B from A can be positive or negative or zero. One flag is for the positive sign and one flag is for the negative sign of the result. These 2 flags can be denoted by PSF .

A MSF or *SF* flag indicates minus sign when $A < B$. However, sometimes two flags, *ZF* and *SF*, suffice to exhibit at the outputs the four conditions, which are possible after comparison, (the imaginary subtraction). These four conditions are zero, nonzero, positive and negative.

Point to Remember

We get a one-bit result of certain arithmetic or logic operation or comparison operation. The result indicates by a bit, which is called flag. Carry, zero, sign and parity flags are the examples of the one-bit results. Comparison is a hypothetical subtraction.

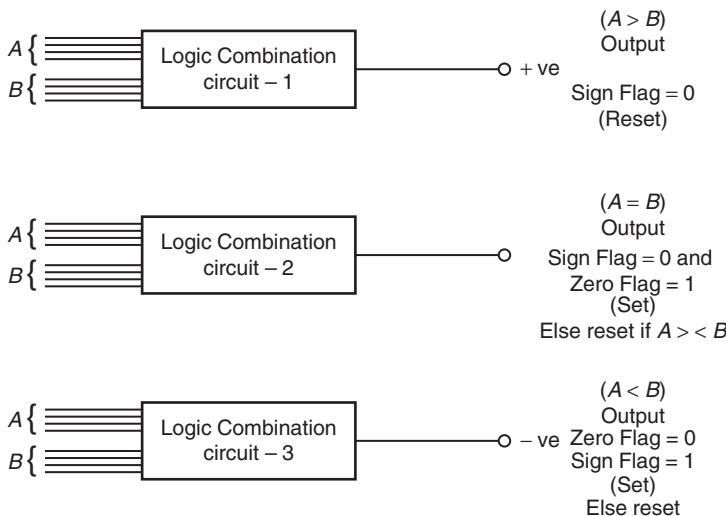
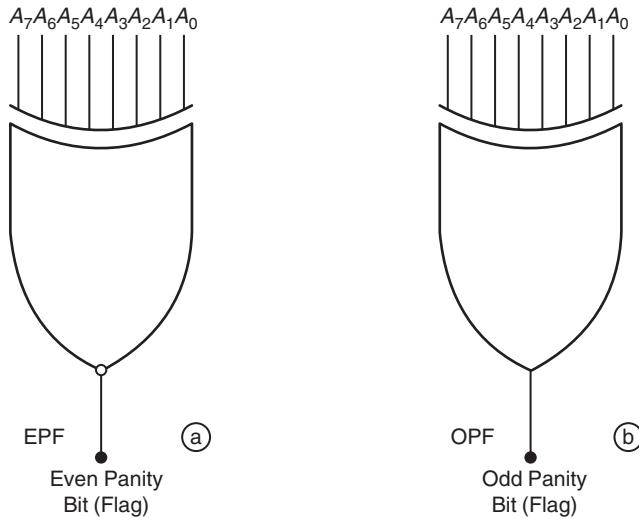


FIGURE 5.2 Block diagram of a four-bit digital magnitude comparator and results at $A < B$, $A = B$ and $A < B$ pins and at *ZF* and *SF*.

5.3 ODD PARITY AND EVEN PARITY GENERATORS

Some times the parity of a set of bit for a number is also important. Figures 5.3(a) and 5.3(b) show the even and odd parity flag (OPF and EPF) bit circuits (Refer also to Example 5.11). When the bits transmit from one source to another, the generated parity is compared at destination with the expected parity from the bits received from the source. If both are same, then it is presumed that there is no error. Parity check is successful when only one bit is in error. When two bits have error, the check is not successful. However, the chances of two errors are much smaller than the chance of a single error.

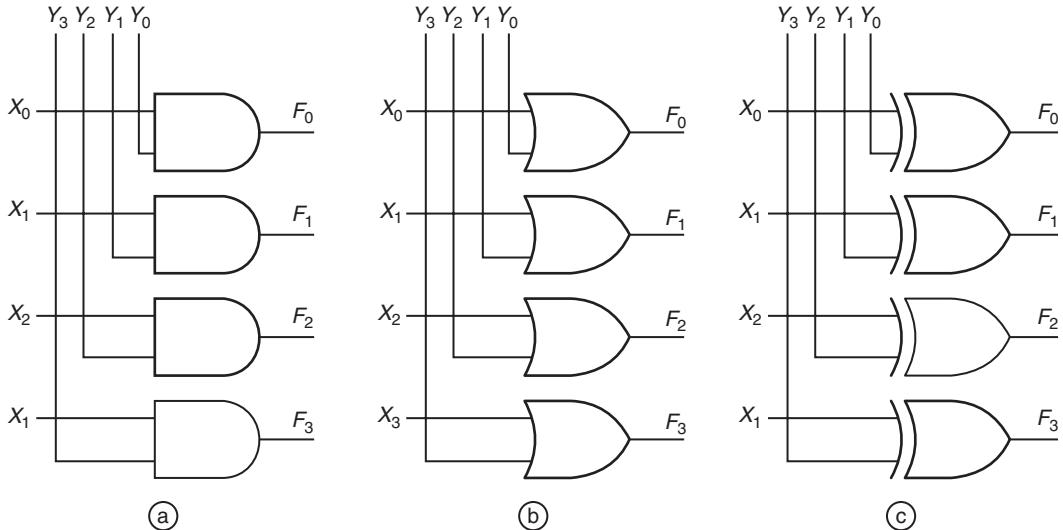


FIGURES 5.3 (a) and (b) Even and odd parity flag (OPF and EPF) bit generator.

5.4 THE 4-BIT AND, OR, XOR BETWEEN TWO WORDS

5.4.1 AND

Figures 5.4(a) shows bit wise AND between two word of four bits. $Y'_0 = A_0 \cdot B_0$; $Y'_1 = A_1 \cdot B_1$; $Y'_2 = A_2 \cdot B_2$; $Y'_3 = A_3 \cdot B_3$; $ZF = Y'_0 \cdot Y'_1 \cdot Y'_2 \cdot Y'_3$ (If bit wise AND results in all output bits = 0s, then ZF sets to 1). (Example 5.13)



FIGURES 5.4 (a) Bit wise AND between two word of four bits (b) Bit wise OR between two word of four bits (c) Bit wise XOR between two words of four bits.

5.4.2 OR

Figure 5.4(b) shows bit wise OR between two word of four bits. $Y'_0 = A_0 + B_0$; $Y'_1 = A_1 + B_1$; $Y'_2 = A_2 + B_2$; $Y'_3 = A_3 + B_3$; $ZF = Y'_0.Y'_1.Y'_2.Y'_3$ [If bit wise OR results in all output bits = 0s, then ZF sets to 1]. (Example 5.12)

5.4.3 XOR

Figure 5.4(c) shows bit wise XOR between two words of four bits. $Y_0 = A_0 \text{XOR} B_0$; $Y_1 = A_1 \text{XOR} B_1$; $Y_2 = A_2 \text{XOR} B_2$; $Y_3 = A_3 \text{XOR} B_3$; $ZF = \bar{Y}_0 \bar{Y}_1 \bar{Y}_2 \bar{Y}_3$ [If bit wise XOR results in all output bits = 0s, then ZF sets to 1. It also means that all the bits are equal $A = B$.] (Example 5.14)

5.4.4 Test

Just as a comparison is hypothetical subtraction, test is a hypothetical AND operation. Both give only flags at the outputs. $Y = \text{test flag} = Y'_0.Y'_1.Y'_2.Y'_3$, where $Y'_0 = A_0.B_0$; $Y'_1 = A_1.B_1$; $Y'_2 = A_2.B_2$; $Y'_3 = A_3.B_3$; the test is to find whether A and B .

We same use the logic processing circuits for a word of 32 bits the AND, OR and XOR with the other 32 bits word in a computer system.

■ EXAMPLES

Example 5.1 Using Equations (5.2), (5.3), (5.4) and (5.5), give the BCD format codes 8421, 7536, 2421 and 5421, respectively for the following decimal numbers in column Table 5.3.

TABLE 5.3 8421, 7536, 2421 and 5421 Codes for decimal numbers

Decimal	8421 Standard BCD	7536	2421	5421
13	0001 0011	1001 0010	0001 0011	0001 0011
5	0101	0100	0101	1000
7	0111	100	0111	1010

We verify each code and find it as per Equations (5.2) to (5.5). For example, in 7536 code, $1 \times (-6) + 0 \times 3 + 0 \times 5 + 1 \times 7 = 1$ and $0 \times (-6) + 1 \times 3 + 0 \times 5 + 0 \times 7 = 3$ in first row for first and second decimal digits.

Example 5.2 Write a general formula for 5043210- code for BCD representation called biquinary code.

Solution

Formula to get the total N for the 5421 BCD code is again as under:

$$N = Y_{p_{\max}-1} \times w'_7 + y_{p_{\max}-2} \times w'_6 + y_{p_{\max}-3} \times w'_5 + \dots + y_2 \times w'_2 + y_1 \times w'_1 + y_0 \times w'_0, \quad \dots(5.6)$$

where $p = p \max - 1, p \max - 2, p \max - 3, \dots, 2, 1$ and 0, and $p \max$ is 8 (maximum the number of places used in the BCD) representation and $w_0 = 0; w_1 = 1; w_2 = 2 w_3 = 3, w_4 = 4 w_5 = 0 w_6 = 5$ for the binary numbers when representing 5043210 BCD code. The y_0, y_1, y_2, \dots are the bit = 0 or 1 at the right-most, left first from that, left second from that, and so on. At lowest level weights are 0, 1, 2, 3, 4, 0 and 5. From the Equation (5.6) decimal digit 9 is 1010000 and decimal 0 is 01000001.

Example 5.3 Write Excess-3 (XS-3) code for 8, 9, 2 and 6.

Solution

Excess-3 code is obtained by adding 0011 binary into the standard binary form. Binary codes for 8, 9, 2 and 6 are 1000, 1001, 0010 and 0110. Therefore XS-3 codes are 1011, 1100, 0101 and 1001 after performing binary addition with 0011.

Example 5.4 Design a four bit binary (ABCD) number to Gray code $Y_3 Y_2 Y_1 Y_0$ converter.

Solution

A binary to Gray code converter design using a Karnaugh map (Table 5.4) is as follows: Y_3 ; 4-bit Gray code Y_3 and A in binary number are equal in the truth table. However, this can also be proven by Karnaugh map for the output Y_3 .

TABLE 5.4 Map to Y_3 in gray code converter

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

An octet at the map forms from the adjacent cells. Therefore

$$Y_3 = A \quad \dots(5.7)$$

Y_2 ; Boolean expression for code can be defined by first filling the Karnaugh map (Table 5.5) cells with 1s from the truth table in Table 5.1.

There are two quads between which there is offset adjacency. Therefore

$$Y_2 = \bar{A} \cdot B + A \cdot \bar{B} = A \cdot \text{XOR} \cdot B \quad \dots(5.8)$$

Y_1 ; Boolean expression for code can be defined by first filling the Karnaugh map (Table 5.6) cells with 1s from the Table 5.1.

There are two quads between which there is offset adjacency. Therefore

$$Y_1 = \bar{B} \cdot C + \bar{C} \cdot B = B \cdot \text{XOR} \cdot C \quad \dots(5.9)$$

TABLE 5.5 Map for Y_2 in gray code inverter

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

TABLE 5.6 Map for Y_1 in gray code converter

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

Y_0 : Boolean expression for a 4-bit Gray code can be defined by first filling the Karnaugh map (Table 5.7) cells with 1s from the Table 5.1.

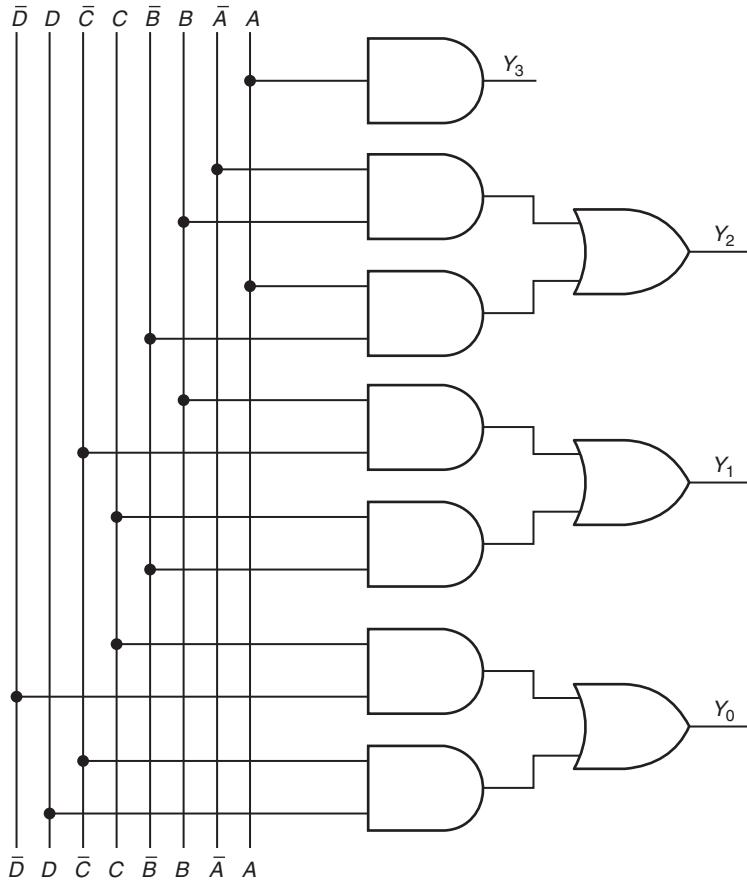
TABLE 5.7 Map for Y_0 in gray code converter

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

There are two quads between which there is offset adjacency. Therefore,

$$Y_0 = \bar{D}.C + \bar{C}.D = C.XOR.D \quad \dots(5.10)$$

Figure 5.5 gives the implementation of binary to Gray code converter using AND-OR array (equivalent to use of NANDs).



FIGURES 5.5 Implementation of the binary 4-bit number to the Gray code converter circuit.

Example 5.5

Define a simple method to convert a binary code to Gray code and show that Gray code converter can be made from adder circuit as a building block.

Solution

From the Boolean expressions in Equations (5.7) to (5.10), we note a simple method as under:

1. Upper most bit-3 of Gray code is taken same as binary number uppermost bit.
2. Lowest Gray code bit is addition without considering any carry of bit 0 and bit 1 of the binary number. (Figure 10.1)
3. Bit 1 (left to the lowest) of Gray code is addition without considering any carry of bit 1 and bit 2 of the binary number. (Figure 10.1)
4. Bit 2 (right to the uppermost) of Gray code is addition without considering any carry of bit 2 and bit 3 of the binary number. (Figure 10.1)

Example 5.6

Convert 1101 binary into Gray code using a simple method of bit addition of each bit by 1 without carry considerations (except Y_3).

Solution

1. Gray code MSB $Y_3 = 1$, because it does not change on conversion.
2. $Y_0 = \text{bit } 0 + \text{bit } 1 = 0 + 1 = 1$.
3. $Y_1 = \text{bit } 1 + \text{bit } 2 = 0 + 1 = 1$.
4. $Y_2 = \text{bit } 2 + \text{bit } 3 = 1 + 1 = 0$.

Gray code is 1011. (Answer is as per Table 5.1). We can use adder circuit in Figure 10.1 as building block.

Example 5.7

Give a simple method to convert Gray code 0011 to binary code converter.

Solution

A careful look at the Table 5.1 reveals a simple method as under: from the truth table in Table 5.1.

1. Upper most bit-3 of binary code be taken as 0 as Gray code upper most bit.
2. Lowest binary code bit 1 is added without considering any carry to 1, 0 and (bits 1, 2, 3).
3. Bit 1 (left to the lowest) of binary code is addition without considering any carry to $1 + 0 + 0$ to get $b_1 = 1$.
4. Bit 2 (right the uppermost) of binary code is addition without considering any carry of bit 2 and bit 3 $0 + 0$ to get $b_2 = 0$.

Karnaugh map based verification of the above method is left as an exercise to the reader (Exercise 7). We get the answer = 0010 as code.

Example 5.8

Convert 0111 Gray code into binary 4-bit code using the simple method of bit addition without carry considerations.

Solution

1. Binary code MSB $A = 0$, because it does not change on conversion.
 2. $D = \text{bit } 0 + \text{bit } 1 + \text{bit } 2 + \text{bit } 3 = 0 + 1 + 1 + 1 = 1$. (Add all 4 bits of Gray code).
 3. $B = \text{bit } 2 + \text{bit } 3 = 1 + 0 = 1$. (Add upper 2 bits of Gray code).
 4. $C = \text{bit } 3 + \text{bit } 2 + \text{bit } 1 = 0 + 1 + 1 = 0$. (Add upper 3 bits of Gray code).
- Binary code ABCD is 0100. (Answer is as per Table 5.1).

Example 5.9

Give truth table of a 3-bit digital comparator for six sample bit combinations for binary numbers **A** and **B**.

Solution

Result is in Table 5.8.

TABLE 5.8 A Three Digital comparator for sample bits for binary numbers **A** and **B**

Inputs						Outputs		
Input A			Input B			ZF	PSF	MSF
A_0	A_1	A_3	B_0	B_1	B_2	=	>	<
0	0	0	0	0	0	1	0	0
1	1	1	1	1	1	1	0	0
1	1	1	1	1	0	0	1	0
1	1	0	1	0	0	0	1	0
1	1	0	1	1	1	0	0	1
1	0	0	1	1	0	0	0	1

PSF bit means $A > B$ bit. MSF bit means $A < B$ bit. ZF means $A = B$ bit.

Example 5.10 Formulate the problem for implementing an 8-bit digital comparator.

Solution

Equality:

$$ZF = (\overline{A_0 \text{XOR } B_0}) \cdot (\overline{A_1 \text{XOR } B_1}) \cdots (\overline{A_7 \text{XOR } B_7})$$

An XOR circuit when both inputs are 0s or 1s gives output = 0. Hence complement after an XOR operation between each bit will give 1s only when each bit $A_0 \dots A_7$ of **A** equals $B_0 \dots B_7$ of **B**. As XOR between X and Y is $X \cdot Y + \overline{X} \cdot Y$, we get the following Boolean expression in terms of AND-OR arrays for zero sign flag output.

$ZF = (\overline{A}_0 \cdot B_0 + \overline{A}_0 \cdot \overline{B}_0) + (\overline{A}_1 \cdot B_1 + A_1 \cdot \overline{B}_1) \cdots + (\overline{A}_7 \cdot B_7 + A_7 \cdot \overline{B}_7)$. Figure 5.6(a) shows the logic circuit from AND-OR array.

Greater $A > B$:

0th stage $P_0 = A_0 \cdot \overline{B}_0$; (find whether $A_0 > B_0$) (shown at Figure 5.6(b) top).

1st stage $P_1 = A_1 \cdot \overline{B}_1 + A_1 \cdot P_0 + \overline{B}_1 \cdot P_0$; (find whether $A_1 > B_1$ or A_1 and P_0 both 1 or $B_1 = 0$ and $P_0 = 1$) (shown at Figure 5.6(b) middle).

2nd stage $P_2 = A_2 \cdot \overline{B}_2 + A_2 \cdot P_1 + \overline{B}_2 \cdot P_1$; (find whether $A_2 > B_2$ or A_2 and P_1 both 1 or $B_2 = 0$ and $P_1 = 1$).

Last stage $P_7 = PSF = A_7 \cdot B_7 + A_7 \cdot P_6 + B_7 \cdot P_6$. (Find whether $A_7 > B_7$ or A_7 and P_6 both 1 or $B_7 = 0$ and $P_6 = 1$). Figure 5.6(b) shows the logic circuit from AND-OR array (shown at Figure 5.6(b) bottom).

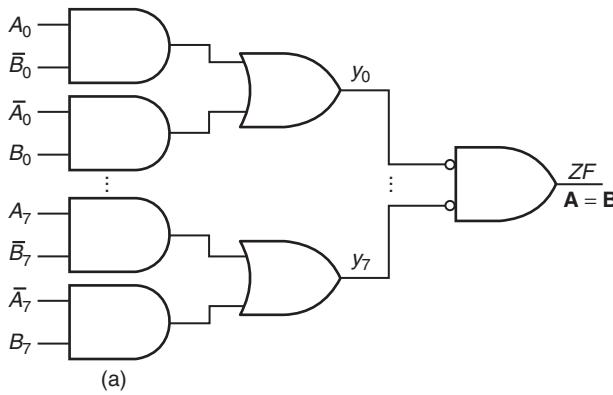
Lesser $A < B$:

0th stage $M_0 = \overline{A}_0 \cdot B_0$; (find whether $A_0 < B_0$).

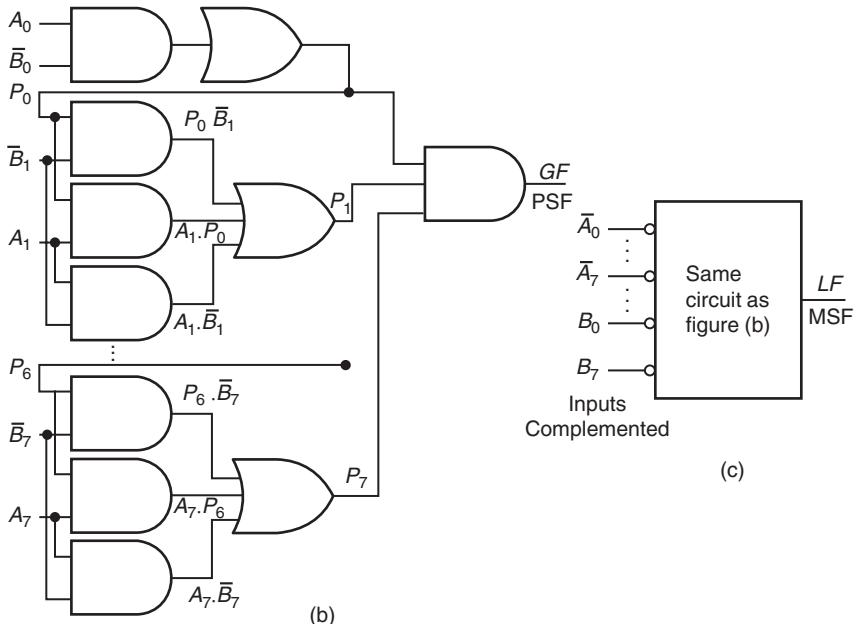
1st stage $M_1 = \overline{A}_1 \cdot B_1 + \overline{A}_1 \cdot M_0 + B_1 \cdot M_0$; (find whether $A_1 < B_1$ or B_1 and P_0 both 1 or $A_1 = 0$ and $M_0 = 1$).

2nd stage $M_1 = \overline{A}_2 \cdot B_2 + \overline{A}_2 \cdot M_1 + B_2 \cdot M_1$; (find whether $A_2 < B_2$ or B_2 and M_1 both 1 or $A_2 = 0$ and $M_1 = 1$).

Last stage $M_7 = MSF = \overline{A}_7 \cdot B_7 + \overline{A}_7 \cdot M_6 + B_7 \cdot M_6$. (Find whether $A_7 < B_7$ or B_7 and M_6 both 1 or $A_7 = 0$ and $M_6 = 1$). Figure 5.6(c) shows the logic circuit from AND-OR array (using the circuit of Figure 5.6(b) and complements of A s and B s).



(a)



(b)

(c)

FIGURES 5.6 (a) AND-OR Array for equality test (b) AND-OR Array network for positive test $A > B$ (c) AND-OR Array network for negative test $A < B$ using the \bar{A} s and B s.

Example 5.11 Implement the parity generator circuits of Figure 5.3(a) and (b) by AND arrays.

Solution

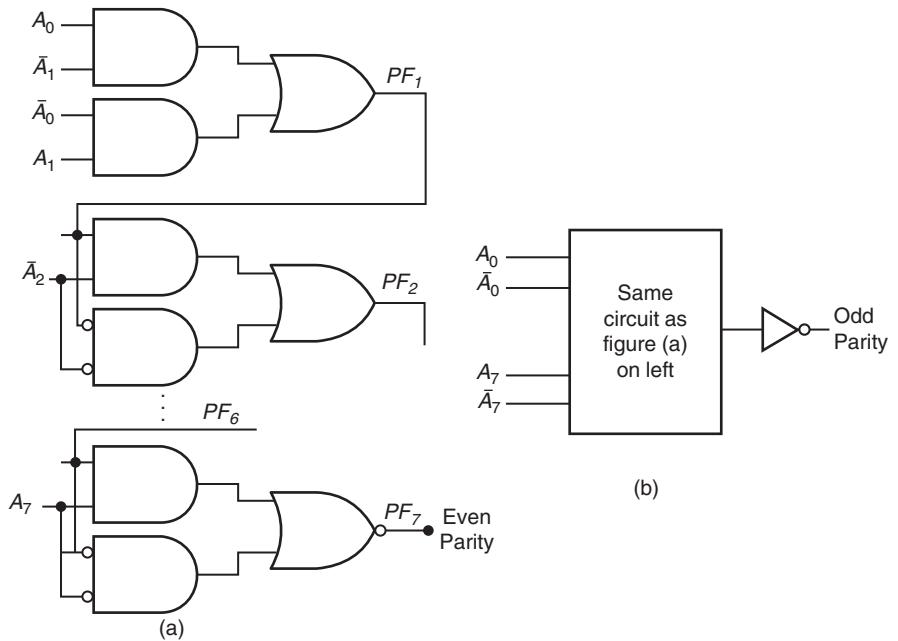
Circuits of Figures 5.3(a) and (b) are same except that there is a NOT gate at the last stage. Figure 5.7(a) implements by AND-OR arrays as follows:

$$0^{\text{th}} \text{ stage } PF_1 = (A_0 \cdot \bar{A}_1 + \bar{A}_0 \cdot A_1);$$

$$1^{\text{st}} \text{ stage } PF_2 = (\bar{A}_1 \cdot PF_1 + A_2 \cdot \bar{PF}_1);$$

$$7^{\text{th}} \text{ stage } PF_7 = (\bar{A}_7 \cdot PF_6 + A_7 \cdot \bar{PF}_6)$$

Figure 5.7 gives the circuit using AND-OR array for a parity generator.



FIGURES 5.7 Parity generator using AND-OR arrays.

Example 5.12 Find bit wise OR of 10101010 and 01010101.

Solution

$$\begin{array}{r} 10101010 \\ 01010101 \end{array}$$

Bit wise OR ‘operation gives all bits Y'_0 to $Y'_7 = 1$. Result is 11111111 and $ZF = 0$.

Example 5.13 Find bit wise AND of 10101010 and 01010101

Solution

$$\begin{array}{r} 10101010 \\ 01010101 \end{array}$$

Bit wise AND operation gives all bits Y'_0 to $Y'_7 = 0$ s. Result is 00000000 and $ZF = 1$.

Example 5.14 Find bit wise XOR of 10101010 and 01010101.

Solution

$$\begin{array}{r} 10101010 \\ 01010101 \end{array}$$

Bit wise XOR ‘operation gives all bits Y'_0 to $Y'_7 = 1$ s. Result is 11111111 and $ZF = 0$.

Example 5.15 Find bit wise NOT of 10101010.

Solution

10101010

Bit wise NOT ‘operation gives bits Y'_7 down to $Y'_0 = 01010101$.

Example 5.16 Design an Excess-3 (XS-3) code generator.

Solution

We take a 4-bit binary adders. It is given inputs **A** and **B**. **B** input pins are given 0011 as input. The output of the adder gives the XS-3 code.

Example 5.17 Design an adder for the numbers in BCD standard (8421) format.

Solution

Figure 5.8 shows the logic circuit of the BCD adder. We take two 4-bit binary adders. Both adders have 0th stage carry input = 0. One is given BCD inputs **A** and **B**. Outputs of this S_3, S_2, S_1 and S_0 are given as an input to **B** input pins of another. A_0 and A_3 pins of second adder are given 0000 as input. The output of second adder gives the lower nibble result. Least significant bit (lsb) of upper nibble (number of 10s) is obtained by following Boolean operation using four NAND gates.

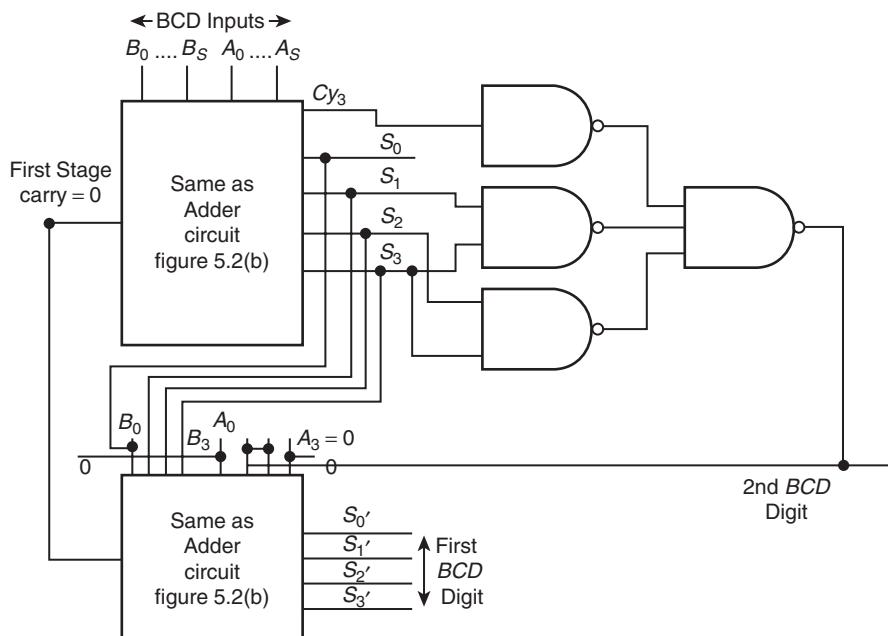


FIGURE 5.8 Logic circuit of the BCD adder using two four-bit binary adders.

Cy_3 is output last stage carry of the first adder. It is independently given to one of the NAND. S_2 and S_3 are inputs to second NAND and S_1 and S_3 are inputs to third NAND. Outputs of these three NANDs are the inputs to the fourth NAND. Fourth NAND output is the lsb of the upper BCD nibble in the result.

$$\text{lsb of Upper nibble} = (\overline{S_1 \cdot S_3} \cdot \overline{S_2 \cdot S_3} \cdot \overline{Cy_3}) = S_1 \cdot S_3 + S_2 \cdot S_3 + Cy_3$$

Example 5.18 Write ASCII codes for nineteen characters in ‘Digital Electronics’.

Solution

Using Table 5.2, we get the following answer:

100 0100; 110 1001; 110 0111; 110 1001; 111 0100; 110 0001; 110 1100; 010 0000; 100 0101; 110 1100; 110 0101; 110 0011; 111 0100; 111 0010; 110 1111; 110 1110; 110 1001; 110 0011; 111 0011.

■ EXERCISES

1. Using Equations (5.2), (5.3), (5.4) and (5.5), give the BCD format codes 8421, 7536, 2421 and 5421, respectively for the following decimal numbers in Table 5.9. Verify your answers for each column.

TABLE 5.9 8421, 7536, 2421 and 5421 Codes for decimal numbers

Decimal	8421 Standard BCD	7536	2421	5421
9				
12				
6				

2. Write 5043210 biquinary code for decimal digits 6, 9 and 12.
3. Write Excess-3 (XS-3) codes for 3 and 7.
4. The outputs from the sensors at the slots when a shaft moves in steps of 30° clockwise are Gray codes 0000; 0001; 0011; 0010; ... If the shaft moves from 0° towards anticlockwise in steps of 60° each, then what shall be the Gray code outputs during one rotation of the shaft.
(Hint: Write Gray codes for 0°, 300°, 240°, 180°, 120°, 60°, 0°).
5. Using Boolean expressions to convert the binary code to Gray codes for 1001, 0010 and 1010. Implement a Gray code converter using decoder as building block.
6. Convert 1001, 0010 and 1010 binary into Gray codes using the simple method of bit addition without carry considerations. Implement Gray code converter by using MUX.
7. Design logic circuit and develop Boolean expressions after minimization using Karnaugh map approach for Gray code to binary code converter.

8. Convert 1001, 0010 and 1010 Gray codes into binary 4-bit code using the simple method of bit addition without carry considerations.
9. Give 8 sample rows of truth table of a 4-bit digital comparator (Table 5.10) for six sample bit combinations for binary numbers A and B .

TABLE 5.10 A Four-bit digital comparator for sample bits for binary numbers A and B

Inputs						Outputs				
A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	ZF	PSF	MSF
=								>		<

10. Formulate the problem and design logic circuit for implementing a 3-bit digital comparator.
11. Implement the odd and even parity generator circuits to get eighth bit of 7-bit ASCII code implement by AND-OR arrays.
12. Find bit wise OR of 11101011 and 01010101.
13. Find bit wise AND of 11101011 and 01010101
14. Find bit wise XOR of 10101010 and 10101010.
15. Find bit wise NOT of 10101010 and then XOR the output and input bits.
16. Design an Excess-3 (XS-3) codes adder.
17. Design an 8421 standard BCD inputs adder/subtractor circuit.
18. Write the ASCII codes for characters in your college or university name.

■ QUESTIONS

1. Explain standard 8421 BCD, 7536, 2421, and 5421 BCD codes by two examples each.
2. What do you mean by self-complementing code?
3. Why is the ASCII code a widely used code? What is the 16-bit new extension of it?
4. Why do we use Gray code in certain applications?
5. Describe a digital comparator circuit.
6. If bit wise XOR the two equal binary numbers, we get resulting output number as zero. Explain it.
7. Show that bit wise XOR between a binary number and its NOT gives all bits as 1s.
8. If we bit wise NOT a number and then add 1 in it, we get two's complement. Design a circuit for finding two's complement.
9. How can we use a two input XOR as a controlled inverted of a logic state?
10. If we AND binary numbers and get the same result as the input numbers, then both numbers are said to be equal. Why?

This page is intentionally left blank.

CHAPTER 6

Sequential Logic, Latches and Flip-Flops

OBJECTIVE

Recall that a combinational circuit is a circuit made up by combining the logic gates such that the required logic at the output(s) depends only on the input logic conditions, both completely specified by either a truth table or by a Boolean expression. Also (i) An output(s) remains constant, as long input conditions do not require the change in the output(s), (ii) An output depends solely on the current input condition(s) and not on any past input condition(s) or past output condition(s), (iii) A combinational circuit has no feedback of the output from a stage to the input of either that stage or any previous stage, and (iv) An output(s) at each stage appears after a delay in of few tens or hundred ns depending upon the type or family of the gate used to implement the circuit. We have learnt the combinational circuits, their logic designs, the problems and ways to implement them by the building blocks, ICs and PLDs in other Chapters.

We will learn here another class of important logic circuits, namely sequential circuits in this and succeeding chapters. We will learn following sequential circuits in this chapter: flip-flops- *SR*, *JK*, *T*, *D*, Master Slave *FF*, triggering conditions and the characteristic equations for their analysis.

Often things are done in a sequential manner. For example, in order to prepare tea, firstly water is boiled, and then leaves are added. A sequential job means (i) to remember what steps are to be done next, and (ii) to recall which step has just been finished.

A storage device or a series of storage devices are needed in order to do the things sequentially, so that a step (or steps) that has been previously done can be recalled. The basic unit to store this information in a digital circuit is *Flip-Flop* (FF). In analog electronics, a capacitor can hold (memorize) the earlier applied potential difference (provided the charge in it does not leak due to its leakage resistance). A flip-flop is a similar basic unit in sequential circuits. A flip-flop offers a stable state (logic state not changing with time) at output even if the inputs are withdrawn (unless of course there is a power failure).

A sequential circuit is a circuit made up by combining the logic gates such that the required logic at the output(s) depends not only on the current input logic conditions but also on the past outputs (hence past inputs) and is specified by a table called state table. A state table gives the past, current and future states at the output.

1. An output(s) can remain stable (constant) even after the input conditions change,
2. An output depends on the current input state and past input states (thus past output states),
3. A sequential circuit has a feedback of the output(s) from a stage to the input of either that stage or any previous stage,
4. A sequential circuit may have a clock (gate) input to control the instance or period in which the output gets effected as per the inputs, and
5. An output(s) at each stage appears after a delay in of few ones or tens or hundred ns depending upon the type or family of the gate used to implement the circuit in case the inputs or feedbacks change.

Point to Remember

A sequential circuit is a circuit made up by combining the logic gates such that the required logic at the output(s) depends not only on the current input logic conditions but also on the past inputs, outputs and sequences.

Figure 6.1(a) shows two basic digital units in a digital circuit.

6.1 FLIP FLOP AND LATCH

Example 6.1 will give a bi-stable circuit made from the common input NANDs or NORs.

It is possible to have a stable state at an output. Flip-Flop (FF) means a digital circuit of two stable states at an output:

- 1 means, rise on top. It means flip, and
0 means fall to ground. It means flop.

The FF is a unit with 1 and 0 as stable states. A FF have two definite (discrete) states. It forms a smallest basic memory unit or a one-bit register unit. (Memory means that even if input is withdrawn, the output remains same as before.)

Figures 6.1(b) and (c) show the classifications of the FFs based on configurations and clocking mechanism, respectively.

A FF has two outputs Q and its complement \bar{Q} for a state of flip or flop. A particular combination of Q and \bar{Q} represents one of the two stable states. One of the

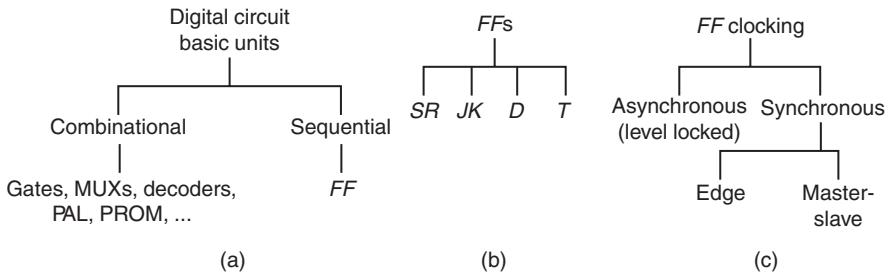


FIGURE 6.1 (a) Division of a digital circuit basic unit. (b) Flip-Flop types (c) Flip-Flop asynchronous and synchronous clocking mechanisms.

stable state is $Q = 1$ and $\bar{Q} = 0$, and other stable state is $Q = 0$ and $\bar{Q} = 1$. We may use for interconnections as well feedbacks as the inputs using the outputs, Q and/or \bar{Q} . A FF has one or two inputs. The logic states at these inputs and the previous Q determine what shall be the current output state.

A sequential circuit has a state table (like truth table in a combinational circuit). A state table of a FF describes how for the different input conditions, the output Q (and/or \bar{Q}) shall be for a given type of FF.

A FF is called latch, if the instance at which the output should change has no control input (clock falling or rising edge input). A latch is a FF without any edge triggered clocking mechanism for its inputs.

Point to Remember

A FF is a circuit, which has two stable (bi-stable) states; 1 (also called *Set*) and 0 (also called *Reset*) and in which output is stable as long as power is not withdrawn or the inputs are not applied such that the output state changes. A timing input (called clock transition or clock edge input) controls the instance at which the output changes.

A latch is a class of FF in which the instance at which output changes is uncontrolled. Uncontrolled means not controlled by the timing of the transition at clock input.

6.2 SR LATCH (SET-RESET LATCH) USING CROSS COUPLED NANDS

An SR latch is a simplest building block of a FF. It is called set-reset latch (*SR*) as it has two input states Set and Reset. It can be made by cross coupling the two NAND gates or by cross coupling the two NOR gates. The coupling of NANDs is done as shown in Figure 6.2(a). Figure 6.2(b) gives the state table for Q and \bar{Q} at the latch. Figure 6.2(c) shows the symbol for the SR latch. Q_n means n^{th} sequence before the application of the present inputs.

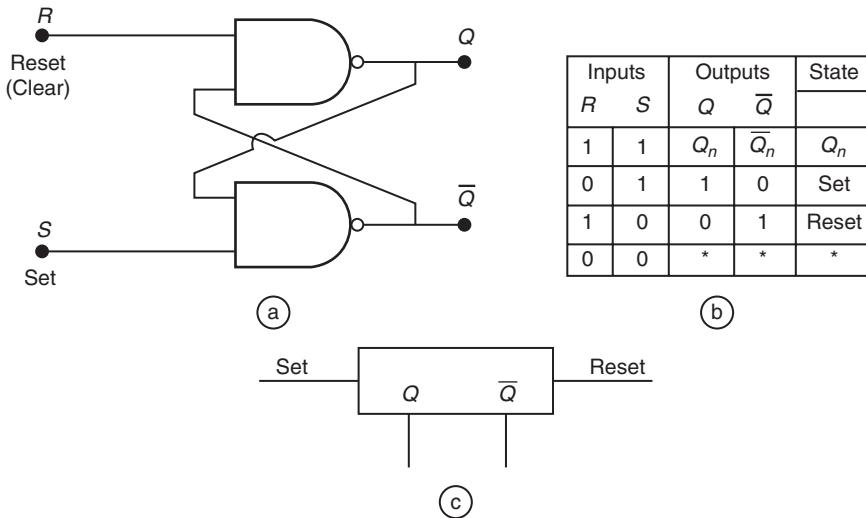


FIGURE 6.2 (a) An SR Latch made by cross-coupling two NANDs with S input to lower NAND, which has output \bar{Q}
(b) State table (c) Logic Symbol of S-R Latch.

6.2.1 SR Latch at Various Input Conditions

An SR latch have an active state for S or R as 1. An $\bar{S} \bar{R}$ latch have an active state for S or R as 0. Let us discuss SR latch.

6.2.1.1 Stable (No change) State when Both Inputs, S and R = 1

Let both inputs, (one is called S (means Set) and other R (means Reset (clear)) are 1s. [1 means that voltage levels are near to 5 V with respect to ground potential in case when TTL NANDs are used, and near to V_{DD} in case of CMOS NAND).]

If in an n^{th} state Q is 1 then \bar{Q} will be 0 due to the basic property of NAND gate, its output = 0 if both inputs (S and Q) = 1. Now, one input of upper NAND gate in Figure 6.2(a) is at 0 because it is connected to \bar{Q} (output of lower NAND gate). Other input, R , is already 1, therefore, Q will remain 1. In other words, Q will remain the complement of \bar{Q} . Here, ($Q = 1, \bar{Q} = 0$) is a stable state of the latch. The circuit of Figure 6.2(a) will remain in its stable state at the n^{th} sequence for the given input condition: both R and $S = 1$.

Now, if at n^{th} sequence $Q = 0$ and $\bar{Q} = 1$ and let R and S both are at 1s. In that case, the upper NAND gate output Q feeds a 0 input to the lower NAND gate. Therefore, \bar{Q} will remain 1. Here, $Q = 1$ and $\bar{Q} = 0$, remains stable at the latch when $R = S = 1$.

Refer first row of the table in Figure 6.2(b). When $R = 1$ and $S = 1$, then \bar{Q} will be complement of Q . It means if $Q = 1$, then $\bar{Q} = 0$, and if $Q = 0$ then $\bar{Q} = 1$. The outputs are as they were before the R and S inputs changed over to both 1s. [Note: Q_n is usually a symbol to represent a previous Q state and means there is no change at $(n + 1)^{\text{th}}$ state from the n^{th} state of a FF circuit. We will understand meanings of n and $n + 1$ stages in our discussions later.]

6.2.1.2 Set State as Stable State of the Latch when $R = 0$ and $S = 1$

If we bring an input terminal to 0 state 0 V to 0.8 V with respect to ground potential for TTL NANDs [V_{SS} to 0.33 ($V_{DD} - V_{SS}$) for CMOS NANDs], then let us assume inactivated. Let us inactivate R input logic state. Let us activate S (Set input) state. It means bring S to the logic state 1. The output Q will become, irrespective of the previous output, 1 and the latch is then said to be set. $Q = 0$ in this case. For $R = 0$ and $S = 1$, the outputs are such that Q is set and is 1 and \bar{Q} is complement of Q , and is 0. Second row of state table in Figure 6.2(b) illustrates this fact. The latch Set state means flip state. Set state inputs are $R = 0$ and $S = 1$. Set state is $Q = 1$ and $\bar{Q} = 0$.

6.2.1.3 Reset State as Another Stable State when for $R = 1$ and $S = 0$

Let us activate now R state and keep S state in the inactivated logic state. When R reset input terminal is made 1 then output Q will be reset when $S = 0$. It means will go to state 0 even if previous Q is 1. Then Q is the output of the upper NAND gate and the NAND has a property that if both inputs become 1, the output shall become 0. The third row of table in Figure 6.2(b) illustrates this fact. The Reset state is Q is 0 and $\bar{Q} = 1$. Reset state inputs are $R = 1$ and $S = 0$.

6.2.1.4 Indeterminate State when Both Inputs R and $S = 0$ (Unstable)

If both R reset and S set inputs are inactivated. It means made 0, both Q and \bar{Q} tend to be 1. The logic of the circuit is not satisfied, and the final state at an instant after the inputs R and S changed to 0, is only a matter of chance. The logic state is said to be indeterminate state or racing state. Each state, (Set $Q = 1$ and $\bar{Q} = 0$), and (Reset $Q = 0$, $\bar{Q} = 1$) trying to race through. This causes the unstable state. The fourth row of table in Figure 6.2(b) illustrates this fact.

Points to Remember

- (1) SR latch with cross-coupled NANDs or NORs has the following features:
(S – input is at that NOR which is giving Q output or S – input is at that NAND which is giving \bar{Q} output).
When $S = 0$ and $R = 0$, the output of SR latch is unstable (meta stable).
When $R = 1$ and $S = 1$, the output of SR latch does not change.
When $R = 1$ and $S = 0$, the output Q resets to 0.
When $R = 0$ and $S = 1$, the output Q sets to 1.
- (2) $\bar{S}\bar{R}$ latch with cross-coupled NANDs or NORs has the following features:
(\bar{S} – input is at the NAND giving \bar{Q} output, or \bar{S} – input is at the NOR giving \bar{Q} output)
When $\bar{S} = 1$ and $\bar{R} = 1$, the output of $\bar{S}\bar{R}$ latch is unstable (meta stable).
When $\bar{R} = 0$ and $\bar{S} = 0$, the output of $\bar{S}\bar{R}$ latch does not change.
When $\bar{R} = 0$ and $\bar{S} = 1$, the output Q resets to 0.
When $\bar{R} = 1$ and $\bar{S} = 0$, the output Q sets to 1.

6.2.2 Difficulties in Using an SR Latch

The difficulties in using an SR latch circuit shown in Figure 6.2(a) or ones described later in the Examples 6.2 to 6.4 are as follows:

1. It has an unstable condition when the input states 0 at R and S both in Figure 6.2(b).
2. When we are interested in setting (i.e. forcing in logic state 1) Q from its reset state (0) or resetting Q from the set state, there will be a certain time interval taken in interchanging the R and S input states. During the intermediate time interval (during the floating of the R and/or S inputs), what happens we cannot predict. Is it not possible that during this intermediate time interval when the input interchanges at the S and R inputs, the latch response is in an output change disabling state? Section 6.2.4 will address this question.

6.2.3 Timing Diagrams of an SR Latch

State table in Figure 6.2(b) can also be shown in terms of a timing diagram. Different input sets (for examples S , R and clock inputs) in the different time intervals are chosen and are plotted as a function of time in a diagram. The output Q or \bar{Q} and \bar{Q} both are also plotted on the same diagram. In actual practice, the change from 0 to 1 or 1 to 0 is not sharp and showing these in a timing diagram by a vertical line is just an assumption only. It is valid when the intervals instances between the input changes are longer than the transition times from 0 to 1 or 1 to 0.

Figure 6.3(a) shows how to represent an S input as a function of time, when $S = 0$ between 0 to T_1 , T_2 to T_3 , T_4 to T_5 and $S = 1$ between T_1 to T_2 and T_3 to T_4 . Figure 6.3(b) shows how to represent an R input as a function of time, when $R = 1$ between 0 to T'_1 , T'_2 to T'_3 , T'_4 to T'_5 and $R = 0$ between T'_1 to T'_2 and T'_3 to T'_4 . Figure 6.3(c) shows the timing diagram, for the outputs Q and \bar{Q} for the SR latch having state table as per Figure 6.2(b). It also shows the unstable (race condition) region during shaded area, which both R and $S = 0$ at the SR latch inputs.

1. Propagation delay, $tp(01)$ or t_{pLH} is the time interval between t' and t'' , where t' is the instance midway between 0 and 1 when an input is changing from 0 to 1 and t'' is the instance midway between 0 and 1 when an output Q is changing from 0 to 1. (Figure 6.3 right side)
2. Propagation delay, $tp(10)$ or t_{pHL} is the time interval between t''' and t'''' , where t'' is the instance midway between 1 and 0 when an input is changing from 1 to 0 and t'''' is the instance midway between 1 and 0 when an output Q is changing from 1 to 0. Average propagation delay, tp of a latch or FF is the average of (01) and $tp(10)$. (The delays $tp(01)$, $tp(10)$ and tp differs due to different impedances of the output stage transistor. These also depend on the types and family of the gates used in designing an FF .)
3. Setup time, t_s is an average of the minimum required time for an input before an enabling input (gate input or clock input) is applied so that the output Q is as per the circuit design and its state table. (Refer Section 6.2.4)

4. Hold time, t_h is an average of the minimum required time for an input to hold its logic state unchanged after an enabling input (gate input or clock input) is applied so that the output Q is as per the circuit design and its state table. (Refer Section 12.2.4)

If the inputs change during interval t_s before a clock input and t_h after the clock input, the meta stable or unpredictable state may result.

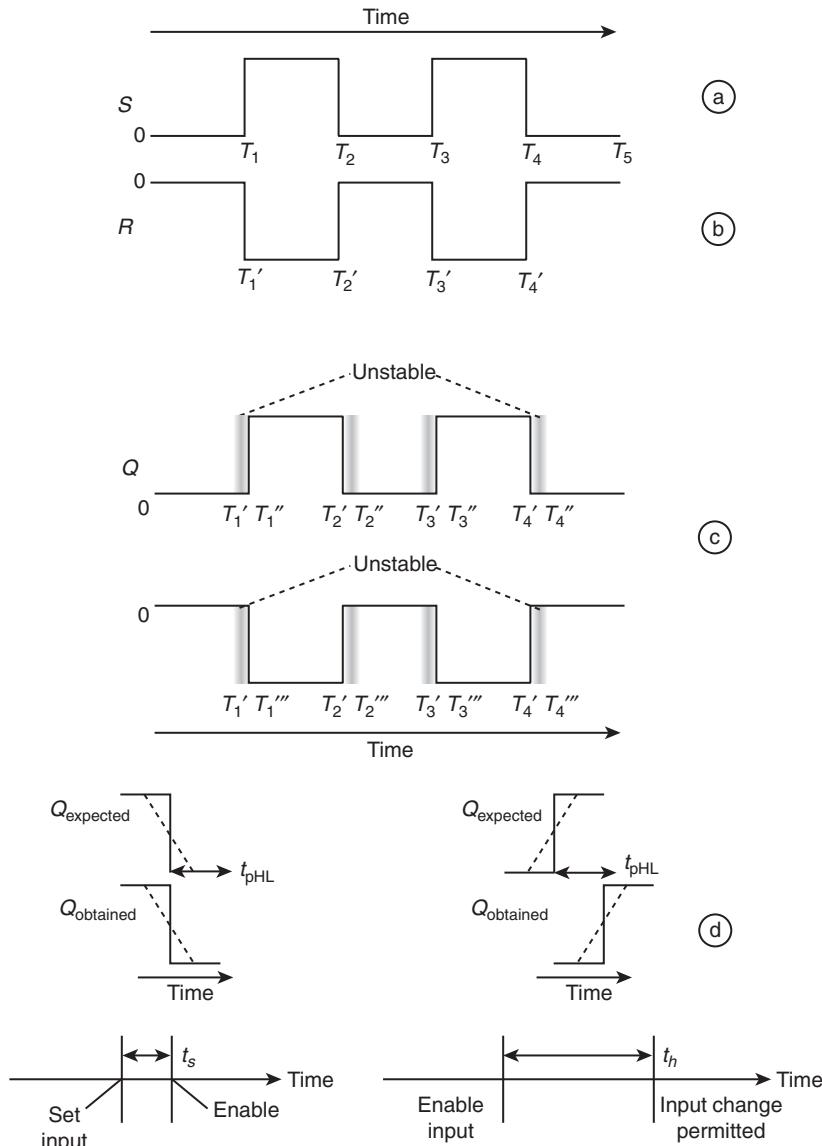


FIGURE 6.3 (a) Timing diagram of S_{input} as a function of time (b) R_{input} as a function of time (c) The timing diagram, for \bar{Q} for the SR (d) Understanding the FF propagation delay, FF setup and the holding times using the timing diagram(s).

Figure 6.3(d) marks the *FF* propagation delay, *FF* setup and the holding times in the timing plots.

6.2.4 Level Clocked SR Latch

Let us add two NAND gates to the two cross-coupled NANDs (Figure 6.4(a)). We get a clocked *SR* latch. It is explained in detail in example 6.6. It takes care of the difficulty 2 mentioned above. There is an *SR* latch circuit, which has a gating (enabling or clocking). The inputs can thus be first interchanged, and when well defined then only we can make clock input $CLK = 1$. It means open the gate to make the *S* and *R* inputs transparent in the latch. When $CLK = 0$, the *S* and *R* are having no effect on the Q s.

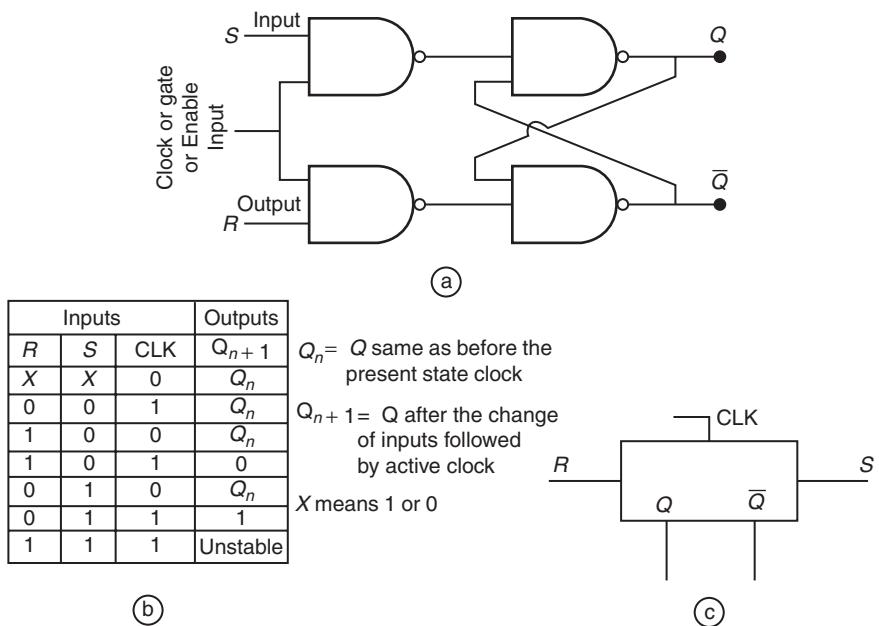


FIGURE 6.4 (a) A level-clocked SR Latch (b) State tables of clocked SR Latch (c) Symbol clocked SR Latch.

Points to Remember

SR latch with a clock input has the following features:

When $S = 1$ and $R = 1$, the output of *SR* latch is unstable (meta stable) during the active state of the clock input.

When $R = 0$ and $S = 0$, the output of *SR* latch does not change during the active or inactive state of the clock input.

When $R = 1$ and $S = 0$, the output Q resets to 0 during the active state of the clock input.

When $R = 0$ and $S = 1$, the output Q sets to 1 during the active state of the clock input.

6.3 JK FLIP-FLOP

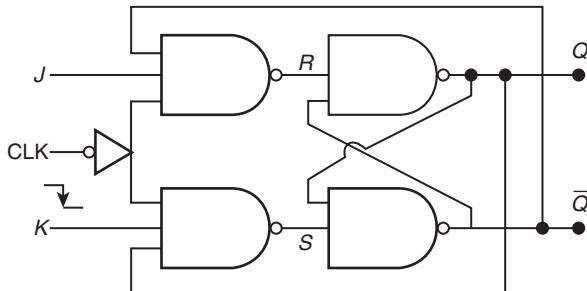
Taking three input NANDs for the inputs let us change the Figure 6.4(a) Section 6.2.4 circuit as follows:

1. The first stage NANDS S input is now labeled as J input and R input as K input.
2. Second input of both NANDs is common and is the clock input has an additional circuitry to make the J and K inputs transparent at an instance corresponding to an edge at the clock input.
3. Third input of upper NAND connects the \bar{Q} output.
4. Third input of lower NAND connects the Q output.

Figures 6.5(a) shows the modified circuit. It shows a JK flip flop, which has an edge triggered clock input so that output state change only at the instance of the edge. Figure 6.5(b) shows the state tables of positive and negative edge triggered JK flip-flops at the left and right sides, respectively. Figure 6.5(c) shows the symbols of +ve edge triggered JK (left) and -ve edge triggered (right) JK flip flops.

Unlike a transparent latch, the FF circuit given in Figure 6.5(a) responds only at an edge at the clock input. A positive edge means a transition from 0 to 1 at an input. A negative edge means a transition from 1 to 0 at an input.

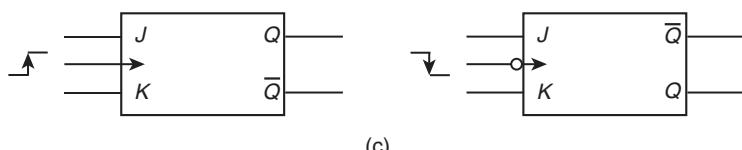
1. A bubble before a triangle at a clock input represent the fact that the clock input is negative edge triggered and output will correspond to the J and K inputs at that - ve edge instance only. At remaining instances of time, Q and \bar{Q} do not get affected ($= Q_n$ and \bar{Q}_n).



(a)

Inputs			Outputs		Inputs			Outputs	
CLK	J	K	Q	\bar{Q}	CLK	J	K	Q	\bar{Q}
↑	0	1	0	1	↓	0	1	0	1
↑	1	0	1	0	↓	1	0	1	0
↑	0	0	Q_n	\bar{Q}_n	↓	0	0	Q_n	\bar{Q}_n
↑	1	1	\bar{Q}_n	Q_n	↓	1	1	\bar{Q}_n	Q_n

(b)



(c)

FIGURE 6.5 (a) An edge triggered JK flip flop (b) State tables of positive and negative edge triggered JK flip flops (c) Symbols of + ve edge triggered JK flip flop (left) and -ve edge triggered JK flip flop (right).

2. A triangle at a clock input represent the fact that the clock input is positive edge triggered and output will correspond to the J and K inputs at that +ve edge instance only. At other instances, Q and \bar{Q} are Q_n and \bar{Q}_n .

6.3.1 Explanation of the State Table for the Logic Circuit of an Edge-Triggered JK FF

An output \bar{Q} is feedback as one of the inputs at the upper left most NAND. Another output \bar{Q} is feedback as one of the input at the lower leftmost NAND. These are the important feature of the J and K inputs flip-flops.

6.3.1.1 $J = 0$ and $K = 0$ Case

Let at the upper NAND gate the input-1 $J = 0$, the output of the first stage NAND will always be 1 whatever may be the other two inputs.

Let at the lower NAND gate the input-1 $K = 0$, the output of first stage NAND will always be 1 whatever may be the other two inputs.

The cross coupled NANDs on the right side of the circuit will have Q and \bar{Q} outputs unchanged as in the circuit of Figure 6.2(a) and first row of state table in Figure 6.2(b). There is no change at the Q output and therefore, $Q_{n+1} = Q_n$. [Q after $(n+1)^{\text{th}}$ clock edge is same as one after the n^{th} clock edge.]

TABLE 6.1 ($J = 0$ and $K = 0$) state Q and \bar{Q}

Inputs					Outputs		State
J	\bar{Q}	CLK	K	Q	Q_{n+1}	\bar{Q}_{n+1}	
0	1	X	0	0	Q_n	\bar{Q}_n	No Change
0	0	X	0	1	Q_n	Q_n	No Change

*X means any input 0 or 1 or positive edge transition 0 to 1 or negative edge transition.

Result is that the output Q remains same as before even after the clock edge when $J = 0$ and $K = 0$ and unstable condition in S-R latch case no longer exists.

Note that unstable condition for last row of state table in SR latch in Figure 6.2(b) does not exist now because all three inputs of upper and lower NANDs are never identical and never all 1s and cross-coupled both S and R NAND inputs are never 0s.

6.3.1.2 $J = 1$ and $K = 0$ Case

When $J = 1$, $K = 0$, the circuit of Figure 6.5(a) behaves as follows: Let lower NAND output $Q_n = 1$. Upper three input NAND output = 0, because two other input are also 1s, because J input is 1 and at i^{th} clock edge (+ve edge) $\text{CLK}_i J$ is 1. Upper NAND output, which is input to cross coupled NANDs = 0. Now lower NAND output, which is input to cross coupled NANDs = 1 due to $K = 0$. The cross-coupled NAND inputs therefore corresponds to $R = 0$ and $S = 1$ case in the circuit of Figure 6.2(a). Hence output Q_{n+1} will become 1 and \bar{Q}_{n+1} will become 0. If Q_n is already 0, Q_n is

already 1, the clock edge will not cause any change from the original SET state because S is already 1.

Result is that the Output Q sets to 1 after the clock edge when $J = 1$ and $K = 0$.

TABLE 6.2 ($J = 1$ and $K = 0$) state Q and \bar{Q} (+ve edge case)

Inputs					Outputs		State
J	\bar{Q}	CLK_i	K	Q	Q_{n+1}	\bar{Q}_{n+1}	$(n + 1)^{th}$
1	1	1 at the +ve edge	0	0	1	0	SET
1	1	0	0	0	Q_n	\bar{Q}_n	No Change
1	1	1	0	0	Q_n	\bar{Q}_n	No Change
1	1	0 at -ve edge	0	0	Q_n	\bar{Q}_n	No Change

6.3.1.3 $J = 0$ and $K = 1$ Case

When $J = 0$, $K = 1$, the circuit of Figure 6.5(a) behaves as follows: Let upper NAND output $Q_n = 1$. Lower three input NAND output = 0, because two other input are also 1s, because K input is 1 and at the clock edge CLK input is 1. Lower NAND output, which is input to cross coupled NANDs = 0. Now upper NAND output, which is input to cross coupled NANDs = 1 due to $J = 0$. The cross-coupled NAND inputs therefore corresponds to $S = 0$ and $R = 1$ case in the circuit of Figure 6.2(a). Hence output Q_{n+1} will become 0 and Q_{n+1} will become 1.

TABLE 6.3 ($J = 0$ and $K = 1$) state Q and \bar{Q} (+ ve edge case)

Inputs					Outputs		State
J	\bar{Q}	CLK_i	K	Q	Q_{n+1}	\bar{Q}_{n+1}	$(n + 1)^{th}$
0	0	1 at the +ve edge	1	1	0	1	RESET
0	0	0	1	1	Q_n	\bar{Q}_n	No Change
0	0	1	1	1	Q_n	\bar{Q}_n	No Change
0	0	0 at -ve edge	1	1	Q_n	\bar{Q}_n	No Change

Result is that the Output Q resets to 0 after the clock edge when $J = 0$ and $K = 1$.

6.3.1.4 $J = 1$ and $K = 1$ Case when $Q_n = 0$ and $\bar{Q}_n = 1$

The lower NAND gate output, which is input to a cross-coupled lower NAND input = 1 and $Q_n = 0$. The upper first NAND gate output, which is input to a cross-coupled upper NAND = 0 after the clock edge, because $J = 1$, CLK input at the edge = 1 and $\bar{Q}_n = 1$. Therefore, this correspond to the case of $R = 0$ and $S = 1$ in the circuit of Figure 6.2(a) for SR latch. Therefore, Q sets. Hence $Q_{n+1} = 1$ and $\bar{Q}_{n+1} = 0$. The outputs Q and \bar{Q} get reversed.

6.3.1.5 $J = 1$ and $K = 1$ Case when $Q_n = 1$ and $\bar{Q}_n = 0$

The upper NAND gate output, which is input to a cross-coupled upper NAND input = 1 because $Q_n = 0$. The lower NAND gate output, which is input to a cross-coupled lower NAND = 0 after the clock edge, because $K = 1$, CLK input at the edge = 1 and $Q_n = 1$. Therefore, this correspond to the case of $R = 1$ and $S = 0$ in the circuit of Figure 6.2(a) for SR latch. Therefore, Q sets. Hence $Q_{n+1} = 0$ and $\bar{Q}_{n+1} = 1$. The outputs Q and \bar{Q} get reversed.

The state table will now be given as follows:

TABLE 6.4 ($J = 1$ and $K = 1$) state Q and \bar{Q}

Inputs					Outputs		State
J	\bar{Q}	CLK_i	K	Q	Q_{n+1}	\bar{Q}_{n+1}	$(n + 1)^{th}$
1	0	1 at the +ve edge	1	1	0	1	Toggle $Q_{n+1} = \bar{Q}_n$
1	1	1 at the +ve edge	1	0	1	0	Toggle $Q_{n+1} = \bar{Q}_n$
1	0	-ve edge or 0 or 1	1	1	1	0	Q_n No Change
1	1	-ve edge or 0 or 1	1	1	1	0	Q_n No Change

Cases in Sections 6.3.1.4 and 6.3.1.5 shows that when $J = 1$ and $K = 1$, the output of JK flip flops toggles (changes to opposite state) on the clock edge.

A timing diagram depicts a state table of any flip-flop more clearly. Example 6.11 will describe that.

Points to Remember

JK-flip flop has the following features:

When $J = 1$ and $K = 1$, the output of JK flip flops toggles (changes to opposite state) on a clock edge.

When $J = 0$ and $K = 0$, the output of JK flip flops does not change on a clock edge.

When $J = 0$ and $K = 1$, the output Q resets to 0 after the clock edge.

When $J = 1$ and $K = 0$, the output Q sets to 1 after the clock edge.

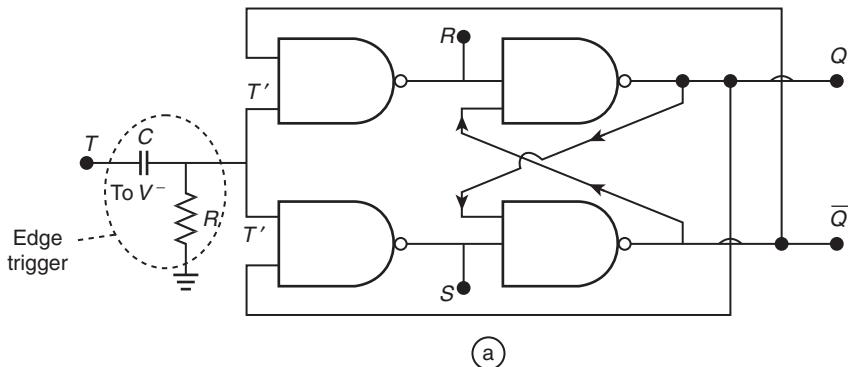
6.4 T FLIP-FLOP

Figure 6.6(a) shows a logic circuit of a T flip flop. Figure 6.6(b) shows the state table of positive edge triggered T -flip flop. Figure 6.6(c) shows the symbol of a +ve edge triggered T -flip flop. Figure 6.6(d) shows the timing diagram for the inputs at the T -input and the outputs Q and \bar{Q} in the TFF .

T stands for toggling of the state at the output. This flip flop changes on each successive input. It means gives output Q , which is complement of the previous output. It means if previous Q is 1 then it changes state to 0, and if earlier output

is 0 then it changes state to 1. This change takes place upon application of a clock transition 0 to 1 at the T input. Figure 6.6(d) shows the outputs, Q and \bar{Q} , for the given input at the terminal T of the T -flip-flop. The detailed explanation of this FF is as follows.

CASE 1: Consider an instant, when the input T becomes 1 from 0. It means instant of first [$(n + 1)^{\text{th}}$ edge] clock transition from 0 to 1. Assume that $Q = 1$ and $\bar{Q} = 0$ before this instance and after the n^{th} edge. The lower left most NAND gate in Figure 6.6(a) has both inputs 1 at that instant. (One input is a feedback from the Q and other is from T input terminal). Therefore, its output is 0. The upper left most NAND gate in this circuit having one of the input 1 but another input is 0 because \bar{Q} is 0. Therefore, this NAND output appears as 1 at R . Now, R input is, therefore, 1 to

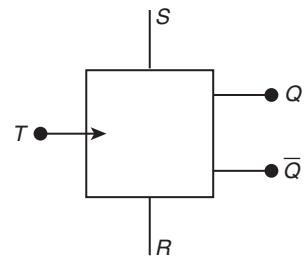


(a)

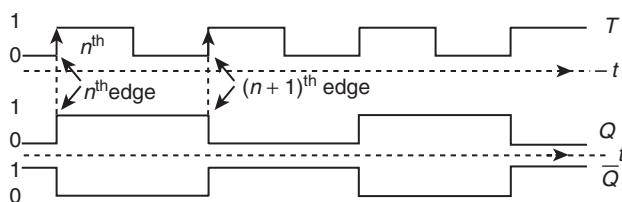
I	Outputs			
	Q_n	\bar{Q}_n	Q_{n+1}	\bar{Q}_{n+1}
1	1	0	0	1
1	0	1	1	0

 Q_n = Output before the transition at the input Q_{n+1} = Output after the transition

(b)



(c)



(d)

FIGURE 6.6 (a) Logic circuit of a T -flip flop (b) State table of positive edge triggered T -flip flop (c) Symbol of + ve edge triggered T -flip flop (d) Timing diagram for the Inputs at the T -input and the outputs Q and \bar{Q} in the T -FF.

the right side cross-coupled NAND gates (a *SR* latch). According to row 4 of the state table Figure 6.4(b), the output Q will become 0 after that instant, and the complemented \bar{Q} will become 1 after the T input becomes 1. The T is now after very short interval (equal to propagation delay, t_p) forces 0s at leftmost and right most NANDs. This causes $R = 1$, $S = 1$ at the cross-coupled NAND latch, and therefore there is no change after this interval from the edge transition. This is because there should be no change in the Q and \bar{Q} outputs as per row 1 of the state table in Figure 6.4(b). Now even if T input becomes 0, the output state will remain as before. This is due to fact that if T is '0' then both S and R inputs remain both 1, and according to row 1 of state table of cross coupled NAND, there should be no change in Q and, therefore, \bar{Q} . Table 6.5 gives the sequences followed in the $(n + 1)^{\text{th}}$ transition at the T -input.

TABLE 6.5 T FF at $(n + 1)^{\text{th}}$ edge

Inputs		Outputs		State	
\bar{Q}_n	T	Q_n	Q_{n+1}	\bar{Q}_{n+1}	
0	1 at the +ve edge	1	0	1	Toggle $Q_{n+1} = \bar{Q}_n$
1	1	0	0	1	No Change
1	-ve edge	0	0	1	No Change
1	0	0	0	1	No Change

CASE 2: Table 6.6 gives the sequences followed in the $(n + 2)^{\text{th}}$ transition at the T -input. Let us now consider the cycle after $Q = 0$, $\bar{Q} = 1$. It means at a 2nd clock edge [$(n + 2)^{\text{th}}$ edge] at the T input. At this second clock edge, when T undergoes transition to logic state 1 from 0, the upper left most NAND gate in Figure 6.6(a) has both inputs 1. So R becomes 0. The lower-most NAND has one input 0 as previous $Q = 0$ and other input is 1 (from the T). So S is 1. Now this state corresponds to 2nd row of the state table in Figure 6.2(b). Therefore, the second clock $(n + 2)^{\text{th}}$ edge causes the outputs $Q = 1$ and $\bar{Q} = 0$.

TABLE 6.6 T FF at $(n + 2)^{\text{th}}$ edge

Inputs		Outputs		State	
\bar{Q}_{n+1}	T	Q_{n+1}	Q_{n+2}	\bar{Q}_{n+2}	
1	1 at the +ve edge	0	1	0	Toggle $Q_{n+2} = \bar{Q}_{n+1}$
0	1	1	1	0	No Change
0	-ve edge	1	1	0	No Change
0	0	1	1	0	No Change

CASE 3: Table 6.7 gives the sequences followed in the $(n + 3)^{\text{th}}$ transition at the T -input. Effect of T logic state 1 is removed shortly in a time, called propagation delay, after the 2nd edge also. This happens due to the feedbacks of the new outputs. After now as well as when the T going transition to 1 from 0, the left most NAND has at least one input 0 so R is 1. The lower left most NAND has inputs so We find that if first clock transition to 1 at T input causes transition to $Q = 0$ (and $\bar{Q} = 1$), then

as S is 1. Therefore, there is no effect at the right side SR latch. The state at output remains same as previous one according to row 1 of state table in Figure 6.2(b) for the cross-coupled NANDs latch.

TABLE 6.7 T FF at $(n + 3)^{\text{th}}$ edge

Inputs			Outputs		State
\bar{Q}_{n+2}	T	Q_{n+2}	Q_{n+3}	\bar{Q}_{n+3}	
0	1 at the +ve edge	1	0	1	Toggle $Q_{n+3} = \bar{Q}_{n+2}$
0	1	0	0	1	No Change
0	-ve edge	0	0	1	No Change
0	0	0	0	1	No Change

the second clock transition to 1 from 0 at the T input causes transition to $Q = 1$ (and $\bar{Q} = 0$). Each 0 to 1 change at the T input will complement the output Q . The Q is stable to either 0 or 1 between the two successive 0 to 1 transitions.

Figure 6.6(d) gives the effect of successive pulses at the T input. If input frequency is 300 kHz, at Q the output frequency is 150 kHz. The T -flip flop is, therefore, like a counter and a frequency divider. The T -flip-flop is also called, scale-of-two circuit. T -FFs are used in the counters. A binary counter has many T -flip flops in cascade such that each T is connected to the output of the previous T -flip flop (refer Chapter 7). Normally, for the counting applications, the flip flops circuits are converted into the several T -flip flop circuits of Figure 6.6(a).

Point to Remember

T -flip flop has the following feature:

On each successive input, Q and \bar{Q} outputs toggle (complements their previous states).

6.4.1 T Flip-Flop with Clear and Preset

T -flip flop with clear and preset is a circuit in Figure 6.6(a) with R and S inputs provided to a user of the T -FF as the additional inputs. If inputs are given at the points R and/or S in Figure 6.6(a) circuit, then the circuit shall respond with a priority to the changes at these inputs. For example, if S input is activated and made 0, the output Q shall stay 1. No matter what are the states occurring at the T -input. The S input and R input are, therefore, having highest priority. Circuit of Figure 6.6(a) also acts as the SR latch if its T input is not used at all and only the R and S inputs used.

Point to Remember

T -flip flop with clear and preset is a circuit of T -FF with the following additional features:

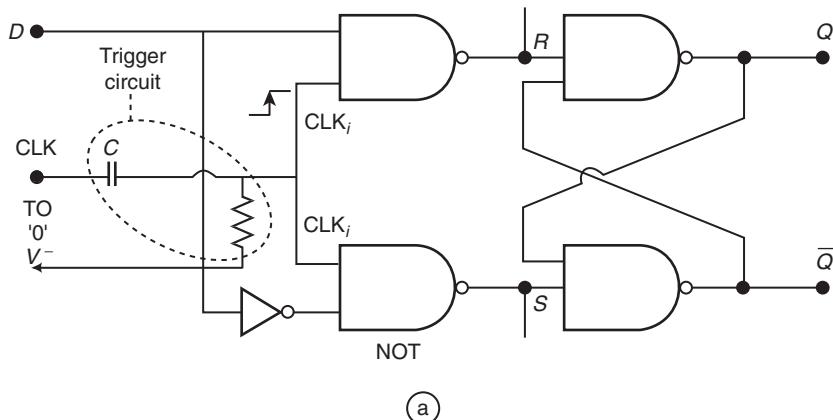
R input (active 0) is used to clear (reset) the output Q .

S input (active 0) is used to preset the output Q .

6.5 D FLIP-FLOP AND LATCH

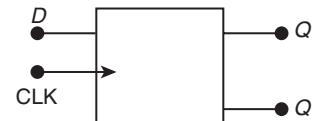
6.5.1 D Flip-Flop

Figures 6.7(a) shows a logic circuit of a *D* flip-flop. Figure 6.7(b) shows the state table of positive edge triggered *D*-flip flop. Figure 6.7(c) shows the timing diagram for the inputs at the *D*-input and the outputs *Q* and \bar{Q} in the *D*-FF. Figure 6.7(d) shows the symbol of a +ve edge triggered *D*-flip flop.



(a)

Inputs		Outputs	
<i>D</i>	CLK	\bar{Q}_{n+1}	\bar{Q}_n
1	0	0	\bar{Q}_n
1	↑	1	0
0	0	Q_n	\bar{Q}_n
0	↑	1	1

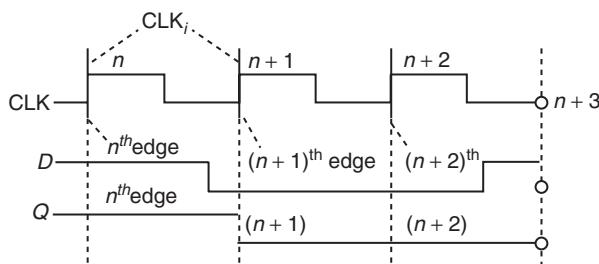


(d)

Q_n = Original state before change at CLK Input

$Q_n \equiv$ Invert of Q_n

(b)



(c)

FIGURE 6.7 (a) Logic circuit of a *D* flip-flop (b) State table of positive edge triggered *D*-flip flop (c) Timing diagram for the Inputs at the *D*-input and the outputs *Q* and \bar{Q} in the *D*-FF (d) Symbol of +ve edge triggered *D*-flip flop.

D stands for delay. Let us consider a datum bit is present at an input, called D input, just before a clock input exhibits a change to a desired logic level. From the state table in Figure 6.7(b), it can be noted that the bit is transferred to the output after the change (0 to 1 transition) at the clock input with a delay equal to propagation delay. (Remember that (i) D -has to be present before a time = setup time from the clock edge. (ii) D -has to be present after a time = hold from the clock edge.) Propagation delay is the time interval between occurrence of the clocking event and the appearance of effect at the output, Q .

From Figure 6.7(a), it can be seen that a D type FF is a modification by using additional gates in the SR cross coupled NANDs latch. When D is 1 and the clock input, CLK , is 0, the upper left most NAND gate shall give 1 in its output. Therefore, R input is 1 to the cross coupled NAND latch formed by the upper and lower NANDs on the right hand side in Figure 6.7(a). The D input after processing through a NOT gate is now 0. Therefore, lower left side NAND gates have both inputs at 0. Therefore, the S input, which is the output of this NAND is at 1. The Q output is per row 1 of state table in Figure 6.2(b) will therefore remain as such. It means unchanged.

Now, if the CLK input undergoes to 1 state from 0 state, the R input to the right side upper NAND is 0 and as the S input is 1, therefore Q output becomes 1. Therefore, Q is now identical to the state of D input just before the transition to 1 from 0 at the CLK input. Table 6.8 shows the sequences.

Similar discussion as above can be used to explain the output when $D = 0$ and $CLK = 0$ before a clock edge transition at the CLK . Q will become 0 in such a situation after the transition to 1 from 0 at the CLK input. The presence of the NOT gate shown in the left bottom corner of Figure 6.7(a) ensures that a situation, $R = 0$ and $S = 0$ is never encountered as the later situation makes an FF circuit unstable. [Recall the circuit of Figure 6.4(a) and its state table in Figure 6.4(b)]. Table 6.9 shows the sequences.

TABLE 6.8 D FF state after $(n + 1)^{th}$ transition

Inputs			Outputs		State	
D	CLK	\bar{Q}_n	Q_n	Q_{n+1}	\bar{Q}_{n+1}	
1	1 at the +ve edge	1	0	1	0	$Q_{n+1} = D = 1$
0 or 1	1	1	1	1	0	No Change
0 or 1	-ve edge	1	1	1	0	No Change
0 or 1	0	1	1	1	0	No Change

TABLE 6.9

Inputs			Outputs		State	
D	CLK	\bar{Q}_n	Q_n	Q_{n+1}	\bar{Q}_{n+1}	
0	1 at the +ve edge	0	1	0	1	$Q_{n+1} = D = 0$
1 or 0	1	1	0	0	1	No Change
1 or 0	-ve edge	1	0	0	1	No Change
1 or 0	0	1	0	0	1	No Change

Note

The clock edge at the CLK input is acting like a shutter of a camera which when clicks (here undergoes transition) causes the photograph at Q of the D input after a delay = propagation delay.

Uses of DFFs

Just as the T -flip flop is a basic unit of the counters, the D -flip flop is a basic unit of storage registers for the data (bits). Both these FF units are widely incorporated into the microprocessor, computer and other circuits.

6.5.2 D Flip-Flop with Clear and Preset

D -flip flop with clear and preset is a circuit of Figure 6.7(a) with R and S inputs also provided to a user of the D - FF as the additional inputs. If inputs are given at the points R and/ or S in Figure 6.7(a) circuit, then the circuit shall respond with a priority to the changes at these inputs. For example, if S input is activated and made 0, the output \bar{Q} shall stay 1. No matter what are the states occurring at the D -input. The S input and R input are, therefore, having highest priority. Circuit of Figure 6.7(a) also acts as the SR latch if its T input is not used at all.

Points to Remember

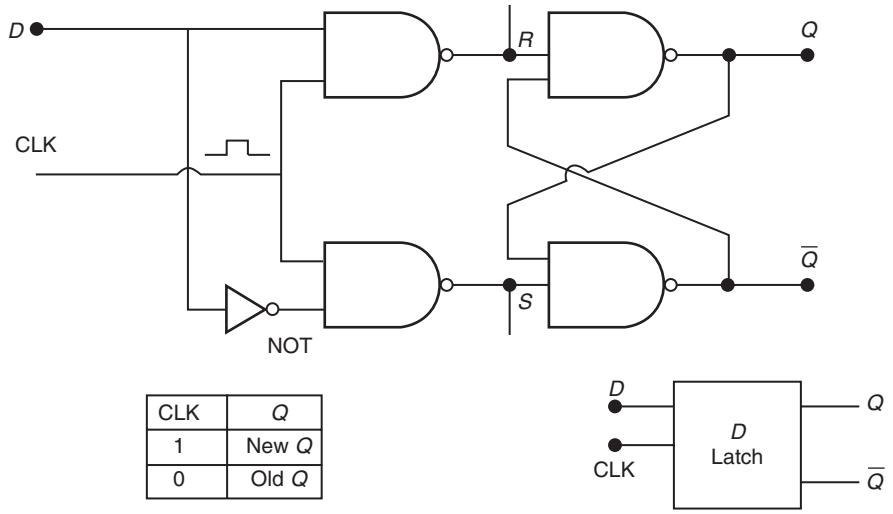
1. D -flip flop gives the output state = D input after the edge transition.
2. D -flip flop with clear and preset is a circuit of D - FF with the following additional features: R input (active 0) is used to clear (reset) the output $Q = 0$, irrespective of previous Q or D states. S input (active 0) is used to preset the output $Q = 1$ irrespective of previous Q or D states.

6.5.3 D Latch

A variation of D - FF is the D -latch shown in Figure 6.8(a). Figure 6.8(a) shows a logic circuit of a D -latch. Figure 6.8(b) shows the state table of D -latch. Figure 6.6(c) shows the symbol of a D -latch. There is no edge triggering circuit. When the clock is at 1, D -input is transferred to Q output like in a transparent latch, and during this state 1 of clock if D input changes then Q output also changes. The circuit is transparent to the input D . When the clock is at logic state 0, the Q output is frozen (not transparent) to the state of D -input prior (how much before, it depends on the setup time) of the 1 to 0 (negative edge) transition. In other words, the D -type latch provides unhindered transfer of input to the output during the period when the clock input remains at 1.

The clock input is acting like a valve which when open (here it means clocking logic level) causes the liquid to flow through a pipe.

Table 6.10 shows the sequences.



New Q = Same after Δt as D .
 Old Q = Same as just before last negative edge.

FIGURE 6.8 (a) Logic Circuit of a D -latch (b) State table of D -latch (c) Symbol of D Latch.

TABLE 6.10 D latch

Inputs			Outputs		State
D	CLK	\bar{Q}_n Q_n	Q_{n+1}	\bar{Q}_{n+1}	
0	1 at the +ve edge	0 1	$Q_{n+1} = 1$	$\bar{Q}_{n+1} = 0$	$\bar{Q}_{n+1} = \bar{Q}_n = 1$
0	1	0 1	0	1	$\bar{Q}_{n+1} = D = 0$
1	Before the -ve edge	1 0	1	0	$Q_{n+1} = D = 1$
1 or 0	At the -ve edge or at 0	1 0	0	1	No Change

6.6 MASTER-SLAVE RS FLIP-FLOP

Figure 6.9(a) shows a logic circuit called master-slave RS flip flop. Figure 6.9(b) shows the state table for master-slave RS flip flop. Figure 6.9(c) shows logic symbol of RS MS-FF. Figure 6.9(d) gives a partial timing diagram. It shows the plots for the pulse clocking input, the periods of master enabling and slave enabling and Q and \bar{Q} .

Why is the circuit called master-slave RS flip flop will be clear shortly. It is also called pulse triggered RS flip flop due to its actions on positive going transition as well as on negative going transition when a clock pulse is applied to it [Refer Figure 6.9(d)].

This flip flop is a combination of the two gated- SR latching circuits. This can be easily visualized by comparing the left and right units in Figure 6.9(a) with the circuit in the Figures 6.4(a).

CLK = 0 – Disabling Master Section

The outputs of the attached gated SR NANDs are 1. For the cross-coupled NANDs (Figure 6.2(a)), the output from a box marked MT (called master) in Figure 6.9(a), will therefore remain same as before. This follows from first row of state table given at Figure 6.2(b). The master gives intermediate outputs, Q' and \bar{Q}' . The outputs of the master are unchanged, therefore, the Q and \bar{Q} , the outputs of box marked SL (called slave) also remain unchanged as SL gets the inputs from the master section.

CLK becomes 1 – Master Section Response

Let us now assume that clock input becomes at logic state 1. Master outputs Q' and \bar{Q}' will now change according to the S and R inputs. This is analogous to action of a transparent latch in Figure 6.4(a). If $S = 0$ and $R = 1$, then the output according to table in Figure 6.4(b) shall be 0. In other words, Q' and \bar{Q}' , the master outputs, shall be 0 and 1, respectively.

The Q and \bar{Q} , the outputs of box marked SL (called slave) also remain unchanged as during $T' - T'$ the CLK' input to SL section is 0 due to the NOT gate before it.

We find that the first gated NAND flip-flop section and the box marked ML is providing the outputs Q' and \bar{Q}' according to S and R , when CLK becomes 1. This is also the reason that the MT section is called Master section. Slave is inactive during the transparency of ML because SL has T' CLK input = 0. When master finishes the action, then only slave can act.

CLK becomes 0 – Slave Section Enabled Master disabled

When Q' is 0 and \bar{Q}' is 1, this means CLR is 0 and PR is 1. Therefore, according to row 2 of table in Figure 6.10(b), the outputs $Q = 0$ and $\bar{Q} = 1$. Had S been 1 and R been 0, the outputs, upon clock going to 1 would have been according to row 1 of the table as $Q = 1$ and $\bar{Q} = 0$.

We find that the second gated NAND flip flop section and the box marked SL is simply providing the outputs Q and \bar{Q} on CLK becoming 0 according to Q' and \bar{Q}' as $CLK' = 1$. The Q' and \bar{Q}' are the outputs of the master section when CLK logic state was 1. This is also the reason that the SL section is called slave section.

The RS flip-flop action arises of the combination of master and slave.

Remember

There is no possibility of a false transition at $(n + 1)^{\text{th}}$ state on the positive going clock pulse (0 to 1) and followed negative going 1 to 0 in a *SR MS FF*. After positive going from 0 to level 1 transition, the master is triggered and the slave is idle. After the negative going from 1 to 0, the slave responds as per master outputs and the master is idle.

When both S and $R = 1$, the circuit is meta stable and outputs are unpredictable.

When both S and $R = 0$ the previous outputs remain unchanged.

When $S = 1$ and $R = 0$ the $Q = 1$.

When $S = 0$ and $R = 1$ the $Q = 0$.

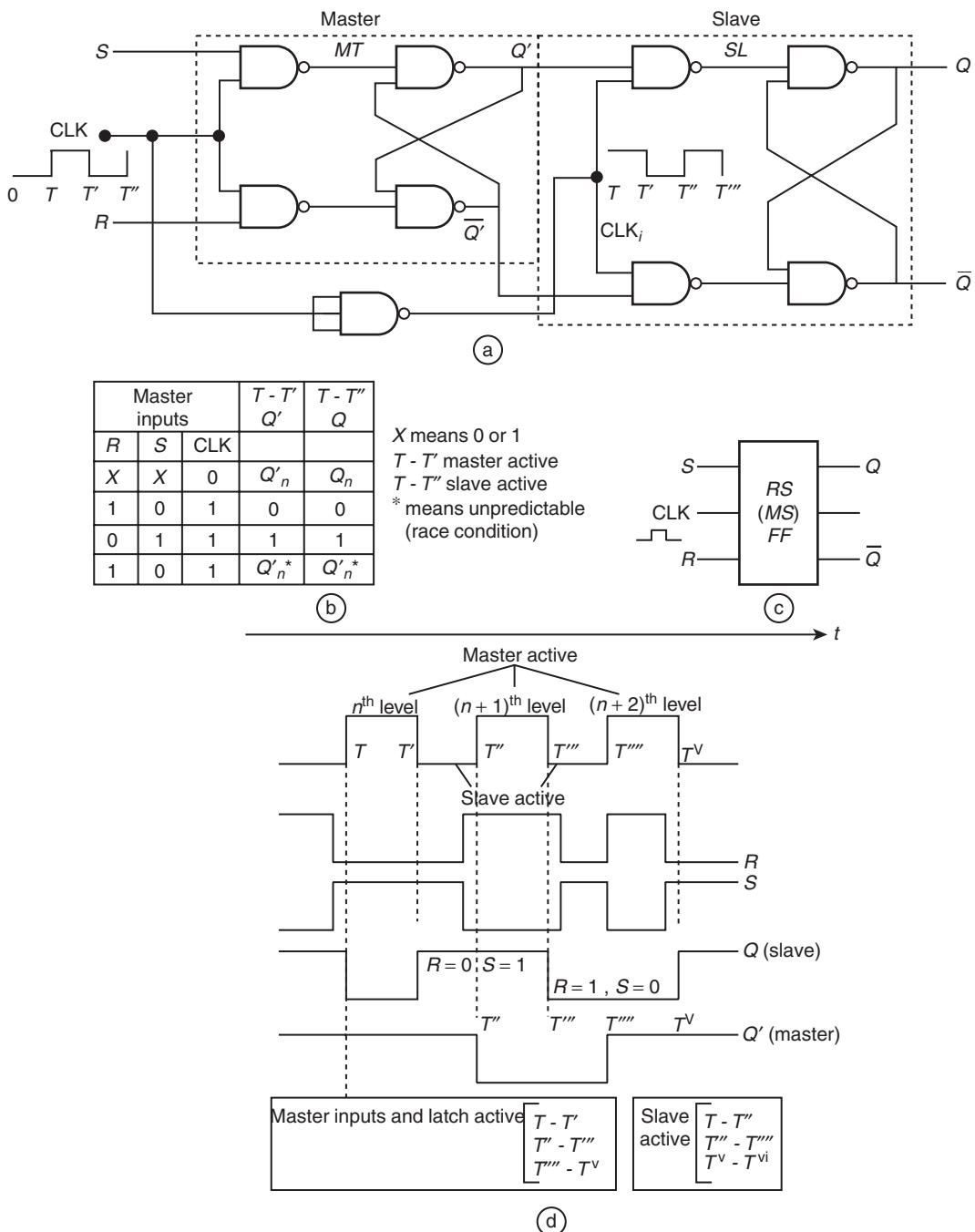


FIGURE 6.9 (a) Master-Slave SR Flip Flop logic diagram (b) State table for Master-Slave RS Flip Flop (c) Logic Symbol of SR MS-FF (d) Timing diagram for the pulse clocking input and periods of master enabling and slave enabling.

6.7 MASTER-SLAVE (PULSE TRIGGERED) JK FLIP-FLOP

R-S flip flop has an indeterminate meta-stable state [Refer state tables at the Figures 6.2(b), 6.4(b) and 6.9(b)]. An additional circuitry can avoid this state. For example, Figure 6.10(a) shows one such circuit.

The inputs are now called *J* and *K* instead of *S* and *R*, respectively. The circuit is called master-slave *JK* flip flop (*JK MS FF*) due to reason, which will be clear shortly. It is also called pulse triggered *JK*-flip flop also for the reasons being described shortly. This circuit will give a predictable determined output even when both inputs are at the state 1.

Figure 6.10(b) gives the state table of a *JK MS FF*, and Figure 6.10(c) shows its symbol. This flip flop is a combination of the gated *SR* and *T* inputs latching circuits. This can be easily visualized by comparing the left and right units in Figure 6.10(a) with two circuits in the Figures 6.4(a) and 6.7(a), respectively.

***J-K FF* Circuit Activation when *PR* and *CLR* = 1 (Inactive and Unconnected)**

It means when these two priority inputs are inactivated, then the outputs shall be according to the last four rows of the state table in Figure 6.10(b) and the outputs now depend only on *J*, *K* and *CLK* inputs. In this case, when only one either *J* or *K* is 0, the action is like the *SR* flip flop after the clock level transition 0 to 1. However, if both *J* and *K* are 0, the outputs after the clock level transition 1 to 0 shall be same as before the clock input change. When both *J* and *K* are 1, the previous output is inverted (complemented). This is unlike the *RS* latch or flip flop in which an output would have been unstable. This fact differentiates, the *JK*-FF from the *RS*-FF as pointed before.

***PR* and *CLR* = 1 (Inactive and Unconnected), and *CLK* = 0**

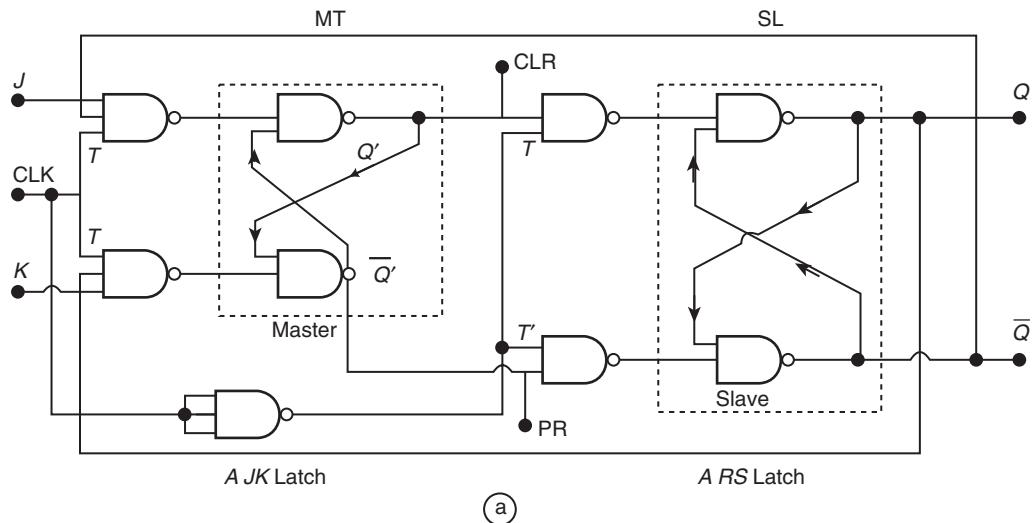
The outputs of the attached gated *SR* NANDs are 1. For the cross-coupled NANDs (Figure 6.2(a)), the output from a box marked *MT* (called master) in Figure 6.10(a), will therefore remain same as before. This follows from first row of state table given at figure 6.2(b). The master gives intermediate outputs, Q' and \bar{Q}' . The outputs of the master are unchanged, therefore, the Q and \bar{Q} , the outputs of box marked *SL* (called slave) also remain unchanged.

CLK becomes 1 – Master Section Response

Let us now assume that clock input becomes at logic state 1. master outputs Q' and \bar{Q}' will now change according to the *J* and *K* inputs. This is analogous to action of a transparent latch in Figure 6.4(a). If *J* = 0 and *K* = 1, then the output according to table in Figure 6.4(b) shall be 0. In other words, Q' and \bar{Q}' , the master outputs, shall be 0 and 1, respectively.

Here, the *J* input (in the master section of Figure 6.10(a) is equivalent to the *S* input in Figure 6.4(a), and *K* input in this *MT* section is equivalent to the *R* input in Figure 6.4(a).

The Q and \bar{Q} , the outputs of box marked *SL* (called slave) also remain unchanged as *T'* CLK input to *SL* section is 0.



Inputs					Outputs	
PR	CLR	J	K	CLK	Q	\bar{Q}
L	H	X	X	X	H	L
H	L	X	X	X	L	H
L	L	X	X	X	Q^*	\bar{Q}^*
H	H	L	L	—	Q_n	\bar{Q}_n
H	H	H	L	—	H	L
H	H	L	H	—	L	H
H	H	H	H	—	\bar{Q}_n	Q_n

* = Unstable

— = Low to high and high to low

Q_n = Pre-state before the clock input change

\bar{Q}_n = Complement of previous state Q_n

$H = 1$ and $L = 0$

X = Immaterial H or L

(b)

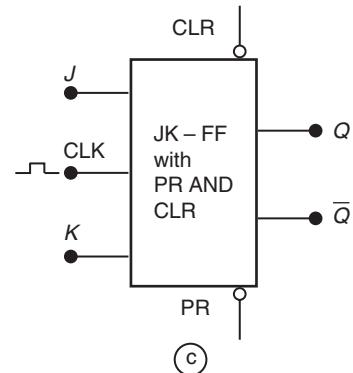


FIGURE 6.10 (a) Master-Slave JK flip flop logic diagram (b) State table for master-slave JK flip flop (c) Logic Symbol of JK FF with preset and clear inputs also present.

We find that the first gated NAND flip flop section and the box marked *ML* is providing the outputs Q' and \bar{Q}' according to J and \bar{K} , when CLK becomes 1. This is also the reason that the *ML* section is called Master section. Slave is inactive during the transparency of *ML* because *SL* has T' CLK input = 0. When the master finishes the action, then only the slave can act.

CLK becomes 0 – Slave Section Response

When Q' is 0 and \bar{Q}' is 1, this means CLR is 0 and PR is 1. Therefore, according to the second row of table in Figure 06.10(b), the outputs $Q = 0$ and $\bar{Q} = 1$. Had J been '1' and K been 0, the outputs, upon clock going to 1 would have been according to row 1 of the table as $Q = 1$ and $\bar{Q} = 0$

We find that the second gated NAND flip flop section and the box marked SL is simply providing the outputs Q and \bar{Q} on CLK becoming 0 according to Q' and \bar{Q}' . The latter are the outputs of master section when CLK logic state was 1. This is also the reason that the SL section is called Slave section. The present JK flip flop action arises of the combination of master and slave.

Points to Remember

There is no possibility of a false transition at $(n + 1)^{\text{th}}$ state on the positive going clock pulse (0 to 1) and followed negative going (1 to 0) in a JK MS-FF. After positive going from 0 to level 1 transition, the master is triggered and the slave is idle. After the negative going from 1 to 0, the slave responds as per master outputs and the master is idle. When both J and $K = 1$, the circuit is such that upon a clock pulse, the complements of the previous output states are obtained at Q and \bar{Q} .

Both J and $K = 0$ the previous outputs remain unchanged.

When $J = 1$ and $K = 0$ the $Q = 1$.

When $J = 0$ and $K = 1$ the $Q = 0$.

Notes

1. Using master-slave we implement equivalent of JK edge triggered FF by using two latches, one as a master and one as a slave.
2. When the triggering of a FF is after the positive and negative edges, it is called pulse triggering or pulse clocking.

6.7.1 MS JK Flip-Flop with Clear and Preset

Following features can be noted from the logic circuit at Figure 6.10(a) and state table in Figure 6.10(b). There are two inputs, called PR (preset) and CLR (Clear). These are like set and reset inputs in state table for the Set-Reset latch shown in Figure 6.2(b). When making one of these 0 activates any of these two inputs, the outputs Q and \bar{Q} are there, and then the J , K and CLK inputs are immaterial. In words, the CLR and PR have the higher priorities than the J , K and CLK.

Points to Remember

JK MS flip flop with clear and preset is a circuit of JK FF with the following additional features:

R input (active 0) is used to clear (reset) the output $Q = 0$.

S input (active 0) is used to preset the output $Q = 1$.

6.8 CLOCK INPUTS

6.8.1 Level Clocking of a Clock Input

Recall the circuit of Figure 6.4(a), called a level clocking circuit. Whenever CLK becomes 1 then only the circuit is transparent (responds) to the R and S inputs.

Level clocking circuit has the following difficulties:

- (i) Often the input state or states are not permitted to change at the clock state \uparrow or at the clock state \downarrow or at the clock state 1. All changes at the inputs S and R must be over when the clock input is in logic state 0.
- (ii) Assume that level 1 activates the clock input. Time interval during which $CLK = 1$ and CLK undergoes 1 to 0 transition is wasted as in any way the input has already been defined at the instance when CLK became 1.

6.8.2 Edge Triggering at a Clock Input

Incorporating an edge triggering circuit at a clock or gate input or change enable input avoids the difficulties (i) and (ii) pointed in Section 6.8.1. Therefore, a FF, which is a circuit with an additional edge triggering facility at the gate (clock) input [for example, circuits of figures 6.6(a) and 6.7(b)] is free from these difficulties.

Figures 6.11(a) and (b) show the edge triggering clock actions.

Figures 6.6 and 6.7 showed a capacitor cum resistor circuit at the clock input to obtain the edge triggering. The refined edge triggering circuits for +ve edged clock and -ve edge clocked circuits are used in an actual integrated circuit chip. A monostable circuit can be used to get positive and negative going edges, and pulses using a push button switch.

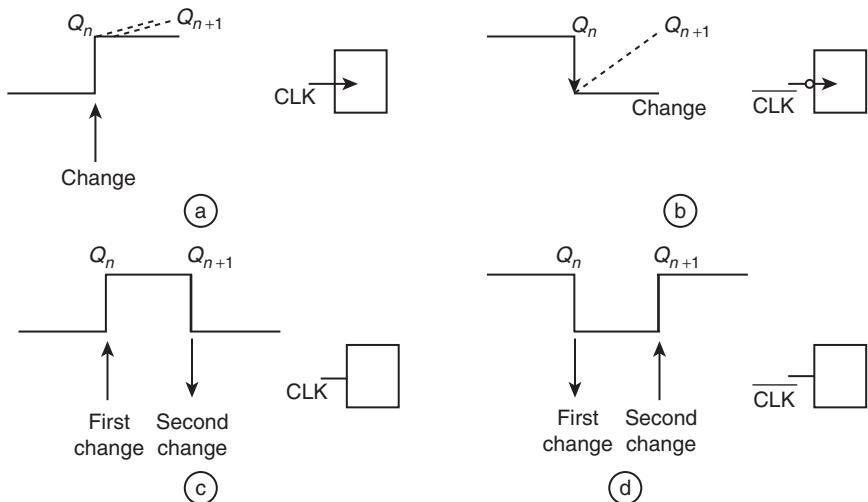


FIGURE 6.11 (a) Positive edge triggering clock action (b) Negative edge triggering clock actions (c) Pulse triggering by positive going followed by negative going transition (d) Pulse triggering by negative going followed by positive going transition.

6.9 PULSE CLOCKING OF THE LATCHES IN THE FLIP-FLOPS

Let us refer to Figures 10.11(c) and (d). These show the pulse triggering by positive going followed by negative going transition and pulse triggering by negative going followed by positive going transition, respectively.

Master-Slave method gives an alternative and is adopted for a pulse clocking. Pulse clocking means that the outputs are obtained after logic level changes firstly from 0 to 1 and then from 1 to 0. The first change puts the datum or data in the master outputs and the second change transfer the data to the slave outputs which provide the final outputs Q and \bar{Q} .

Sign ($\uparrow\downarrow$) at the CLK shows a positive going pulse clocking [Figure 6.11(c)].

It is also possible to design a latch based FF such that it responds first when there is a change at clock input from 1 to 0 and the second by the change from 0 to 1 as in Figure 6.11(d). Figure 6.11(d) shows the –ve going pulse clocking by the sign ($\downarrow\uparrow$).

6.10 CHARACTERISTIC EQUATIONS FOR THE ANALYSIS

Combinational circuits are represented by the Boolean expressions and SOPs. Equations can also be defined for the latches and flip flops.

Let Q_{n+1} is next succeeding state after the clock-triggered (level or edge or pulse) transition. Q_n is the state before the transition.

The following is the SR latch or FF characteristics equation:

$$Q_{n+1} = S + \bar{R}.Q_n \quad \dots(6.1)$$

The following is the JK FF characteristics equation:

$$Q_{n+1} = J.\bar{Q}_n + \bar{K}.Q_n \quad \dots(6.2)$$

The following is the T FF characteristics equation.

$$Q_{n+1} = \bar{T}.Q_n + T.\bar{Q}_n = T.XOR.Q_n \quad \dots(6.3)$$

The following is the D latch or FF characteristics equation:

$$Q_{n+1} = D \quad \dots(6.4)$$

■ EXAMPLES

Example 6.1

Two NOTs (or common input NANDs or NORs) are cross-coupled as shown in Figure 6.12 by feedback of Q to first NOT and of \bar{Q} to the second NOT. Explain how the circuit will work as bi-stable element.

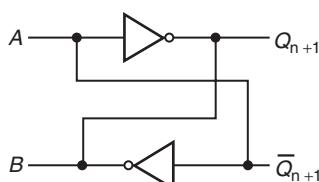


FIGURE 6.12 Cross-Coupled NOTs as a bistable element.

Solution

Let at the upper NOT gate the input $A = 1$ when the $Q = 1$ at an n^{th} instance. $Q = 0$ at the $(n + 1)^{\text{th}}$. Then due to cross coupling of Q with B , the $B = 0$ and hence \bar{Q} remains stable at 1. This is one stable state of the circuit, called Reset state— $Q = 0$ and $\bar{Q} = 1$.

Let at the upper NOT gate the input $A = 0$ when $Q = 0$ at an m^{th} instance. $Q = 1$ at $(m + 1)^{\text{th}}$. Then due to cross coupling of Q with B , the $B = 1$ and hence \bar{Q} remains stable at 0. This is another stable state of the circuit called Set state— $Q = 1$ and $\bar{Q} = 0$.

Whether A input is 1 or 0 is just a matter of chance or can change due to thermal noise. Circuit can be meta-stable for an unpredictable period and changing from one state to another. The state table will now be given as follows:

TABLE 6.11

Inputs		Outputs		State
A		Q_{n+1}	\bar{Q}_{n+1}	
0		1	0	SET (Stable State 1)
1		0	1	RESET (Stable State 2)
<i>Unstable Between 1 and 0</i>		1 or 0	0 or 1	Unpredictable*

*Meta stable state. However a noise, which makes at an instance $A = 1$ or a noise that makes at an $A = 0$ will bring back the circuit in stable state, reset or set, respectively.

Example 6.2

If in Figure 6.2(a) circuit S denotes the R input and R denotes S input, show that we get an $S - R$ latch. Why?

Solution

Circuit works as $\bar{S} - \bar{R}$ latch. Reset state activates on $\bar{R} = 0$ and $S = 1$ and set state activates on $S = 0$ and $R = 1$. Figure 6.2(b). State table columns for two inputs will interchange and the table will now be given as follows:

TABLE 6.12 A SR latch with 0 state activating an R or S

Inputs		Outputs		State
S	R	Q_{n+1}	\bar{Q}_{n+1}	
0	0	Q_n	\bar{Q}_n	No Change
1	0	1	0	RESET
0	1	0	1	SET
1	1	*	*	Unstable

*Unpredictable behaviour, when inputs R and S simultaneously becomes 0 immediately afterwards and meta stable state results.

Example 6.3

Explain the working of an *S-R* latch with circuit similar to Figure 6.2(a) made from the NORs. (Set occurs on *S* input = 0 and Reset occurs on *R* input = 0).

Solution

Circuit works as *SR* latch with reset state activating on $R = 0$ and $S = 1$ and set state activating on $S = 0$ and $R = 1$. The state table will now be given as follows:

TABLE 6.13 State table for cross coupled NORs

Inputs		Outputs		State
<i>S</i>	<i>R</i>	Q_{n+1}	\bar{Q}_{n+1}	
0	0	Q_n	\bar{Q}_n	No Change
1	0	1	0	RESET
0	1	0	1	SET
1	1	*	*	Unstable

*Unpredictable behaviour, when inputs *R* and *S* simultaneously becomes 0 immediately afterwards and meta stable state results.

Example 6.4

Explain the working of an *SR* latch with circuit similar to Figure 6.2(a) made from the NORs but the *S* input is at the upper NOR, which give Q output and *R* input is at the lower NOR, which gives \bar{Q} output.

Solution

The state table will now be given as follows:

TABLE 6.14 State table for cross coupled NORs with S input to NAND providing Q output

Inputs		Outputs		State
<i>S</i>	<i>R</i>	Q_{n+1}	\bar{Q}_{n+1}	
0	0	Q_n	\bar{Q}_n	No Change
1	0	1	0	SET
0	1	0	1	RESET
1	1	*	*	Unstable

*Unpredictable behaviour, when inputs *R* and *S* simultaneously becomes 0 immediately afterwards and meta stable state results.

Example 6.5

Why does a meta-stable (unstable for a short period) state exist in the circuit of examples 6.1 to 6.4 for a very brief but for an unknown amount of period?

Solution

Meta stable state occurs either due unpredictable 1 or 0 in example 6.1 and or due to both *R* and *S* input becoming identical and bring the circuit to unstable state and next to that *R* and *S* both becomes identically to ‘no change state’. However, a noise

always occurs in a circuit, may be for a very brief period. For example in circuit of example 6.1, when a noise makes at an instance $Q = 1$ or a noise that makes at an instance $\bar{Q} = 0$ will bring back the circuit in stable state, Reset or Set, respectively. Q becoming 1 or 0 at an instance will also bring the circuit back to stable state.

Example 6.6

Explain a clocked (gated) SR latch (an SR latch with a clock input). Why is it called a transparent latch?

Solution

Let us add two NAND gates to the two cross-coupled NANDs in circuit of Figure 6.2(a). Figure 6.4(a) showed this new circuit. It is an *R-S* latch circuit, which has a gating. It means provide a clocking facility. The *S* and *R* inputs are at the different NANDs. *S* input is now to upper left most NAND, and *R* to lower left most. It is also called a *RS* transparent latch or a clocked or gated *SR* latch.

The state tables of Figures 6.2(b) and 6.4(b) are for NANDs based Set Reset latch and a gated or clocked *RS* latch, respectively. These differ in the following respects.

Let Q_n denotes the output at terminal Q before the clock input becomes at a logic state 1, and let Q_{n+1} denotes the output at terminal Q after the clock input sets to 1.

1. Q_{n+1} equals to Q_n for $R = 0, S = 0$ and $CLK = 1$. (No change of state).
2. $Q_{n+1} = 0$ for $R = 1, S = 0$ after the clock becomes 1. It means a clear or reset signal 1 makes Q clear (whatever may be previous Q) after the clock input equals 1.
3. Q is Set to 1 for $S = 1, R = 0$ after the clock becomes 1.
4. While $S = 0$ and $R = 0$ after a Set or Reset causes no difference as per condition 1 above.
5. $R = 1$ and $S = 1$ after the clock becomes 1 causes an unstable output. It means the output becomes indeterminate or races through at an instant.
6. $CLK = 0$ inactivates the effect of the *R* and *S* inputs, the output remains same as before, $Q_{n+1} = Q_n$.

The two rows of the state tables of Figures 6.2(b) and 6.4.(b) are exactly opposite for defining the ‘No change’ (Q_{n+1} is equal to Q_n) and unstable states.

Points to Remember

In a clocked (gated) *SR* latch, the output can change only after the clock logic state transition from 0 to 1 or whenever the clock input is at logic state 1 or before the clock state transition from 1 to 0, but there is no change at any outputs if clock input is at 0. Clocking input is acting like a gate for the *S* and *R* inputs. It acts like a change-enable input bit. For this very reason, the present circuit of Figure 6.4(a) is also called a transparent *SR* latch. There is a transparency when the clock input = 1. A gate (or clock) pin controls the period during which input changes can affect the Q and \bar{Q} and latch is transparent.

The gating NANDs latch can be designed using a CMOS IC 4011B. Using cross-coupled NANDs, we get the SR latch with R , S , Q and \bar{Q} terminals.

Example 6.7

Explain the difficulties in using a clocked (Transparent) SR Latch?

Solution

The unstable state exists for an unpredictable period for a certain condition of $R = S$ input. (Condition whether $R = S = 1$ or $R = S = 0$ depends on the circuit made by NANDs or NORs).

Example 6.8

Show the timing diagram for a clocked SR latch with clock input is active level 1.

Solution

Figure 6.13 shows the timing diagram of a clocked SR latch having state table as per Figure 6.4(a). Q changes only during $CLK = 1$ and are as per S and R .

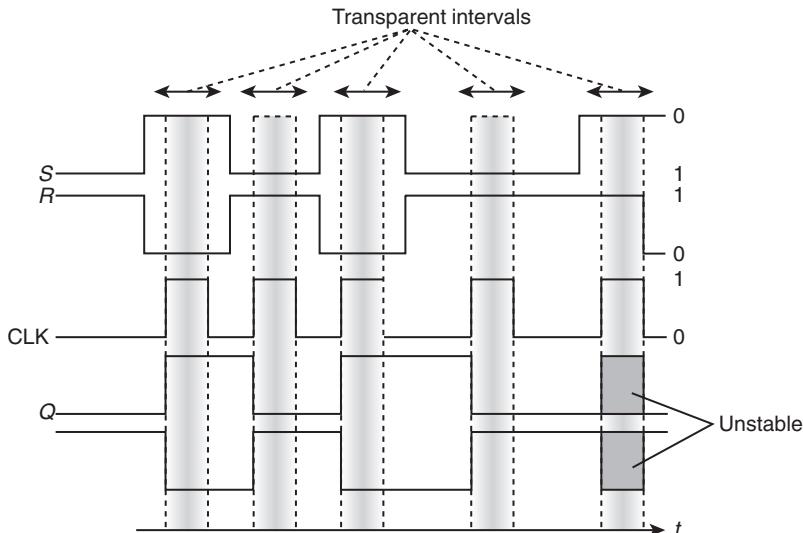


FIGURE 6.13 Timing diagram of a clocked SR latch having state table as per.

Example 6.9

Describe edge triggered $SR FF$.

Solution

Figure 6.14 (a) the logic circuit of an edge triggered SR latch in which during gating instead of clock at level input 1 in Figure 6.4(a), there is an edge at which the inputs can reflect the change at the output Q and hence at \bar{Q} . Figure 6.14 (b) shows the state table for the same. Figure 6.14 (c) shows symbol of the $S-R FF$.

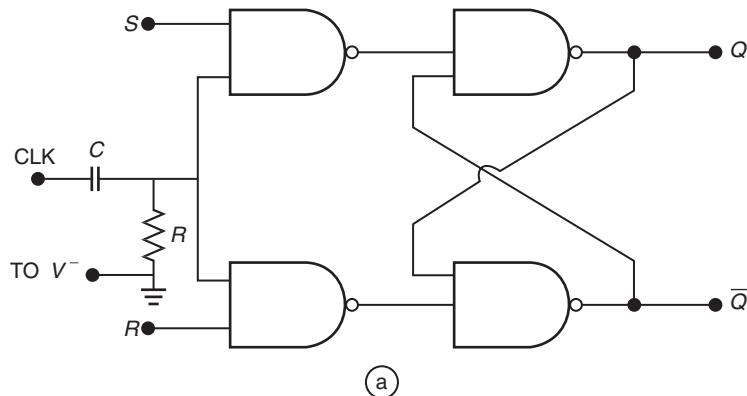
Example 6.10

How does a latch differ from a flip flop in strict sense?

Solution

A latch is a class of flip flop in which the instance at which output changes is uncontrolled. [Uncontrolled means not controlled by the timing input (called clock edge input).] A timing input (called clock edge input) can control the instance at which the output changes. The outputs can reflect change during a transparency interval.

A flip flop on the other hand is expected to have defined by a controlled instance of a clock edge after which the output changes occur. Note: Inputs must be set t_s before the edge and must hold, for t_h the hold period after the edge. Sometimes, a transparent latch is also loosely referred as a flip flop.



(a)

R	S	CLK	Q_{n+1}
X	X	$0, \downarrow, 1$	Q_n
1	1	\uparrow	*
0	0	\uparrow	Q_n
1	0	\uparrow	0
0	1	\uparrow	1

(b)

* Unstable



(c)

FIGURE 6.14 (a) Logic circuit of an edge triggered SR flip flop (b) State table of an SR flip flop (c) Symbol of a positive edge triggered S-R flip flop.

Example 6.11

Show the two timing diagrams concurrently for both the +ve and -ve edge triggered JK flip flops.

Solution

A timing diagram depicts a state table of any flip flop more clearly. Figure 6.15 shows two timing diagrams concurrently for both the +ve and -ve edge triggered JK flip flops. Two top most plots show the format of the chosen inputs for J and K . The middle part shows the chosen clock input format. It also shows + edge instance for

the case of the positive edge triggered JK FF. The two bottom-most plots explain the functions of +ve and -ve edge triggering in the JK FFs.

Example 6.12

Explain timing diagram drawn in Figure 6.15 for a logic circuit of an edge triggered JK FF.

Solution

Let J and K inputs vary with time as shown in Figure 6.15. Let CLK input also vary as shown here. This type of CLK input shows positive edges at $n, n+1, n+2, n+3$ that are denoted in the Figure by up arrows at various instances. It shows negative edges at $n, n'+1, n'+2, n'+3$, which are denoted in the Figure by down arrows.

Let us first consider Q at a positive edge triggered JK FF. We find that before the n^{th} edge $J, K = 0, 1$, respectively. Therefore, Q becomes 0 after this edge and before the $(n+1)^{\text{th}}$ edge, J, K becomes 1, 0. Therefore, at $(n+1)^{\text{th}}$ edge Q becomes 0. Just before the $(n+2)^{\text{th}}$ edge and $(n+3)^{\text{th}}$ edges, both $J, K = 1, 1$, therefore, Q toggles to 0. At $(n+3)^{\text{th}}$ edge, JK FF finds both $J, K = 0, 0$, therefore, Q remains 0.

Now let us consider output expected from a negative edge triggered JK FF. J and K inputs are 0, 1 at the n' edge. Therefore, $Q \rightarrow 0$. Just before $(n'+1)^{\text{th}}$ edge, J and K are 1 and 0 therefore $Q \rightarrow 1$. Just before $(n'+2)^{\text{th}}$ edge, $J, K = 1, 1$. Therefore, Q toggles to '0'. Before $(n'+3)^{\text{th}}$ edge, both J and $K = 0$, therefore, Q remains 0.

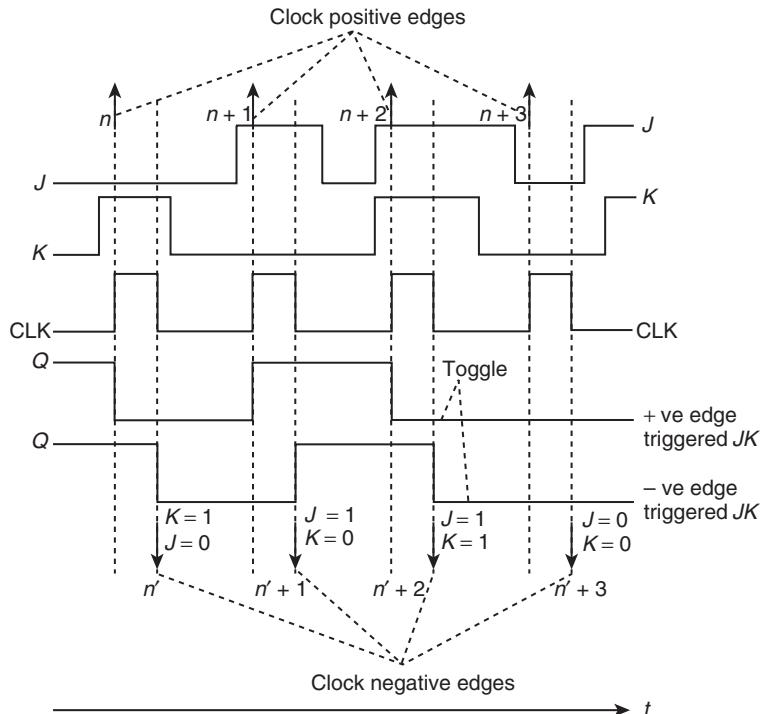


FIGURE 6.15 Timing diagrams for both the + ve and -ve edge triggered JK flip flops.

A negative edge triggered *JK FF* finds extensive application as building block in the counters where the *J* and *K* inputs are pegged to 1, and *JK FF* functioning is then analogous to that of a *T FF* circuit of Figure 6.5(a).

Example 6.13

How will we use the *J-K FF* for designing a *T-FF*?

Solution

A *JK FF* when $J = 1$ and $K = 1$ toggles on the clock input. Therefore, either by leaving the *J* and *K* input floating in case of a TTL gate or connecting these to logic 1 state (necessary for CMOS), we would get a *T FF* and we use the clock edge input as *T* input.

Example 6.14

How will we use the *JK FF* for designing a *D-FF*?

Solution

A *JK FF* when $J = 1$ and $K = \bar{J} = 0$ on the clock input gives the *Q* output = 1. The *JK FF* when $J = 0$ and $K = J = 1$ on the clock input gives the *Q* output = 0. For a *D FF*, the *D* input passes to *Q* output on a clock transition. Therefore, if we connect *K* input from *J* through a NOT gate (or a common input NAND), we can use *J* input as *D* input. We get a *D FF* when we use the *J* input as the *D*-input provided *K* is made complement of *J*.

Example 6.15

How will we use the *SR latch* for designing a *D latch*?

Solution

An *SR latch* when $R = 1$ and $S = \bar{R} = 0$ on the clock input gives the *Q* output = 0 = *S*. The *S-R latch* when $R = 0$ and $S = \bar{R} = 1$ on the clock input gives the *Q* output = 1. For a *D FF*, the *D* input passes to *Q* output on a clock transition. Therefore, if we connect *S* input to *R* through a NOT gate (or a common input NAND), we can use *S* input as *D* input. We would get a *D latch* when we use the *S* input as the *D* input provided *R* is made complement of *S*.

Example 6.16

How will we use a *T FF* as a pulse frequency divider and how will we use a chain of *T FFs* as a counter?

Solution

Figure 6.6(d) showed the effect of successive pulses at the *T* input. If input frequency is 300 kHz, at *Q* the output frequency is 150 kHz. The *T flip flop* is, therefore, like a counter and a frequency divider. The *T flip flop* is also called, scale-of-two circuit. *T FFs* are used in the counters. A binary counter has many *T flip flops* in cascade such that each *T* is connected to the output of the previous *T flip flop* (refer Chapter 7). Normally, for the counting applications, the *JK flip flops* circuits are converted into the cascaded *T flip flop* circuits of Figure 6.6(a).

Example 6.17

How will we use the MSI ICs for designing a (i) D -FFs and (ii) T FFs based sequential circuit?

Solution

- (i) The D FFs designed from TTL 74x74 family ICs, TTL 74x374 family and CMOS 74HC374 family. IC 74x374 has eight number D FFs with Q_0 to Q_7 as the eight outputs. Each output is in tristate and is enabled (to logic 1 or 0) only when an out enable input is given after the clock transition. This is due to incorporation of a tristate buffer at each of the Q output with a common out enable input for the eight outputs.
- (ii) The T FFs are designed from IC 74x109 family of gates.

Example 6.18

Describe functioning of a JK Master Slave flip flop.

Solution

When the input at CLK is 0, the slave output is fixed according to Q' and \bar{Q}' . It means according to the state of outputs from the master circuit. This is so because in that case T' is 1 because when the CLK is 0, the T' is complement of CLK due to a NOT gate between the CLK and T' . The MT section does not respond, and the slave section can be said to be locked when $CLK = 0$. When the CLK will become again 0 from 1, the T' shall be becoming 1 and the Slave will now respond.

In other words, master section accepts and open locks for the J and K inputs when the clock moves from logic level 0 to 1 and the slave section accepts the master section outputs when the clock moves from 1 to logic level 0.

If both the J and K inputs are 1 the CLK input acts like a toggling input. This can be seen as follows. When CLK is 0, there is no effect and the slave simply accepts the Q' and \bar{Q}' . When CLK moves to 1 state, the MT box gets both set and reset inputs as 1. According to table in Figure 6.4(b), the outputs Q and \bar{Q} shall remain as before. Upon CLK again moving to level 0, the slave will provide Q and \bar{Q} same as before toggling.

Example 6.19

What are the uses of JK MS flip flop clear and preset inputs?

Solution

JK flip flop with clear and preset is a circuit of Figure 6.10(a) with R and S inputs also provided to a user of the JK FF as the additional inputs. If inputs are given at the points R and/or S in Figure 6.10(a) circuit, then the circuit shall respond with a priority to the changes at these inputs. For example, if S input is activated and made 0, the output Q shall stay 1. No matter what are the states occurring at the T -input. The S input and R input are, therefore, having highest priority. Circuit of Figure 6.10(a) also acts as the SR latch if its T input is not used at all.

Example 6.20

How will we use the MSI ICs for designing (i) D latches based (ii) -ve edge triggered JK and (iii) Pulse triggered (Master Slave) JK sequential circuits?

Solution

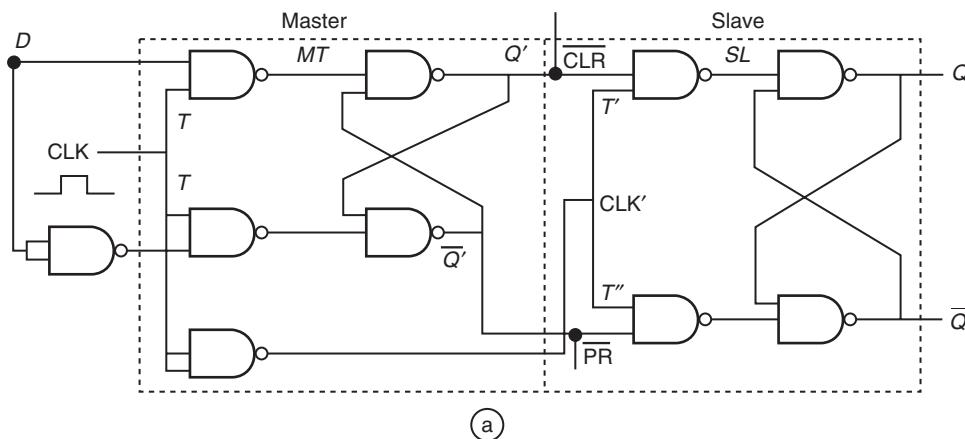
IC TTL 74x75 or IC CMOS 74HC75 provides the D latches. D latch is also cheaper than a D flip flop because there are less numbers of gates. The each of the ICs TTL 74373 and CMOS 74HC373 have eight numbers D latches with each output in tristate. All eight outputs are enabled (to logic 1 or 0 as per corresponding D input) only when an out enable input is given after the clock is inactivated. This is due to incorporation of a tristate buffer at each of the Q output with a common out enable input for the eight outputs.

A –ve edge triggered JK and a pulse clocked (master slave) JK FFs: These are made by using the ICs 74HC108 and 74HC76, respectively.

Example 6.21 Explain master slave (pulse triggered) D FF.

Solution

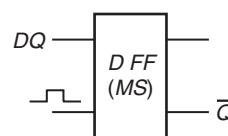
Figure 6.16(a) shows master slave (pulse triggered) D-FF. There is one input called D input in place of S input in RS- MS FF of Figure 6.9(a) and J input of Figure 6.10(a). The circuit is called master-slave D flip flop (D MS FF) due to reason that a master section drives a slave section when CLK input becomes 0. It is also called pulse triggered D flip flop also due to pulse on going positive and negative transitions both are needed to obtain the output Q and \bar{Q} .



(a)

Master inputs		$T - T'$	$T' - T''$
S	CLK	Q'	Q
1	—	1	1
0	—	0	0
1	0	Q_n	Q_n
0	1	Q_n	Q_n

(b)



(c)

FIGURE 6.16 (a) Master-Slave D flip flop logic diagram (b) State table for master slave D flip flop (c) Logic symbol of D -FF.

Figure 6.16(b) gives the state table of a *D MS FF*, and Figure 10.10(c) shows its symbol. This flip flop consists of a combination of the gated *SR* and *T* inputs latching circuits. The *S* input is connected permanently to *R* input through a NOT gate and is labeled as *D* input.

Master Section Response

Let us now assume that clock input becomes at logic state 1. Master outputs Q' and \bar{Q}' will now change according to the *D* input. This is analogous to action of a transparent latch in Figure 6.4(a). If $D = 0$, then the output according to table in Figure 6.4(b) shall be 0. In other words, Q' and \bar{Q}' , the master section outputs, shall be 0 and 1, respectively.

Here, we *D* input [in the master section of Figure 6.16(a) is equivalent to the *S* input in Figure 6.4(a), and inverted *D* input in this *MT* section is equivalent to the *R* input in Figure 6.4(a)].

The Q and \bar{Q} , the outputs of box marked *SL* (called slave) also remain unchanged as *T'* CLK input to *SL* section is 0 due to NOT before CLK'.

We find that the first gated NAND flip flop section and the box marked *ML* is providing the outputs Q' and \bar{Q}' according to *D* (and internally created \bar{D}), when CLK becomes 1. This is also the reason that the *ML* section is called master section. Slave is inactive during the transparency of *ML* because *SL* has *T'* CLK input = 0. When master finishes the action, then only slave can act.

CLK becomes 0 – Slave Section Response

When Q' is 0 and \bar{Q}' is 1, this means CLR is 0 and PR is 1. Therefore, according to second row of table in Figure 6.16(b), the outputs $Q = 0$ and $\bar{Q} = 1$. Had *D* been 1, the outputs, upon clock going to 1 would have been according to first row of the table as $Q = 1$ and $\bar{Q} = 0$.

We find that the second gated NAND flip flop section and the box marked *SL* is simply providing the outputs Q and \bar{Q} on CLK becoming 0 according to Q' and \bar{Q}' . The latter are the outputs of master section when CLK logic state was 1. This is also the reason that the *SL* section is called slave section.

The present circuit *D* flip flop action arises of the combination of master and slave. There is no possibility of a false transition at $(n+1)^{\text{th}}$ state on the positive going clock pulse (0 to 1) and followed negative going (1 to 0) in a *D MS FF*. After positive going from 0 to level 1 transition, the master is triggered and the slave is idle. After the negative going from 1 to 0, the slave responds as per master outputs and the master is idle.

1. When $D = 1$ the $Q = 1$ after a delay following the application of the positive going the clock pulse.
2. When $D = 0$ the $Q = 0$ 1 after a delay following the application of the positive going the clock pulse.

We also note as follows:

1. *D MS FF* delay = Propagation delay + setup time and hold time and propagation delay = pulse duration between positive and negative going transitions.
2. Using master slave we implement equivalent of *D* edge triggered *FF* by using two latches, one as a master and one as a slave.

3. When the triggering is at the positive and negative edges both, it is called pulse triggering or pulse clocking.

Example 6.22

What is the advantage of edge triggering in the FFs over pulse triggering FFs?

Solution

When there is pulse clocking mechanism, the important condition is inputs must be held constant for a time, which is longer than the clock duration between \uparrow (0 to 1 transition) and \downarrow (1 to 0 transition) or between \downarrow and \uparrow transitions. This condition puts the pulse-clocking mode of clock input (for changing the outputs as per other inputs) at a disadvantage.

Edge triggering mode obviates this disadvantage. The FF accepts the input data and transfers as per state table to the Q and \bar{Q} outputs either upon \uparrow as in Figure 6.11(a) at the CLK input or upon \downarrow as in Figure 6.11(b). The circuits of Figure 6.6(a) and Figure 6.7(a) are the examples of edge triggering.

The FF responds to either rising edge at the CLK input or at the falling edge at the CLK input. The FF response is quite fast in the edge-triggering mode. The disadvantage of edge triggering is the necessity of a sharp rise or fall edge generating circuit. [Use of a RC pair or a Schmitt trigger circuit at the CLK input. The latter is more refined circuit than C and R based circuit shown in Figures 6.6(a) and 6.7(a). Nowadays the superior circuits have become available for fast edge-triggering clock generation.]

Example 6.23

Show the timing diagram for a D-MS FF with Preset and Clear inputs.

Solution

Figure 6.17 shows the timing diagram of a D MS FF and having state table as per Figure 6.16(b).

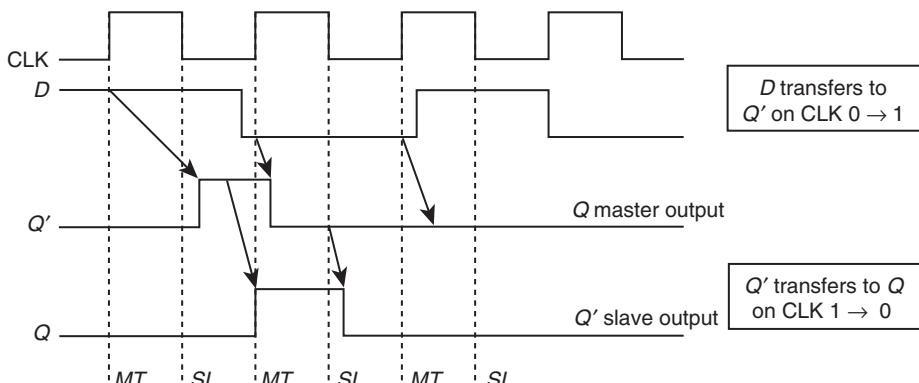


FIGURE 6.17 Timing diagram of a D-MS FF having state table as per Figure 6.16(b) (Q' is as per D at $0 \rightarrow 1$ and Q is as per Q' at $1 \rightarrow 0$).

Example 6.24

How will we use an *S-R* latch to Design a Bounce less switch?

Solution

When a switch is pressed or released, there are ups and downs for a short period in the switched current or voltage. Effect of the bounces on the switched voltage appears like one shown in Figure 6.18(a). This effect arises due to the spring actions and reactions inside the switches. An *SR* latch based debouncing circuit is shown in Figure 6.18(b).

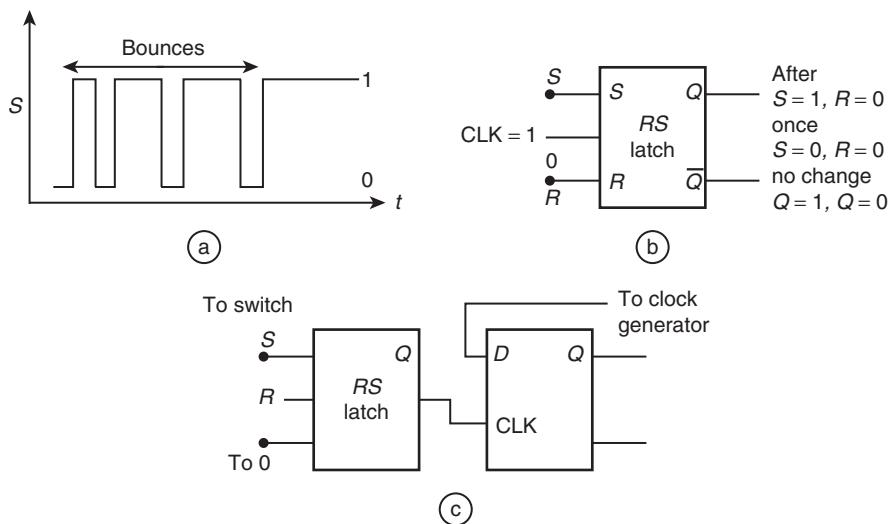


FIGURE 6.18 (a) Effect of the bounces on the switched voltage through after pressing a switch due to the effects of the spring actions and reactions inside the switches. (b) An *SR* latch based debouncing circuit. (c) A clock start and stop circuit using *SR* latch and *D* latch.

Let the state at Q is idle (rest or steady) state of the switch when a switch is in the middle. A cross-coupled gated NAND latch at left side of Figure 6.4(a) is now useful for the debouncing action. In the idle state, the both inputs, R and S are 1, and the output, Q , is as per table in Figure 6.4(b). The Q is unchanged (stable at 1 or 0). When the switch is moved up side at first time contact, the R input is 0 while S input is at 1 (being connected to far of down end of the switch to the supply V_{CC} end). For $R = 0$ and $S = 1$, the Q becomes 1. When switch is bouncing that making repeated contacts for a short while. It means S repeatedly becomes 0 and 1, the Q remains 1. Finally when the switch settles towards up, Q is set to 1.

Now let us consider, situation when the switch is moved downwards to make at first contact $R = 1$ and $S = 0$. Therefore, as per state table of cross-coupled gated NANDs based *SR* latch, Q resets to become 0 (refer fourth row of Figure 6.4(b)).

Advantages of the gated cross-coupled NANDs based debouncing:

1. Use of *SR* latch obviates the need of placement of the external components like a capacitor.
2. Further the latch acts fast. Only first contact will produce desired

action in a time just equal to the set up time for $S(t_s)$ plus the propagation delay time (t_p) of the latch.

Example 6.25 How will we design the clock input start and stop circuit using an SR latch?

Solution

Refer to the Figure 6.18(a) circuit. A debouncer circuit can also be used to start and stop the clock pulses from a clock generator (sequentially bistable circuit). The generator may be designed using an IC 555 or 7555 or another. The generator pulses its CLK output to a $D\text{ FF}$. The Q output of SR latch is CLK input to a D -latch. The output of SR latch is the desired output controlled by S . Figure 6.18(c) shows a circuit.

■ EXERCISES

1. The timing plots of S and R inputs are as shown in Figure 6.19. Copy these on a graph paper. Now show the plots of Q and \bar{Q} from a circuit of figure 6.2(a) on the same graphs and same X -axis.
2. If Figure 6.19 R and S inputs connect a NOR based SR and S input is to that NOR, which is giving Q output. Draw the timing diagram for R , S , Q and \bar{Q} .
3. Explain the working of an S-R latch with a gate (clock) input.
4. Figure 6.20 shows a timing diagram with four plots for A , B , CLK and Q . Find what are A and B ? (i) R and S (ii) S and R (iii) S and R . (iv) D and D 's complement (v) J and K .
5. Figure 6.21 shows a timing diagram with four plots for A , B , CLK and Q . Name the flip-flop or latch, which you will use to obtain this timing diagram.
6. A D -latch with propagation delay = 10 ns has its \bar{Q} output connected to D -input. Show that pulses of 20 ns will start appearing as long as clock input remains 1.
7. A D -latch with propagation delay = 10 ns has its \bar{Q} output connected to D -input. A clock input connects to the pulses 1 for first 75 ns and 0 for next 25 ns, again 1 for next 75 ns and 0 for 25 ns. Show the output Q in the total period 0 to 200 ns on a graph paper. (Hint: During first 75 ns, from 10ns onwards, the Q will toggle every 10 ns).

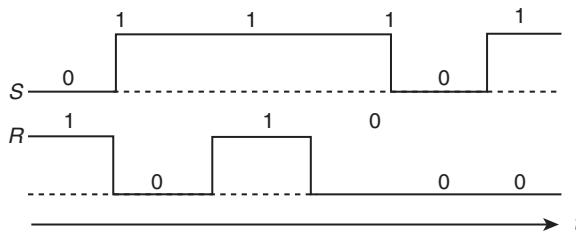


FIGURE 6.19 The timing plots of S and R inputs.

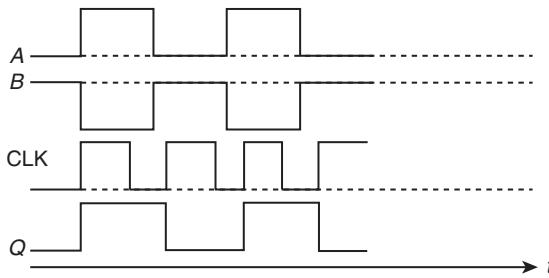


FIGURE 6.20 Timing diagram with four plots for A, B, CLK and Q to find what are the A and B.

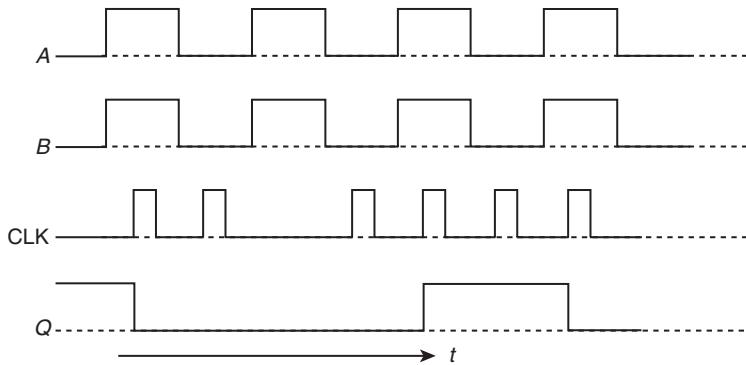
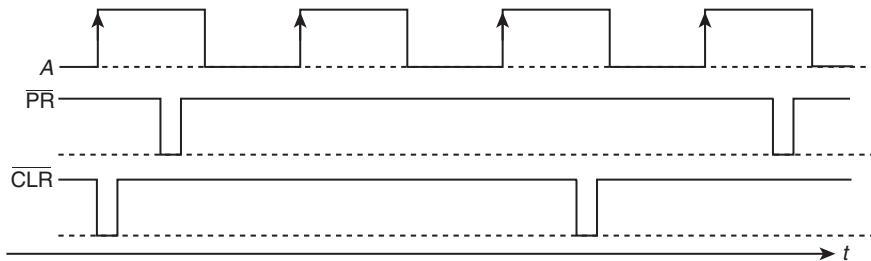


FIGURE 6.21 Timing diagram with four plots for A, B, CLK and Q to find what is the name of the FF or latch.

8. Figure 6.22 shows a timing diagram for the preset \overline{PR} , clear \overline{CLR} and T inputs of a TFF. Draw the time plots for the Q and \overline{Q} .
9. Why do you use a D latch if the instance at which input will be arriving is not known exactly, only the interval during which it will arrive is known?
10. Why do you use D FF, when input availability is certain and it is to be registered at the output?
11. Show the timing diagram for a negative edge triggered D FF.


 FIGURE 6.22 Timing diagram for the Preset, clear and T inputs of a TFF for obtaining the time plot for the Q and \overline{Q} .

12. Draw a logic circuit for a master slave JK flip flop, the master section of which triggers by a negative going pulse.
13. How will we use the SR latch sub unit of a $MS-JK\ FF$?
14. How will we use the $JK\ FF$ for designing a negative edge triggered $T\ FF$?
15. Setup time in a FF with propagation delay of 20 ns is 5 ns and hold time is 5 ns. What is the minimum period for which input should be stable?
16. Show the timing diagram for a $D\ MS\ FF$ with preset and clear inputs with negative going pulse needed for triggering the master.
17. How will we use a NOR based SR latch to design a bounce less switch?
18. How will we design the clock input start and stop circuit using a SR latch? (Hint: Convert it first to the D -latch).

■ QUESTIONS

1. What is the meaning of a flip flop circuit? What is the meaning of a flip? What is the meaning of a flop?
2. What is a latch? Explain it with example of a cross coupled NANDs? What will be the resulting state table if cross coupling of NORs done?
3. What is difference between the latch, flip flop and master slave flip flop?
4. Why is a flip flop also called a bistable?
5. How does a SR latch differ from a gated RS latch?
6. How is a $JK\ FF$ freed from unstable condition found in an SR latch, whether gated or simple?
7. What is the use of the preset and clear inputs in a $D\ FF$ or a D latch or a $JK\ FF$?
8. What are the differences in a Master Slave $JK\ FF$, a +ve edge triggered $JK\ FF$ and a -ve edge triggered $JK\ FF$?
9. Why does a D flip flop give a storage register unit and a T type flip flop give a basic counting unit?
10. A $JK\ FF$ is with J and K inputs not provided. When can it work as equivalent of a $T\ FF$?
11. What is the difference between a D Latch and a $D\ FF$? Explain uses of each.
12. What are the differences between D , T and $JK\ FF$?
13. Explain master slave circuit in a $JK\ FF$?
14. Normally, \bar{Q} is complement of Q . When is these not so? What is that state and in which cases, this condition encountered?
15. What do we mean by a state table in a FF or latch? How does it differ for a simple (without output feeding back an input) logic circuits' truth table? It is called state table. Why?

16. What is the meaning of a small circle at the (a) clock input, (b) PR input, and (c) CLR input?
17. What is meaning of a triangle at a clock input?
18. When is a circle followed by a triangle at a clock input?
19. What is the meaning of a transparent latch? What is transparent in a transparent latch?
20. Q output of $D\ FF$ or D latch is same as D input. Then, why is the $D\ FF$ more useful?
21. Explain, when does a $JK\ FF$ act as a divide by 2 circuit?
22. Explain, how does a T flip flop act as a divide by 2 circuit?
23. What is the meaning of bounce? What is the type of bouncing signal from a switch noticed in the digital circuits? How is its effect removed from the switched voltage or current?
24. Why is an SR latch used for debouncing and it is a better option than an Schmitt trigger circuit based denouncing?
25. How is an SR latch after a switch used to start and stop propagation of the digital signals?

CHAPTER 7

Sequential Circuits Analysis, State Minimization, State Assignment and Circuit Implementation

OBJECTIVE

We learnt in Chapter 6 the following concepts:

1. A sequential circuit is a circuit made up by combining the logic gates such that the required logic at the output(s) depends not only on the current input logic conditions but also on the past inputs, outputs and sequences.
2. A sequential circuit has the memory elements like flip flops and a combinational circuit(s) has no memory elements.
3. A clock input (or a set of inputs is used) to cause a transition to next state. The clock signal can be a gate (control) input 1 or control input 0 or a positive edge state (\uparrow) or at the clock state or a negative edge (\downarrow) or a positive going pulse ($\uparrow\!\!\uparrow$) or a negative going pulse ($\downarrow\!\!\downarrow$).

We have also learnt in Chapter 6 the SR , D , JK and T flip flops as the basic memory elements. A flip-flop *next state* is expressed by one of the following expressions:

$$SR\text{ FF:} \quad Q_{n+1} = S + R \cdot Q_n$$

$$D\text{ FF:} \quad Q_{n+1} = D$$

$$JK\text{ FF:} \quad Q_{n+1} = J \cdot \bar{Q}_n + \bar{K} \cdot Q_n$$

$$T\text{ FF:} \quad Q_{n+1} = \bar{T} \cdot Q_n + T \cdot Q_n = T \cdot \text{XOR} \cdot Q_n$$

Here, Q_{n+1} is the *next state* after the $(n + 1)^{\text{th}}$ transition and Q_n is present state.

There can be in general a sequential circuit consisting of a complex circuit, which combines the combinational circuit(s) and the memory section. We shall learn in this chapter the analysis of the clocked sequential circuits—their design, state minimization, state assignment and circuit implementation.

7.1 GENERAL SEQUENTIAL CIRCUIT WITH A MEMORY SECTION AND COMBINATIONAL CIRCUITS AT THE INPUT AND OUTPUT STAGES

A general sequential circuit network has memory section and combination circuits at the memory inputs and outputs. Assume the followings structure of a general sequential circuit.

1. The circuit memory section consists of the m of flip flops. These have a set $\underline{\mathbf{Q}}$ with the m present state outputs $Q_0, Q_1 \dots$ and Q_{m-1} .
2. There is a set $\underline{\mathbf{X}}$ of i inputs X_0, X_1, \dots, X_i . These are applied to a combinational circuit, the j outputs of which are the inputs to the memory section.
3. The present state of $\underline{\mathbf{Q}}$ changes on the clock transition(s) to next state $\underline{\mathbf{Q}'}$. The transition is by clock 1 or 0 or \uparrow or \downarrow or $\overline{\uparrow\downarrow}$ or $\overline{\downarrow\uparrow}$.
4. Further, a set $\underline{\mathbf{Y}}$ with outputs Y_0, Y_1, Y_j is obtained from the network as per $\underline{\mathbf{Q}}$ (or $\underline{\mathbf{Q}}$ and $\underline{\mathbf{X}}$) using a combinational circuit at the output stages.

Figures 7.1(a) and (b) show two general sequential circuits with above features, one for $\underline{\mathbf{Y}}$ as per $\underline{\mathbf{Q}}$ only and other $\underline{\mathbf{Y}}$ as per $\underline{\mathbf{Q}}$ and $\underline{\mathbf{X}}$, respectively. The outputs are also called Moore machine and Mealy machine outputs, respectively. *A sequential circuit is like a machine producing the states.*

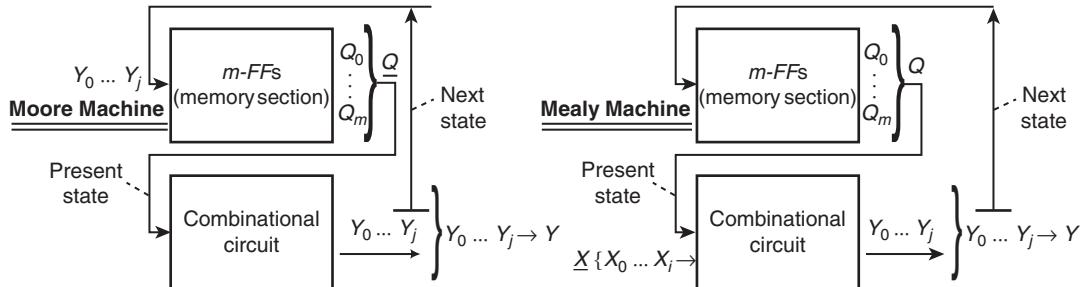


FIGURE 7.1 (a) A general sequential circuit-1 in which the final stage combinational circuit output depends on the $\underline{\mathbf{Q}}$ (Moore Machine Sequential Circuit) (b) A general sequential circuit-2 in which the final stage combinational circuit output depends on the $\underline{\mathbf{Q}}$ and $\underline{\mathbf{X}}$ (Mealy Machine Sequential Circuit).

7.2 SYNCHRONOUS AND ASYNCHRONOUS SEQUENTIAL CIRCUITS

7.2.1 Synchronous Sequential Circuit

Synchronous sequential circuit is a circuit in which the output $\underline{\mathbf{Y}}$ depends on present state $\underline{\mathbf{Q}}$ and present inputs $\underline{\mathbf{X}}$ at the clocked instances only. These instances can be defined by a clock input \uparrow or \downarrow or $\overline{\uparrow\downarrow}$ or $\overline{\downarrow\uparrow}$ and memory section activates to give next state $\underline{\mathbf{Q}'}$. Figure 7.2(a) shows synchronous sequential circuit in there the memory section undergoing change of state at the discrete clock instances c_i, c_j, \dots at the FFs.

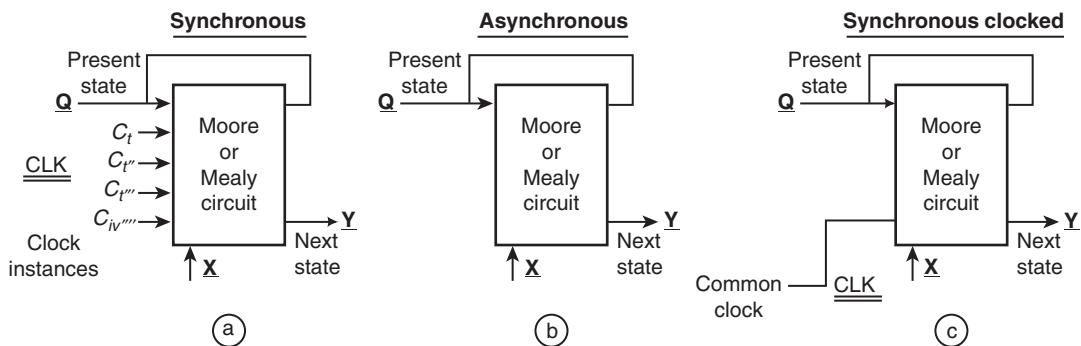


FIGURE 7.2 (a) Synchronous sequential circuit in which the memory section undergoes change of state only at the discrete clock instances (b) Asynchronous sequential circuit has the memory state changes at undefined instances and output is dependent on sequences of the inputs (c) Synchronous Clocked sequential circuit [Note: Memory section with the identical clock instances at each flip-flop].

7.2.2 Asynchronous Sequential Circuits

Asynchronous sequential circuit is a circuit in which not only the present inputs \underline{X} and present state \underline{Q} but also the sequences of changes affect the output \underline{Y} . The changes can be at the undefined instances of time. Figure 7.2(b) shows asynchronous sequential circuit with the memory section with the undefined clock instances and output dependent on the sequences of inputs.

7.3 CLOCKED SEQUENTIAL CIRCUIT

A clocked sequential circuit is a synchronous sequential circuit in which the output \underline{Y} depends on present state \underline{Q} and present inputs \underline{X} at a clocked instance, which is identical at all the flip flops at the memory section. Figure 7.2(c) shows clocked sequential circuit. Note that in the figure that the memory section undergoes change to next state input after an identical clock instance at each flip flop.

7.4 CLASSIFICATION OF SEQUENTIAL CIRCUIT AS MOORE AND MEALY STATE MACHINE CIRCUITS

7.4.1 Classification of a Sequential Circuit as Moore Model Circuit

Recall the sequential circuit models in Figure 7.1(a). A general clocked sequential circuit in which \underline{Y} final stage combinational circuit output is as per \underline{Q} only is called a circuit implementing a Moore machine and is Moore model of the sequential circuit.

Moore Model

$\underline{\mathbf{Q}}' = F_Q(\underline{\mathbf{Q}})$ (Next state outputs $\underline{\mathbf{Q}}'$ are the function of past state $\underline{\mathbf{Q}}$ as the present inputs at the clocking instance) and $\underline{\mathbf{Y}} = F_Y(\underline{\mathbf{Q}})$ ($\underline{\mathbf{Y}}$ is a function of present state outputs $\underline{\mathbf{Q}}$ before the clocking instance).

Figure 7.3(a) shows using two D flip flops a clocked sequential circuit-3 in which the final stage combinational circuit output $\underline{\mathbf{Y}}$ depends on the $\underline{\mathbf{Q}}$ as per Moore model. Following are the inputs, present states, outputs, input functions and output function of circuit-3:

Output Y is taken from AND gate having inputs from the between Q_1 and Q_2 of the FFs.

Input D_2 is from an AND gate having inputs X and Q_1 . Input D_1 is from the AND-OR array implementing $(X \cdot \bar{Q}_2 \cdot Q_1 + X \cdot Q_2 \cdot \bar{Q}_1)$.

7.4.2 Classification of a Sequential Circuit as Mealy Model Circuit

Recall the sequential circuit models in Figure 7.1(b). A general sequential circuit in which the final stage outputs $\underline{\mathbf{Y}}$ as per $\underline{\mathbf{Q}}$ and $\underline{\mathbf{X}}$ is called a circuit implementing a Mealy machine and is Mealy model of the sequential circuit.

Mealy Model

$\underline{\mathbf{Q}}' = F_Q(\underline{\mathbf{X}}, \underline{\mathbf{Q}})$ (Next state outputs $\underline{\mathbf{Q}}'$ are the function of past state $\underline{\mathbf{Q}}$ and present inputs $\underline{\mathbf{X}}$ at the clocking instance) and $\underline{\mathbf{Y}} = F_Y(\underline{\mathbf{Q}}, \underline{\mathbf{X}})$ ($\underline{\mathbf{Y}}$ is a function of present state outputs $\underline{\mathbf{Q}}$ before the clocking instance).

Figures 7.3(b) shows using one D and one JK flip flops an exemplary clocked sequential circuit 4 in which the final stage combinational circuit output $\underline{\mathbf{Y}}$ depends on the $\underline{\mathbf{Q}}$ and $\underline{\mathbf{X}}$ as per Mealy model. Following are the inputs, present states, outputs, input functions and output function of circuit 3

$$\text{Output } Y = \bar{X} \cdot Q_2 + Q_1.$$

$$\text{Input } J = Q_1 \text{ and } K = X. \text{ Input at } D = (\bar{Q}_1 + \bar{Q}_2) \cdot X.$$

Note

Any sequential circuit for the $\underline{\mathbf{Y}}$ can be designed as Mealy machine can be converted to a Moore machine by using appropriate set of $(\underline{\mathbf{X}}, \underline{\mathbf{Q}}, \underline{\mathbf{F}}_Q, \underline{\mathbf{F}}_Y)$. Similarly any sequential circuit for the $\underline{\mathbf{Y}}$ designed as a Moore machine can be converted to a Mealy machine by using appropriate set of $(\underline{\mathbf{X}}, \underline{\mathbf{Q}}, \underline{\mathbf{F}}_Q, \underline{\mathbf{F}}_Y)$.

7.5 ANALYSIS PROCEDURE

An analysis is important for implementing a sequential circuit. Analysis also provides a tabular description of the circuit. The methods for analysis are as follows:

1. Draw a logic circuit diagram.
2. Perform state variables assignments and excitation (means FF inputs) variables assignments.

3. Find the expressions for the excitations from the flip flop characteristics equations as per the excitations and make an excitation table. In other words, find $\underline{\mathbf{Q}}' = F_O(\underline{\mathbf{X}}, \underline{\mathbf{Q}})$ and $\underline{\mathbf{Y}}$
4. Make a transition table from the expressions for $\underline{\mathbf{Y}} = F_O(\underline{\mathbf{X}}, \underline{\mathbf{Q}})$ in case of Mealy model and $\underline{\mathbf{Y}} = F_O(\underline{\mathbf{Q}})$ in case of Moore model.
5. Perform state minimization and make minimal state table.
6. Draw the state diagram.

These steps are explained by taking examples of the circuit-3 and circuit-4 shown in Figures 7.3(a) and 7.3(b), respectively. *Examples 7.3 and 7.4 will explain the procedure.*

Definitions for the different terms used during the analysis of the circuit or during the design and synthesis for implementing the circuit mentioned above are as follows:

7.5.1 Excitation Table

An excitation table is a tabular representation of $\underline{\mathbf{X}}$ and $\underline{\mathbf{Q}}$ at the FFs and of $\underline{\mathbf{Y}}$ as per F_o for the combination circuit at the output stages. It gives present states and the inputs given at the memory section. It also gives the outputs that follow the excitations at the memory section.

Number of rows in each column equals 2^m where m is the number of flip flops because each flip flop has one Q output and m flip-flops can have 2^m different combinations of the states at the Q s. For example, if (Q_1, Q_2) are the Q s of two FFs, then $(Q_1, Q_2) = (0, 0), (0, 1), (1, 0)$ and $(1, 1)$ are the four combinations possible for the four different states of the memory-section present outputs.

First column of excitation table gives a present state (Q_1, Q_2) in its each row.

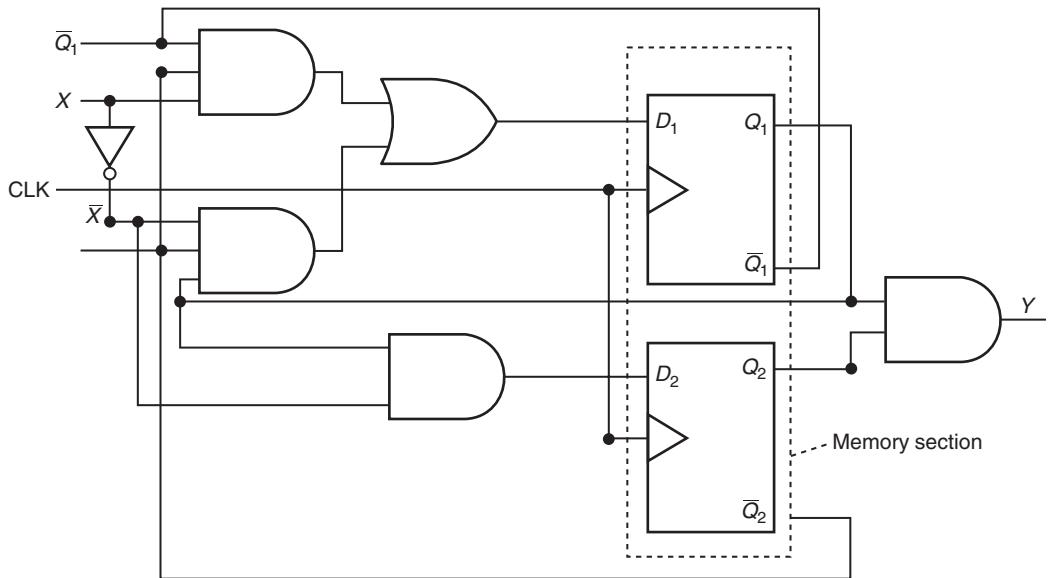
The number of columns for the excitation inputs $\underline{\mathbf{Q}}$ equals the number of possible combinations of external inputs in the set $\underline{\mathbf{X}}$. It equals 2^i if there are i distinct literal to represent the inputs when there are i inputs X_1, X_2, \dots, X_i . For example, if (X_1, X_2) are the external input to the memory section then $(X_1, X_2) = (0, 0), (0, 1), (1, 0)$ and $(1, 1)$ are the four combinations possible for the four different states of the memory section present outputs. For each set of inputs, there is a set of excitation inputs to the memory section, for example, corresponding to each set of external inputs, there will be four sets of inputs to (D_1, D_2) in case of two D FFs at the memory section.

Mealy Model

The number of columns for the output $\underline{\mathbf{Y}}$ also equals the number of possible 2^i combinations of external inputs in the set $\underline{\mathbf{X}}$. Suppose output stage has two outputs, Y_1 and Y_2 . Then there will be four columns for the case of four sets of external inputs and each column having entries for pair of outputs of (Y_1, Y_2) .

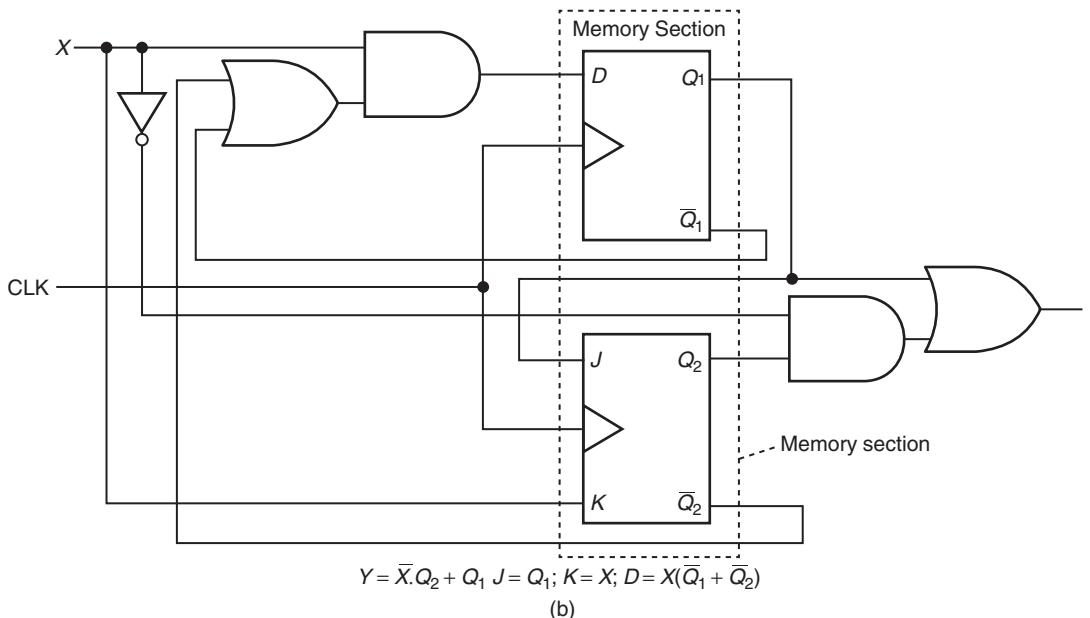
Moore Model

The number of column = 1 for the output $\underline{\mathbf{Y}}$ as in Moore model Y depends on Q s only. The column entries for values of (Y_1, Y_2) as per the combinational circuit between the memory section output Q s and Y s.



$$Y = Q_1 \cdot Q_2 \quad D_1 = (X \cdot \overline{Q}_1 \cdot \overline{Q}_2 + \overline{X} \cdot Q_1 \cdot Q_2); \quad D_2 = X \cdot Q_1$$

(a)



$$Y = \overline{X} \cdot Q_2 + Q_1 \quad J = Q_1; \quad K = X; \quad D = X(\overline{Q}_1 + \overline{Q}_2)$$

(b)

FIGURE 7.3 (a) Using two D flip-flops a clocked sequential circuit-3 in which the final stage combinational circuit output depends on the Q as per Moore model (b) Using two JK flip-flops an exemplary clocked sequential circuits-4 in which the final stage combinational circuit output depends on the Q and X as per Mealy model.

7.5.2 Transition Table

A transition table is a tabular representation of F_Q and F_O . It shows how the sequential circuit FFs will respond to all the present inputs X s and Q s and will generate Y s from the Q s.

Number of rows in each column equals 2^m for the m -FFs.

First column of transition table gives a present state (Q_1, Q_2) in its each row for the case of $m = 2$. This is because there are four combinations possible for the four different states (of Q s) of the memory section present outputs.

The number of columns for the next state \underline{Q}' equals the number of possible combinations of external inputs in the set \underline{X} . It equals 2^i . For each set of inputs, there is a set of memory section outputs after the transition at the memory section, for example, corresponding to each set of external inputs, there will be a set of next state outputs (Q'_1, Q'_2) in case of two FFs at the memory section.

Mealy Model

The number of columns for the output \underline{Y} also equals the number of possible 2^i combinations of external inputs in the set \underline{X} .

Moore Model

The number of column = 1 for the output \underline{Y} as in Moore model Y depends on Q s only.

7.5.3 State Table

A state table is a tabular representation of the present state in column 1 and next state after the transition in a set of succeeding columns. The outputs are shown in next set of columns. Number of rows in each column equals 2^m for a memory section of m flip-flops.

For example, if (Q_1, Q_2) are the Q s of two FFs, then let us assume the followings:

$$S(Q_1, Q_2) = S(0, 0) = S_0 \text{ for } (0, 0) \text{ values of } (Q_1, Q_2),$$

$$S(Q_1, Q_2) = S(0, 1) = S_1 \text{ for } (0, 1) \text{ values of } (Q_1, Q_2),$$

$$S(Q_1, Q_2) = S(1, 0) = S_2 \text{ for } (1, 0) \text{ values of } (Q_1, Q_2), \text{ and}$$

$$S(Q_1, Q_2) = S(1, 1) = S_3 \text{ for } (1, 1) \text{ values of } (Q_1, Q_2).$$

A state in a row on first column is written as either S_0, S_1, S_2 or S_3 for the row 1, row 2, row 3 or row 4, respectively. These are as per four possible values of (Q_1, Q_2) in case of two FFs used at the memory section.

The number of columns for the next state \underline{S} equals the number of possible combinations of external inputs in the set \underline{X} . It equals 2^i . For each set of inputs, there is a set of memory section states resulting after the transition at the memory section, for example, corresponding to each set of external inputs, there will be four sets of next states. A state in a row is either S_0 or S_1, S_2 or S_3 as per the post transition values of next state (Q'_1, Q'_2) in case of two FFs used at the memory section.

Moore Model

The number of columns for the output \underline{Y} also equals the number of possible 2^i combinations of external inputs in the set \underline{X} .

Moore Model

The number of column = 1 for the output \underline{Y} as in Moore model Y depends on Q_s only.

7.5.4 State Diagram

A state diagram is diagrammatic representation of the state table. A set of present Q_0, Q_1, \dots is denoted by a state. There are z ($= 2^m$) maximum possible states S_0, S_1, \dots, S_{z-1} in a sequential circuit with m -FFs.

1. The number of nodes = number of rows in the state table.
2. For two flip-flops, there are four states S_0, S_1, S_2 and S_3 . So four circles are drawn for the four nodes of a graph.
3. A directed arc from the present state node to the next state node shows a transition.
4. A small diameter circular directed arc marks a transition in which the state remains unchanged.
5. The number of directed arcs equals the number of transitions in which the state changes.
6. The number of directed circular arcs equals the number of transitions in which the state does not change.

Mealy Model

1. States S_0, S_1, S_2 and S_3 are then labeled at the centers of each circle representing a node.
2. Each arc or circular arc is labeled with present input and the output after the transition.
3. Each arc or circular arc can have more than one set of (pre-transition input/ post transition output) labeled on it if there are more than one sets of (pre-transition input/ post transition output) that are having the same transition from a node to another.

Moore Model

1. States S_0, S_1, S_2 and S_3 as per the present output are then labeled as (state name or representation/present output) at the centers of each circle representing a node.
2. Each arc or circular arc is labeled with present input (from Q_s).
3. Each arc or circular arc can have more than one set of (pre-transition input) labeled on it if there are more than one sets of (pre-transition input) that are having the same transition from a node to another.

7.6 CONDITIONS OF STATES EQUIVALENCY

State table may have equivalent pair of states.

Two states S_i and S_j are equivalent in a synchronized clocked sequential circuit, if both the following conditions are fulfilled:

1. The present outputs at S_i and S_j are identical for all possible combinations of the input variables X , and
2. The next states of S_i and S_j after the transitions are also identical for all possible combinations of the input variables X .

7.6.1 State Reduction and Minimization Procedure

State minimization is done by state reduction after finding the pair of equivalent states and then the equivalent classes of the states. The two sequential circuits after state reduction or minimization are said to be equivalent when producing an identical output sequences for the possible input sequences.

7.6.1.1 Equivalency by State Table Inspection

One method of finding equivalency is by finding, what the states, which have the identical present outputs for each possible combination of X and identical next states. For example S_i and S_j has same outputs for all X and have same set of next states for all X , then replace S_j by S_i in the state table. Further continue this process, till no further reduction is feasible.

Consider a state table in Table 7.1.

TABLE 7.1 State table for a Moore model sequential circuit-*i*

Present state		Next state after transition (Q'_1, Q'_2, Q'_3)				Present output Y
State	(Q_1, Q_2, Q_3)	Input $X_1, X_2 = (0, 0)$	Input $X_1, X_2 = (0, 1)$	Input $X_1, X_2 = (1, 0)$	Input $X_1, X_2 = (1, 1)$	
S_1	0, 0, 0	S_4	S_3	S_2	S_4	0
S_2	0, 0, 1	S_2	S_1	S_2	S_1	1
S_3	0, 1, 0	S_2	S_1	S_2	S_1	1
S_4	0, 1, 1	S_4	S_3	S_2	S_4	0
S_5	1, 0, 0	S_4	S_3	S_2	S_4	0
S_6	1, 0, 1	S_2	S_1	S_2	S_1	1
S_7	1, 1, 0	S_3	S_4	S_3	S_4	0
S_8	1, 1, 1	S_1	S_2	S_1	S_2	1

We find that output $Y=0$ is same for S_1, S_4, S_5 and S_7 . This fulfills condition 1 of equivalency. However, S_1 and S_7 have different next states for inputs $(0, 0), (0, 1)$. Hence condition 2 is not fulfilled and these two are not equivalent. However, S_1, S_4 and S_5 have same next states for all the four possible combination of the inputs: $(0, 0), (0, 1), (1, 0)$ and $(1, 1)$. Hence S_1, S_4 and S_5 are equivalent. We replace S_4 by S_1 in state table and reconstruct state table as in Table 7.2.

We find that output $Y=1$ is same for S_2, S_3, S_6 and S_8 . This fulfills condition 1 of equivalency. We however find that S_2, S_3 and S_6 have identical next states for

7.10 Digital Systems: Principles and Design

TABLE 7.2 State table for a Moore model sequential circuit-i

Present state		Next state after transition (Q'_1, Q'_2, Q'_3)				Present output Y
State	(Q_1, Q_2, Q_3)	Input $X_1, X_2 = (0, 0)$	Input $X_1, X_2 = (0, 1)$	Input $X_1, X_2 = (0, 1)$	Input $X_1, X_2 = (0, 1)$	
$S_1,$ $S_4,$ $S_5,$	$(0, 0, 0)$ $(0, 1, 1)$ and $(1, 0, 0)$	S_1	S_3	S_2	S_1	0
S_2	$0, 0, 1$	S_2	S_1	S_2	S_1	1
S_3	$0, 1, 0$	S_2	S_1	S_2	S_1	1
S_6	$1, 0, 1$	S_2	S_1	S_3	S_1	1
S_7	$1, 1, 0$	S_3	S_1	S_3	S_1	0
S_8	$1, 1, 1$	S_1	S_2	S_1	S_2	1

TABLE 7.3 State table for a Moore model sequential circuit-i

Present state		Next state after transition (Q'_1, Q'_2, Q'_3)				Present output Y
State	(Q_1, Q_2, Q_3)	Input $X_1, X_2 = (0, 0)$	Input $X_1, X_2 = (0, 1)$	Input $X_1, X_2 = (0, 1)$	Input $X_1, X_2 = (0, 1)$	
$S_1,$ $S_4,$ $S_5,$	$(0, 0, 0)$ $(0, 1, 1)$ and $(1, 0, 0)$	S_1	S_2	S_2	S_1	0
$S_2,$ $S_3,$ $S_6,$	$(0, 0, 1)$ $(0, 1, 0)$ and $(1, 0, 1)$	S_2	S_1	S_2	S_1	1
S_7	$1, 1, 0$	S_2	S_1	S_2	S_1	0
S_8	$1, 1, 1$	S_1	S_2	S_1	S_2	1

inputs. Now replace S_3 by S_2 in the state table everywhere we get the Table 7.2 and also delete the rows for S_3 and S_8 . We reconstruct Table 7.2 as Table 7.3.

None of the rows have Y and next states for the inputs both identical. Therefore, Table 7.3 is a state minimal table. Next section gives another procedure for a systemic determination of equivalency of pairs and solution of state minimization problem.

7.6.1.2 Equivalency Determination and Minimization of States in State Table by a Procedure of Constructing Implication Table

Following are the steps for building an implication table for synthesis of the circuit after finding the reduced number of states and state minimal table.

Step 1: Construct an implication table as shown in Table 7.14 (Example 7.7) for a five state state-table.

Step 2: Mark the state pair *not to be considered* for equivalency determination in a matrix of cell as follows:

A (S_i, S_j) cell is redundant for equivalency determination and an equivalent state pair (S_p, S_q) is same as pair (S_j, S_i) . In a matrix of $n \times n$ cells, $(n^2 - n)/2 = 10$ are the off-diagonal right side cells when $n = 5$ are the cells along the diagonal. Hence total 15 cells are not to be taken into consideration for pairing at the implication table for state table for five states. Cells marked by sign \wedge mark are not to be considered.

For example, we put \wedge sign in 15 cells in the implication table $(S_1, S_1), (S_2, S_2), (S_2, S_1), (S_3, S_3), (S_3, S_2), (S_3, S_1), (S_4, S_4), (S_4, S_3), (S_4, S_2), (S_4, S_1), (S_5, S_5), (S_5, S_4), (S_5, S_3), (S_5, S_2)$ and (S_5, S_1) .

Step 3: Also mark and put a # sign at a cell not having the same set of output for all the input combinations.

Step 4: Fill the unmarked cells by the next state values. For example, if (S_2, S_4) cell is unmarked then put entries of the next state pairs for each combination of X s in this cell.

First iteration of cells filling is now over. Now next iteration on marking implication tables is as follows and Table 7.15 shows the result in Example 7.7.

Step 5: Discard the cells not generating equivalent next states (not fulfilling condition 2) by following process of further marking of # signs.

Let an entry be (S_p, S_m) in a cell (S_i, S_j) . If cell (S_p, S_m) has a # sign previously placed, then put # sign in cell (S_i, S_j) also. Now ignore any other entry in the cell (S_i, S_j) , else look at other entry(ies) also in the cell (S_i, S_j) .

Continue this process for all those cells not having a # sign by now.

Step 6: Continue process in Step 5 till no more # sign needs to be placed. All the cells not generating equivalent next states (not fulfilling condition 2) are now free from # sign.

From these cells, which has neither # nor \wedge sign, build a new state table and build state minimal table by grouping the states that are equivalent after the above state minimization process.

Example 7.7 and 7.8 explain these steps.

7.6.2 Assignment of Variables to a State

Let there are five states in a state diagram for the three FFs. Then there are five rows one each for a in a state table. Assuming that one FF can undergo only one change of states, we need eight state assignments for a binary representation of a state at the output of three FFs. Even if number of states are eight, we need minimum three bit binary number to represent in order to assign a state at the memory section of the given clocked sequential circuit.

We can assign three binary numbers to the five states. However by increasing the number of binary bits to assign in a state table with 5 rows, it may also be possible to actually reduce the total number of gates. Suppose there are three binary

bits for each state. There are $(2^3)! / (2^3 - 5)! = 8 \times 7 \times 6 \times 5 \times 4$ assignment ways to code the given states.

Few guidelines are as follows: (a) Two or more states at present giving the same output \underline{Y} for a given \underline{X} should be made adjacent. (b) Make two or more states adjacent in case they have a same next state for a given input set. (c) Make two next states adjacent for a present state and two input combinations that are adjacent.

For a Moore model sequential clocked circuit, the state assignment can be made same as number of outputs.

Assuming the five rows only in a state table, Figure 7.4 (a) shows a state assignment map for maximum possible eight and figure 7.4(b) for sixteen states.

7.7 IMPLEMENTATION PROCEDURE

Circuit synthesis or implementation and designing steps are opposite to that of analysis (Section 7.5). The procedure for implementation is as follows:

Q_3	\bar{Q}_1	\bar{Q}_3		
$Q_1 Q_2$	0	1		
$\bar{Q}_1 \bar{Q}_2$	00	S_1	S_3	
$\bar{Q}_1 \bar{Q}_2$	01	S_2	S_4	
$\bar{Q}_1 \bar{Q}_2$	11	S_5		
$\bar{Q}_1 \bar{Q}_2$	00			

$Q_3 Q_4$	$\bar{Q}_3 \bar{Q}_4$	$\bar{Q}_3 \bar{Q}_4$	$Q_3 Q_4$	$Q_3 \bar{Q}_4$		
$Q_1 Q_2$	0 0	0 1	0 1	0 1		
00	S_1	S_3	S_2	S_4		
01	S_5					
11						
11						

FIGURE 7.4 (a) A state assignment map for maximum 8 rows (states) in a state table (b) A state assignment with maximum 16 rows in a state table.

1. Get specifications of the sequential circuit.
2. Derive the states and draw the state diagram.
3. Make a state table.
4. Perform state minimization so that there are minimum number of rows in the table.
5. Make a transition table and find the state variables number of rows in the table and excitation variables needed.
6. Find the flip flop characteristics expressions, which implement the transitions as per the table.
7. Implement the circuit as per the expressions.

Example 7.9 will explain the circuit implementation.

■ EXAMPLES

Example 7.1

Define a Mealy Machine for a general sequential circuit. Why can a Mealy machine produce pulsed outputs? Why is it that a Mealy machine can produce false outputs and glitches?

Solution

- (1) Mealy machine can be defined as a machine consisting of set of $(\underline{X}, \underline{Q}, \underline{Y}, F_Q$ and $F_Y)$, where \underline{X} is non-empty finite set of inputs, \underline{Q} is non-empty finite set of states, \underline{Y} is non-empty finite set of outputs, the F_Q is state transition function changing \underline{Q} as a function of \underline{X} and \underline{Q} , and the F_Y is an output function of states \underline{Q} and inputs \underline{X} for producing \underline{Y} .
- (2) Since F_Y is an output function of states \underline{Q} and inputs \underline{X} for producing \underline{Y} and changes in \underline{X} causes the changes in \underline{Q} later to \underline{Q}' , therefore the transition produces a pulsed \underline{Y} after the next steady state output \underline{Q}' . This is because the time taken in transition at the memory section is different than the time taken at the combinational section for inputs to the memory section.
- (3) The pulses are also called glitches for the outputs between two intervals, the transition instance and the excitation instance.
- (4) The outputs can be false during the period before the excitation inputs are applied as these can change and are unstable after the \underline{X} is applied. False outputs can also occur due to propagation delay at the memory section.

Example 7.2

Define a Moore machine for a general sequential circuit. Why can a Moore machine model sequential circuit produce constant steady state outputs?

Solution

Moore machine can be defined as a machine consisting of set of $(\underline{X}, \underline{Q}, \underline{Y}, F_Q$ and $F_Y)$, where \underline{X} is non-empty finite set of inputs, \underline{Q} is non-empty finite set of state, \underline{Y} is non-empty finite set of outputs, the F_Q is state transition function changing \underline{Q} as a function of \underline{X} and \underline{Q} but the F_Y is an output function of states \underline{Q} only for producing \underline{Y} .

Since F_Y is an output function of present state \underline{Q} only, \underline{Y} is a steady state output as per the current state only.

Example 7.3

Analyze the sequential circuit-3 shown in Figure 7.3(a).

Solution

Let us analyze the circuit-3 in Figure 7.3(a) by the six-step procedure mentioned in Section 7.5 above.

Step 1: Logic circuit drawing:

Logic circuit drawing is as per Figure 7.3(a).

Step 2: Performing state variables assignments and excitation (FF triggering) variables assignments:

State variables in the circuit are Q_2 , \bar{Q}_2 , Q_1 and \bar{Q}_1 at the lower and upper FFs, respectively.

Excitation variables are D_1 and D_2 .

Step 3: Finding the expressions for the excitations from the flip flop characteristics equations as per the excitations and make an excitation table: Therefore,

the excitation expressions for next state Q_s s are as follows:

$$Q'_1 = D_1 \quad \dots(7.1)$$

$$Q'_2 = D_2 \quad \dots(7.2)$$

Now, the steps for finding the transition equation are the followings using the input variables for the excitations of the FFs. Two expressions for the excitation inputs for the Figure 7.3(a) combinational circuit are as follows:

$$D_1 = (X \cdot \bar{Q}_1 \cdot \bar{Q}_2 + \bar{X} \cdot Q_1 \cdot \bar{Q}_2) \text{ and} \quad \dots(7.3)$$

$$D_2 = X \cdot Q_1 \quad \dots(7.4)$$

Therefore the next state after the state transition equations are as per the excitation expression will be given by

$$Q'_1 = (X \cdot \bar{Q}_1 \cdot \bar{Q}_2 + \bar{X} \cdot Q_1 \cdot \bar{Q}_2) \text{ and} \quad \dots(7.5)$$

$$Q'_2 = X \cdot Q_1 \quad \dots(7.6)$$

The output equation for the Y is as follows:

$$Y = Q_1 \cdot Q_2 \quad \dots(7.7)$$

As Y depends on Q_s s only, the present circuit is a Moore model sequential circuit. Using the equations (7.3) and (7.4) and (7.7), Table 7.4 gives the excitation table for present states, excitation inputs and the present output Y [Post state transition $Y = Q'_1, Q'_2$].

Step 4: Make transition table from the expressions for $\underline{Y} = F_o(\underline{Q})$ and \underline{Q}' .

TABLE 7.4 Excitation table for the sequential circuit-3

Present state	Excitation inputs (D_1, D_2)		Present output state Y before the transition
(Q_1, Q_2)	Input $X = 0$	Input $X = 1$	
0, 0	0, 0	1, 0	0
0, 1	0, 0	0, 0	0
1, 0	1, 0	1, 1	0
1, 1	0, 0	0, 1	1

TABLE 7.5 Transition table for the sequential circuit-3

Present state	Next state after transition (Q'_1, Q'_2)		Present output state Y before the transition
(Q_1, Q_2)	Input $X = 0$	Input $X = 1$	
0, 0	0, 0	1, 0	0
0, 1	0, 0	0, 0	0
1, 0	1, 0	1, 1	0
1, 1	0, 0	0, 1	1

Table 7.5 gives the transition table made by using the Equations (7.5) and (7.6) and (7.7).

It is same as Table 7.6 because a D FFs reflects the D at Q .

Step 5: Perform state minimization and make minimal state table.

TABLE 7.6 State table for the sequential circuit-3

Present state		Next state after transition (Q_1' , Q_2')		Present output state Y before the transition
State	(Q_1 , Q_2)	Input $X = 0$	Input $X = 1$	
S_0	(0, 0)	$S_0 [= (0, 0)]$	S_2	0
S_1	(0, 1)	$S_0 [= (0, 0)]$	S_0	0
S_2	(1, 0)	$S_2 [= (1, 0)]$	S_3	0
S_3	(1, 1)	$S_0 [= (1, 0)]$	S_1	1

There are four possible states (Q_1 , Q_2). These are (0, 0), (0, 1), (1, 0) and (1, 1). Let us designate (assign) these as S_0 , S_1 , S_2 and S_3 . We therefore get the state table as follows. Table 7.6 gives the state table made from the Table 7.5.

Reason for next state in column 3 as S_0 , S_0 , S_2 and S_0 when $X = 0$ is also mentioned to clarify to the reader how to make the state table. Normally the reason is not given, just the states are mentioned in the columns showing the state transitions.

Step 6: Draw the state diagram

From the state table the state diagram is drawn. Figure 7.5(a) shows the state diagram for the sequential circuit-3. The directed arc represents a transition. It is labeled with present input and the output after the transition. This is clarified in the followings for each state at the rows of the table.

Consider row 1 of state table in Table 7.6.

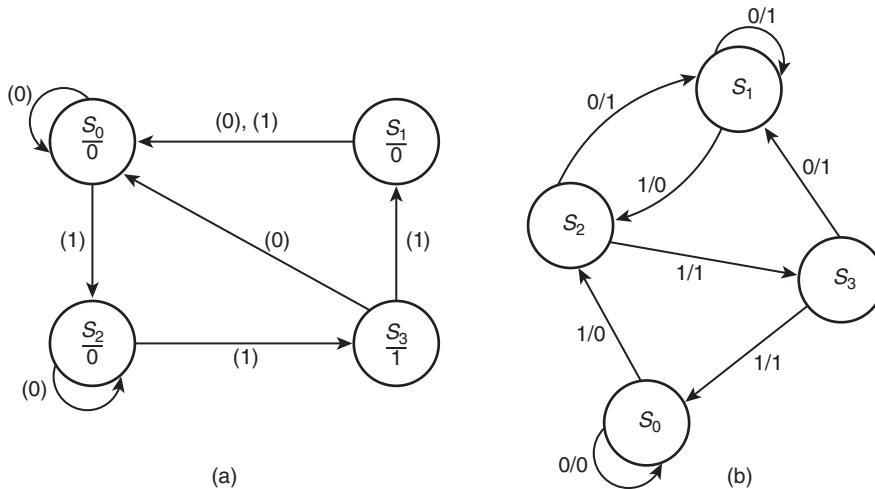


FIGURE 7.5 (a) State diagram for the sequential circuit-3 (b) State diagram for the sequential circuit-4

1. Transition is from S_0 to S_0 itself when input $X=0$ and output is $Y=0$. Node is labeled at the center by $S_0/0$ for the node S_0 in Figure 7.5(a). The arc is labeled 0.
2. Transition is from S_0 to S_2 when input $X=1$ and output is $Y=0$. Hence the directed arc in Figure 7.5(a) is from S_0 to S_2 and is also labeled 1.

Consider **row 2** of state table in Table 7.6.

3. Transition is from S_1 to S_0 when input $X=0$ and output is $Y=0$. Hence the directed arc in Figure 7.5(a) is from S_1 to S_0 and is labeled 0. Node for state S_1 is labeled at the center by $S_1/0$.
4. Transition is from S_1 to S_0 when input $X=1$ and output is $Y=0$. Hence the directed arc in Figure 7.5(a) is from S_1 to S_0 and is also labeled (1).

Since for $X=0$ and $X=1$ the directed arcs are identical, we label S_1 to S_0 arc as (0), (1).

Consider **row 3** of state table in Table 7.6.

5. Transition is from S_2 to S_2 itself when input $X=0$ and output is $Y=0$. Hence a directed circular arc is shown at the node S_2 in Figure 7.5(a). It is labeled 0. Node for S_2 is labeled at the center by $S_2/0$.
6. Transition is from S_2 to S_3 when input $X=1$ and output is $Y=0$. Hence the directed arc in Figure 7.5(a) is from S_2 to S_3 and is also labeled 1.

Consider **row 4** of state table in Table 7.6.

7. Transition is from S_3 to S_0 when input $X=0$ and output is $Y=1$. Hence a directed arc is shown from the node S_3 to node S_0 in Figure 7.5(a). It is labeled 0. $S_3/1$ is the label at state S_3 node is placed at the center.
8. Transition is from S_3 to S_1 when input $X=1$ and output is $Y=1$. Hence the directed arc in Figure 7.5(a) is from S_3 to S_1 and is also labeled 1. $S_3/1$ is the label at the node center because $Y=1$ at S_3 .

Example 7.4 Analyze the sequential circuit-4 shown in Figure 7.3(b).

Solution

Let us analyze the circuit-4 in Figure 7.3(b) by the six-steps procedure mentioned in Section 7.5 above.

Step 1: Logic circuit drawing:

Logic circuit drawing is as per Figure 7.3(b).

Step 2: Performing state variables assignments and excitation (*FF triggering*) variables assignments:

State variables in the circuit are Q_2 , \bar{Q}_2 , Q_1 and \bar{Q}_1 at the lower and upper FFs, respectively.

Excitation variables are D , J and K .

Step 3: Finding the expressions for the excitations from the flip flop characteristics equations as per the excitations and make an excitation table:

Therefore the excitation expressions are as follows:

$$Q'_1 = D \quad \dots(7.8)$$

From Equation 6.3 for the *JK FF*, we get the following equation.

$$Q_2' = J \cdot \bar{Q}_2 + \bar{K} \cdot Q_2 \quad \dots(7.9)$$

Now, the steps for finding the transition equations are the followings using the input variables for the excitations of the FFs.

Two expressions for the combinational circuit for the excitation inputs are as follows:

$$D = X \cdot (\bar{Q}_1 + \bar{Q}_2) \quad \dots(7.10)$$

$$J = Q_1 \text{ and } K = X \quad \dots(7.11)$$

Therefore, the next state after the state transition equations are as per the excitation expression and are given by

$$Q_1' = X \cdot (\bar{Q}_1 + \bar{Q}_2) \text{ and} \quad \dots(7.12)$$

$$Q_2' = Q_1 \cdot \bar{Q}_2 + \bar{X} \cdot Q_2 \quad \dots(7.13)$$

The output equation for the Y is as follows:

$$Y = \bar{X} \cdot Q_2 + Q_1 \quad \dots(7.14)$$

Since combinational circuit inputs at the memory section outputs Y is such that the Y depend on X , we have a circuit, which is a Mealy model sequential circuit. Using the equations (7.10) and (7.11) and (7.14), Table 7.7 gives the excitation table for present states, excitation inputs and the present output Y .

TABLE 7.7 Excitation table for the for the sequential circuit-4

Present state	Excitation inputs $D, (J, K)$		Present output state after the X inputs but before the transition	
(Q_1, Q_2)	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$
0, 0	0, (0, 0)	1, (0, 1)	0	0
0, 1	0, (0, 0)	1, (0, 1)	1	0
1, 0	0, (1, 0)	1, (1, 1)	1	1
1, 1	0, (1, 0)	0, (1, 1)	1	1

Step 4: Make transition table from the expressions for $Y = F_o(X, Q')$ and Q' .

Table 7.8 gives the transition table made by using the equations (7.12) and (7.13) and (7.14).

TABLE 7.8 Transition table for the sequential circuit-4

Present state	Next state after transition (Q_1', Q_2')		Output state Y	
(Q_1, Q_2)	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$
0, 0	0, 0	1, 0	0	0
0, 1	0, 1	1, 0	1	0
1, 0	0, 1	1, 1	1	1
1, 1	0, 1	0, 0	1	1

Step 5: Perform state minimization and make minimal state table.

There are four possible states (Q_1, Q_2) : These are (0, 0), (0, 1), (1, 0) and (1, 1).

TABLE 7.9 State table for the sequential circuit-4

Present state		Next state after transition (Q_1' , Q_2')		Output state Y	
State	(Q_1 , Q_2)	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$
S_0	0, 0	S_0	S_2	0	0
S_1	0, 1	S_1	S_2	1	0
S_2	1, 0	S_1	S_3	1	1
S_3	1, 1	S_1	S_0	1	1

Let us designate (assign) these as S_0 , S_1 , S_2 and S_3 . We therefore make the state table. Table 7.9 gives the state table made from the Table 7.8.

Step 6: Draw the state diagram.

From the state table the state diagram is drawn. Figure 7.5(b) shows the state diagram for the sequential circuit-4.

Since the circuit is a Mealy model sequential circuit, the nodes S_0 , S_1 , S_2 and S_3 are labeled at the center of a circle. The directed arc represents a transition. It is labeled with present input and the output after the transition. It is clarified in the following for a state present at each row.

Consider **row 1** of state table in Table 7.9.

1. Transition is from S_0 to S_0 itself when input $X=0$ and output is $Y=0$. Hence a directed circular arc is shown for the node S_0 in Figure 7.5(b). It is labeled 0/0.
2. Transition is from S_0 to S_2 when input $X=1$ and output is $Y=0$. Hence the directed arc in Figure 7.5(b) is from S_0 to S_2 and is also labeled 1/0.

Consider **row 2** of state table in Table 7.9.

3. Transition is from S_1 to S_1 itself when input $X=0$ and output is $Y=1$. Hence the directed circular arc in Figure 7.5(b) is shown at the node S_1 and is labeled 0/1.
4. Transition is from S_1 to S_2 when input $X=1$ and output is $Y=0$. Hence the directed arc in Figure 7.5(b) is from S_1 to S_2 and is also labeled 1/0.

Consider **row 3** of state table in Table 7.9.

5. Transition is from S_2 to S_1 when input $X=0$ and output is $Y=1$. Hence a directed arc is shown from the node S_2 to S_1 in Figure 7.5(b). It is labeled 0/0.
6. Transition is from S_2 to S_3 when input $X=1$ and output is $Y=1$. Hence the directed arc in Figure 7.5(b) is from S_2 to S_3 and is also labeled 1/1.

Consider **row 4** of state table in Table 7.9.

7. Transition is from S_3 to S_1 itself when input $X=0$ and output is $Y=1$. Hence a directed arc is shown from the node S_3 to node S_0 in Figure 7.5(b). It is labeled 0/1.
8. Transition is from S_3 to S_0 when input $X=1$ and output is $Y=1$. Hence the directed arc in Figure 7.5(b) is from S_3 to S_0 and is also labeled 1/1.

Example 7.5

When are the two states equivalent in a clocked sequential circuit?

Solution

Two states S_i and S_j are equivalent in a synchronized clocked sequential circuit, if both the following conditions are fulfilled:

1. The present outputs at S_i and S_j are identical for all possible combinations of the input variables X .
2. The next states of S_i and S_j after the transitions are also identical for all possible combinations of the input variables X .

Example 7.6

Perform state minimization for the state table (Table 7.10) for a Mealy model sequential circuit- k by inspection procedure.

TABLE 7.10 State table for a Mealy model sequential circuit- k

Present state		Next state after transition (Q'_1, Q'_2, Q'_3)			Present output Y	
State	(Q_1, Q_2, Q_3)	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$	
S_1	0, 0, 0	S_1	S_4	0	1	
S_2	0, 0, 1	S_2	S_4	0	0	
S_3	0, 1, 0	S_4	S_3	0	1	
S_4	0, 1, 1	S_1	S_4	0	1	
S_5	1, 0, 0	S_2	S_4	0	0	
S_6	1, 0, 1	S_2	S_1	1	1	
S_7	1, 1, 0	S_1	S_3	1	1	
S_8	1, 1, 1	S_2	S_1	0	0	

S_1 and S_4 have all values of Q' and therefore, post transition Y_s same for the inputs $X = 0$ and 1. We can replace S_4 by S_1 . S_2 and S_5 states have all values of Q' same for all the inputs and have same next states S_2 and S_4 . Reduced six rows at table are now as follows:

TABLE 7.11 Reduced state table for a Mealy model sequential circuit- k

Present state		Next state after transition (Q'_1, Q'_2, Q'_3)			Present output Y	
State	(Q_1, Q_2, Q_3)	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$	
$S_1,$ S_4 and $(0, 1, 1)$	(0, 0, 0)	S_1	S_1	0	1	
$S_2,$ S_5 and $(1, 0, 0)$	(0, 0, 1)	S_2	S_1	0	0	
S_3	0, 1, 0	S_1	S_3	0	1	
S_6	1, 0, 1	S_2	S_1	1	1	
S_7	1, 1, 0	S_1	S_3	1	1	
S_8	1, 1, 1	S_2	S_1	0	0	

7.20 Digital Systems: Principles and Design

S_3 and S_7 now have all values of post transition Y same for the inputs $X = 0$ and 1 . We can replace S_4 by S_1 . Therefore, reduced table of 4 rows is now as follows:

TABLE 7.12 New reduced state table for a Mealy model sequential circuit-k

Present state		Next state after transition (Q'_1, Q'_2, Q'_3)			Present output Y	
State	(Q_1, Q_2, Q_3)	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$	
S_1, S_4	(0, 0, 0) and (0, 1, 1)	S_1	S_1	0	1	
S_2, S_5	(0, 0, 1) and (1, 0, 0)	S_2	S_1	0	0	
S_3, S_7	(0, 1, 0) and (1, 1, 0)	S_1	S_3	0	1	
S_6, S_8	1, 0, 1 1, 1, 1	S_2	S_1	1	1	
		S_2	S_1	0	0	

Table 7.12 is now a minimized state table for the circuit - k. Since the numbers of rows = 5, an implementation will still need three flip flops as two flip flops can implement only four states.

Example 7.7 Perform state reduction or minimization of state table in Table 7.13, if feasible.

Solution

Consider the state table for a Mealy model sequential circuit-l in Table 7.13.

TABLE 7.13 State table for a Mealy model sequential circuit-l

Present state		Next state after transition (Q'_1, Q'_2, Q'_3)			Present output Y	
		Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$	
S_1	S_2	S_1		1	1	
S_2	S_1	S_3		0	1	
S_3	S_4	S_1		1	1	
S_4	S_4	S_5		0	1	
S_5	S_3	S_4		0	0	

Step 1: Construct an implication table as shown in Table 7.14 for a five states state table.

Step 2: Marked the state pairs not to be considered for equivalency determination in matrix of cells.

A pair (S_i, S_j) is redundant for equivalency determination and an equivalent state transition pair (S_i, S_j) is same as pair (S_j, S_i) . In a matrix of $n \times n$ cells, $(n^2 - n)/2 = 10$ are the off diagonal right side cells when $n = 5$ are the cells along the diagonal. We put ^ sign at the 15 cells in implication table $(S_1, S_1), (S_2, S_2), (S_2, S_1), (S_3, S_3),$

(S_3, S_2) , (S_3, S_1) , (S_4, S_4) , (S_4, S_3) , (S_4, S_2) , (S_4, S_1) , (S_5, S_5) , (S_5, S_4) , (S_5, S_3) , (S_5, S_2) and (S_5, S_1) .

Step 3: Put # sign for a cell not having the same set of Y outputs for all the input combinations for X . Table 7.14 state table shows that for a set of inputs, which are possible, the Y outputs are same $Y(1, 1)$ only for $(S_2$ and $S_4)$ states and same $Y(0, 1)$ for $(S_1$ and $S_3)$ states. Except in cells for (S_1, S_3) pair and for (S_2, S_4) pair, we therefore mark remaining cells by # sign.

Step 4: Fill the entries at the unmarked cells by the next-state values from the state table. It means fill (S_2, S_4) pair for $X=0$ and (S_1, S_1) pair for $X=1$ in the cell for (S_1, S_3) and fill (S_1, S_4) for $X=0$ and (S_3, S_5) for $X=1$ in cell for (S_2, S_4) .

TABLE 7.14 Implication table first iteration for a Mealy model sequential circuit-/

	S_1	S_2	S_3	S_4	S_5
S_5	#	#	#	#	\wedge
S_4	#	$(S_1, S_4) (S_3, S_5)$	#	\wedge	\wedge
S_3	$(S_2, S_4) (S_1, S_1)$	#	\wedge	\wedge	\wedge
S_2	#	\wedge	\wedge	\wedge	\wedge
S_1	\wedge	\wedge	\wedge	\wedge	\wedge

First iteration of cells filling is now over. Now next iteration on marking at the implication table is as follows and Table 7.15 shows the result.

Step 5: Remove the cell pairs not generating equivalent next states (not fulfilling condition 2).

Consider cell for (S_1, S_3) . It has entries (S_2, S_4) for $X=0$ and (S_1, S_1) for $X=1$. None of the paired cells have # sign. Hence leave this entry as such.

Consider cell for (S_2, S_4) . It has entries (S_1, S_4) for $X=0$ and (S_3, S_5) for $X=1$. The paired cell (S_3, S_5) there is # sign. Hence, place the # sign in this cell also as this pair does not fulfill condition 2.

TABLE 7.15 Implication table second iteration for a Mealy model sequential circuit-/

	S_1	S_2	S_3	S_4	S_5
S_5	#	#	#	#	\wedge
S_4	#	$(S_1 S_4) \# \}$ $(S_3 S_5) \# \}$	#	\wedge	\wedge
S_3	$(S_2, S_4) \}$ $(S_1, S_1) \}$	#	\wedge	\wedge	\wedge
S_2	#	\wedge	\wedge	\wedge	\wedge
S_1	\wedge	\wedge	\wedge	\wedge	\wedge

Second iteration of cells filling is now over. Only cell pair (S_1, S_3) is left without any sign. Now next iteration on marking implication tables is as follows and Table 7.16 shows the result.

Step 6: Again remove the cell pairs not generating equivalent next states (not fulfilling condition 2).

Consider cell for (S_1, S_3) . It has entries (S_2, S_4) for $X = 0$ and (S_1, S_1) for $X = 1$. Now in this iteration (S_2, S_4) paired cell has # sign. Hence we put the # sign here also.

TABLE 7.16 Implication table third iteration for a Mealy model sequential circuit-l

	S_1	S_2	S_3	S_4	S_5
S_5	#	#	#	#	\wedge
S_4	#	$(S_1, S_3) \} \#$ $(S_4, S_5)) \}$	#	\wedge	\wedge
S_3	$(S_2, S_4) \} \#$ $(S_1, S_1) \} \#$	#	\wedge	\wedge	\wedge
S_2	#	\wedge	\wedge	\wedge	\wedge
S_1	\wedge	\wedge	\wedge	\wedge	\wedge

All cell pairs have a sign. Hence none of the states is found equivalent in circuit-l.

Example 7.8 Perform state reduction or minimization of state table in Table 7.17, if feasible. Use implication table approach.

Solution

Consider the state table for a Moore model sequential circuit-m in Table 7.17.

TABLE 7.17 State table for a Moore model sequential circuit-m

Present state	Next state after transition (Q_1', Q_2', Q_3')		Present output Y
	Input $X = 0$	Input $X = 1$	
S_1	S_2	S_3	0
S_2	S_4	S_5	1
S_3	S_1	S_6	1
S_4	S_5	S_3	1
S_5	S_7	S_8	0
S_6	S_2	S_8	0
S_7	S_4	S_6	1
S_8	S_6	S_5	0

Step 1: Construct an implication table as shown in Table 7.18 for a five states state-table.

Step 2: Mark the cells for state pairs not to be considered for equivalency determination in matrix of cell-pairs:

A (S_i, S_j) pair is redundant for equivalency determination and an equivalent state pair (S_i, S_j) is same as pair (S_j, S_i) . In a matrix of $n \times n$ cells, $(n^2 - n)/2 = 28$ are the off diagonal right side cells and $n = 8$ are the cells along the diagonal.

Hence total 36 cells are not to be taken into consideration for pairing at the implication table for eight states table. Cells for the pairs marked by sign \wedge marked are not to be considered. We put \wedge sign in 36 cells in the implication table.

Step 3: Put # sign for a cell pair, which is not having the same set of Y output. Table 7.18 state table shows that for a set of inputs, which are possible, the outputs are same $Y=1$ for $(S_2 \text{ and } S_7)$ states and same $Y=0$ for $(S_5 \text{ and } S_6)$ states. We also mark the remaining cells by # sign, which do not show same Y .

Step 4: Fill the unmarked cells by the next state values. For example, (S_2, S_6) and (S_3, S_5) entries in cell (S_1, S_8) .

TABLE 7.18 Implication table first iteration for a Moore model sequential circuit-m

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
S_8	$(S_2, S_6),$ (S_3, S_5)	#	#	#	(S_6, S_7) (S_5, S_8)	(S_2, S_6) (S_5, S_8)	#	\wedge
S_7	#	$(S_4, S_4),$ (S_5, S_6)	$(S_1, S_4),$ (S_6, S_6)		#	#	\wedge	\wedge
S_6	(S_2, S_2) (S_3, S_8)	#	#	#	(S_2, S_7) (S_8, S_8)	\wedge	\wedge	\wedge
S_5	(S_2, S_7) (S_3, S_8)	#	#	#	\wedge	\wedge	\wedge	\wedge
S_4	#	(S_4, S_5) (S_3, S_5)	(S_1, S_5) (S_3, S_6)		\wedge	\wedge	\wedge	\wedge
S_3	#	(S_1, S_4) (S_5, S_6)		\wedge	\wedge	\wedge	\wedge	\wedge
S_2	#		\wedge	\wedge	\wedge	\wedge	\wedge	\wedge
S_1	\wedge	\wedge	\wedge	\wedge	\wedge	\wedge	\wedge	\wedge

First iteration of filling and marking of the cells is now over. Now next iteration on marking at the implication table is as follows and Table 7.19 shows the new result.

Step 5: Remove the cell pairs not generating equivalent next states (not fulfilling condition 2):

Consider cell for (S_2, S_7) . It has entries $(S_4, S_4), (S_5, S_6)$. We mark # for the cells according to following rule: "Let an entry be (S_p, S_m) in a cell (S_i, S_j) . If cell (S_p, S_m) has a # sign previously placed, then put # sign in cell (S_i, S_j) also. Now ignore any other entry in the cell (S_i, S_j) , else look at other entry(ies) also in the cell (S_i, S_j) ."

TABLE 7.19 Implication table second iteration for a Moore model sequential circuit-m

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
S_8	$(S_2, S_6), #$ (S_3, S_5)	#	#	#	$(S_6, S_7), #$ (S_5, S_8)	$(S_2, S_6) #$ (S_5, S_8)	#	\wedge
S_7	#	$(S_4, S_4),$ (S_5, S_6)	$(S_1, S_4), #$ (S_6, S_6)		#	#	\wedge	\wedge

7.24 Digital Systems: Principles and Design

S_6	(S_2, S_2) $(S_3, S_8)\#$	#	#	#	(S_2, S_7) (S_8, S_8)	^	^	^
S_5	$(S_2, S_7)\#$ (S_3, S_8)	#	#	#	^	^	^	^
S_4	#	$(S_4, S_5)\#$ (S_3, S_5)	(S_1, S_5) $(S_3, S_6)\#$	^	^	^	^	^
S_3	#	$(S_1, S_4)\#$ (S_5, S_6)	^	^	^	^	^	^
S_2	#	^	^	^	^	^	^	^
S_1	^	^	^	^	^	^	^	^

Second iteration of cells filling is now over. Only cell pair (S_2, S_7) and (S_5, S_6) are left without any sign. Now next iteration marking of implication table is as follows and Table 7.20 shows the result.

Step 6: Again remove the cell pairs not generating equivalent next states (not fulfilling condition 2).

Table 7.20 Implication table third iteration for a Moore model sequential circuit-m

S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	
S_8	$(S_2, S_6), \#$ (S_3, S_5)	#	#	#	$(S_6, S_7)\#$ (S_5, S_8)	$(S_2, S_6) \#$ (S_5, S_8)	#	^
S_7	#	$(S_4, S_4),$ (S_5, S_6)	$(S_1, S_4), \#$ (S_6, S_6)	#	#	#	^	^
S_6	(S_2, S_2) $(S_3, S_8)\#$	#	#	#	(S_2, S_7) (S_8, S_8)	^	^	^
S_5	$(S_2, S_7)\#$ (S_3, S_8)	#	#	#	^	^	^	^
S_4	#	$(S_4, S_5) \#$ (S_3, S_5)	(S_1, S_5) $(S_3, S_6)\#$	^	^	^	^	^
S_3	#	$(S_1, S_4) \#$ (S_5, S_6)	^	^	^	^	^	^
S_2	#	^	^	^	^	^	^	^
S_1	^	^	^	^	^	^	^	^

Both (S_5, S_6) and (S_2, S_7) pair cells have no # sign. So this iteration results in no change in Table 7.19. All cell have a sign # or ^ except (S_2, S_7) cells and (S_5, S_6) cells.

We get the following implication table as Table 7.21 for equivalency determination.

TABLE 7.21 Implication table equivalency cells for a Moore model sequential circuit-m

S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
S_7	$(S_4, S_4),$ (S_5, S_6)						
S_6				(S_2, S_7) (S_8, S_8)			

We find $S_2 = S_7$, and $S_5 = S_6$. Remaining states do not appear here. Table 7.22 gives the state minimal table.

TABLE 7.22 State table for a Moore model sequential circuit-*m*

Present state	Next state after transition (Q_1' , Q_2' , Q_3')		Present output Y
	Input $X = 0$	Input $X = 1$	
S_1	S'	S_3	0
$(S_2, S_7) = S'$	S_4	S''	1
S_3	S_1	S''	1
S_4	S''	S_3	1
$(S_5, S_6) = S''$	S'	S_8	0
S_8	S''	S''	0

Example 7.9

Make state table, and transition tables for a clocked sequential circuit-*n* from the state diagram given in Figure 7.6(a).

Solution

For the state diagram in Figure 7.6, we find that there are four states S_0 , S_1 , S_2 and S_3 . Therefore the number of rows in state, transition and excitation tables shall also be four.

The diagram has present/present output on the directed arc. It means we will be having Mealy sequential circuit. It means output *Y*s for different values of *X*.

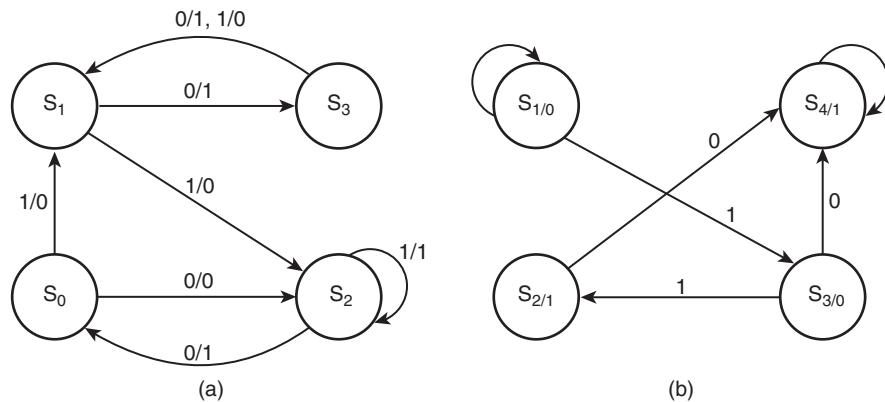
Now, there is only one external input *X*. Hence there will be two combinations, $X = 0$ and $X = 1$ for the present output *Y* in each of the tables: state, transition and excitation.

Now, there is only one external input *X*. Hence there will be two combinations, $X = 0$ and $X = 1$ for the next state and excitation inputs in tables for the state, transition and excitation, respectively.

Table 7.23 gives a state table for the state diagram in Figure 7.6(a).

TABLE 7.23 State table for the sequential circuit-*n*

Present state	Next state after transition (Q_1' , Q_2')		Present output state Y before the transition	
State (Q_1 , Q_2)	Input $X = 0$	Input $X = 1$	$X = 0$	$X = 1$
S_0 0, 0	S_2	S_1	0	0
S_1 0, 1	S_3	S_2	1	0
S_2 1, 0	S_0	S_2	1	1
S_3 1, 1	S_1	S_1	1	0

FIGURE 7.6 (a) State diagram for a circuit- n implementation (b) State diagram for a circuit- t implementation (Exercise 10).TABLE 7.24 Transition table for the sequential circuit- n

Present state		Next state after transition (Q_1', Q_2')		Present output state Y before the transition	
State	(Q_1, Q_2)	Input $X = 0$	Input $X = 1$	$X = 0$	$X = 1$
S_0	0, 0	1, 0	0, 1	0	0
S_1	0, 1	1, 1	1, 0	1	0
S_2	1, 0	0, 0	1, 0	1	1
S_3	1, 1	0, 1	0, 1	1	0

Example 7.10 Table 7.24 gives the corresponding transition table.

TABLE 7.25 Karnaugh Map for Q_1'

$\bar{Q}_1\bar{Q}_2 \backslash X$	\bar{X}	X	
$\bar{Q}_1\bar{Q}_2$	0	1	
$\bar{Q}_1\bar{Q}_2$	00	1	0
\bar{Q}_1Q_2	01	1	1
$Q_1\bar{Q}_2$	11		
Q_1Q_2	10		1

Karnaugh Map for Q_2'

$\bar{Q}_1\bar{Q}_2 \backslash X$	\bar{X}	X	
$\bar{Q}_1\bar{Q}_2$	0	1	
$\bar{Q}_1\bar{Q}_2$	00		1
\bar{Q}_1Q_2	01	1	
$Q_1\bar{Q}_2$	11	1	1
Q_1Q_2	10		

Let us construct Karnaugh map to simplify the circuit specified state diagram of Figure 7.6(a) and the synthesize the circuit and find the expressions.

Solution

Table 7.25 gives the corresponding Karnaugh map.

The Boolean expressions for Q_1' and Q_2' are as follows:

$$Q_1' = \bar{Q}_1 \cdot \bar{X} + \bar{Q}_1 \cdot Q_2 + Q_1 \cdot \bar{Q}_2 \cdot X \quad \dots(7.15)$$

$$Q_2' = Q_2 \cdot \bar{X} + Q_1 \cdot Q_2 + \bar{Q}_1 \cdot \bar{Q}_2 \cdot X \quad \dots(7.16)$$

Example 7.11 Make the excitation table for circuit-n.

Solution

Table 7.26 gives the excitation table when using D-FFs. $Q' = D$. Therefore, simply replace Q_1' and Q_2' in the transition table by D_1 and D_2 to get the excitation table.

TABLE 7.26 Excitation table for the sequential circuit-n

Present state (Q_1, Q_2)	Excitation inputs before the transition (D_1, D_2)		Present output state Y before the transition	
	Input $X = 0$	Input $X = 1$	$X = 0$	$X = 1$
0, 0	1, 0	0, 1	0	0
0, 1	1, 1	1, 0	1	0
1, 0	0, 0	1, 0	1	1
1, 1	0, 1	0, 1	1	0

■ EXERCISES

- Draw a state diagram for Mealy Machine for which a general sequential circuit has excitation table (Table 7.27) as follows:

TABLE 7.27 Excitation table for the for the sequential circuit-4

Present state (Q_1, Q_2)	Excitation inputs (J, K)		Present output state after the X inputs but before the transition	
	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$
0, 0	(1, 0), (1, 1)	(0, 0), (0, 0)	0	1
0, 1	(0, 1), (0, 0)	(1, 1), (1, 0)	1	0
1, 0	(1, 1), (0, 0)	(1, 0), (1, 1)	0	1
1, 1	(0, 0), (1, 0)	(1, 1), (0, 0)	1	1

- Draw a Moore state machine diagram for which a general sequential circuit has the excitation table (Table 7.28) as follows:

7.28 Digital Systems: Principles and Design

TABLE 7.28 Excitation table for the sequential circuit-4

Present state (Q_1, Q_2)	Excitation inputs (J, K), D		Present output Y before the transition
0, 0	Input X = 0 (1, 0), 0	Input X = 1 (0, 0), 0	0
0, 1	(0, 1), 1	(1, 1), 1	1
1, 0	(1, 1), 0	(1, 0), 1	0
1, 1	(0, 0), 1	(1, 1), 0	1

3. Analyze the sequential circuit given by following expressions and draw excitation, transition and state tables.

$$Q'_1 = D_1$$

$$Q'_2 = D_2$$

$$D_1 = (X \cdot \bar{Q}_1 \cdot Q_2 + X \cdot Q_1 \cdot \bar{Q}_2) \text{ and}$$

$$D_2 = X \cdot \bar{Q}_1$$

$$Y = Q_1 \cdot \text{XOR} \cdot Q_2$$

4. Analyze the sequential circuit-0 with following characteristics equations. Draw the state diagram.

$$Q'_1 = D_1$$

$$Q'_2 = D_2$$

$$Y = \bar{Q}_1 \cdot X + \bar{Q}_2 \cdot \bar{X}$$

$$Q'_2 = J \cdot Q_2 + K \cdot \bar{Q}_2$$

$$J = Q_1 \cdot X \text{ and } \bar{K} = \bar{X}$$

5. Analyze the sequential circuit-p with following transition table. Draw the state diagram. If JKFFs are used make the required excitation table (Table 15.29).

6. Draw the state diagram for the following transition table (Table 15.30).

TABLE 7.29 Transition table for the sequential circuit-4

Present state (Q_1, Q_2)	Next state after transition (Q'_1, Q'_2)		Output state Y	
(Q ₁ , Q ₂)	Input X = 0	Input X = 1	Input X = 0	Input X = 1
0, 0	1, 0	1, 1	1	0
0, 1	0, 1	1, 0	1	0
1, 0	1, 1	0, 0	0	1
1, 1	1, 0	0, 1	1	1

TABLE 7.30 State transition table for the sequential circuit-*q*

Present state		Next state after transition (Q_1' , Q_2')		Output state Y	
State	(Q_1 , Q_2)	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$
S_0	0, 0	S_2	S_0	0	1
S_1	0, 1	S_2	S_1	0	0
S_2	1, 0	S_3	S_1	1	1
S_3	1, 1	S_0	S_1	1	0

7. When are the two states not equivalent in a clocked sequential circuit?
8. Perform state minimization for the State table (Table 15.31) for a Moore model sequential circuit-*r* by inspection procedure.

 TABLE 7.31 State table for a Moore model sequential circuit-*r*

Present state		Next state after transition (Q_1' , Q_2' , Q_3')		Present output Y	
State	(Q_1 , Q_2 , Q_3)	Input $X = 0$	Input $X = 1$		
S_1	0, 0, 0	S_4	S_1	1	
S_2	0, 0, 1	S_1	S_2	0	
S_3	0, 1, 0	S_3	S_4	1	
S_4	0, 1, 1	S_1	S_1	1	
S_5	1, 0, 0	S_4	S_2	0	
S_6	1, 0, 1	S_1	S_2	1	
S_7	1, 1, 0	S_3	S_1	1	
S_8	1, 1, 1	S_1	S_2	0	

9. Perform state reduction or minimization of state table in Table 7.32, if feasible. Consider the state table for a Mealy model sequential circuit-*s* in Table 7.32

 TABLE 7.32 State table for a Mealy model sequential circuit-*s*

Present state		Next state after transition (Q_1' , Q_2' , Q_3')		Present output Y	
		Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$
S_1		S_1	S_2	1	0
S_2		S_3	S_5	1	1
S_3		S_1	S_4	0	0
S_4		S_5	S_4	0	1
S_5		S_4	S_3	0	1

10. Make state table, transition and excitation tables and the synthesize a clocked sequential circuit-*t* from the state diagram given in Figure 7.6.(b)

■ QUESTIONS

1. Explain meaning of a state in a sequential circuit.
2. What is the difference between the clocked sequential and asynchronous sequential circuits?
3. Describe procedure for analysis of a given logic circuit based on Moore model.
4. Describe procedure for analysis of a given logic circuit based on Mealy model.
5. How will you make a transition table from a given excitation table?
6. An excitation table is given to you for D FFs at the memory section. How will you convert it for an excitation table using JK FFs?
7. A transition table is given to you. How will you make a state table from it? How will you draw state diagram from it?
8. Explain state minimization procedure.
9. Explain the steps needed for implementing a two-bit counter circuit having four states.
10. Describe rules for the state assignments.

CHAPTER 8

Sequential Circuits for Registers and Counters

OBJECTIVE

We learnt in Chapter 6 the basic memory section elements SR , D , JK and T flip flops. A D flip flop registers at the Q output the bit at D input. This fact can be used to design the multi-bit registers. JK flip-flop, when both J and K equal 1, and T flip flop toggles the Q output on each excitation. This fact can be used to design a divide-by-two element and a multi-bit counter.

We learnt the circuit implementation concepts for the clocked sequential circuits in Chapter 7. Using these, we design a large number of useful circuits, for example, for registers and counters.

We will learn the designs of registers, shift registers, ripple and synchronous counters in this chapter. We will also learn their timing diagrams.

8.1 REGISTERS

Registers can be designed using the D - FF or an $SRFF$ with PR and CLR inputs for presetting all outputs as 1s or 0s, respectively. Note that one type of FF converts by adding some suitable circuit to another type of FF . For example, a JK FF converts to D FF , if a NOT gate is placed before K from J . An SR FF converts to D FF , if a NOT gate is placed before R from S .

Figure 8.1(a) shows a parallel-in parallel-out four-bit register. A register simplest design is made by a combination of D FFs . It is capable for storing logic outputs for a nibble or byte or word (of 4 or 8 or n bits). The number of Q

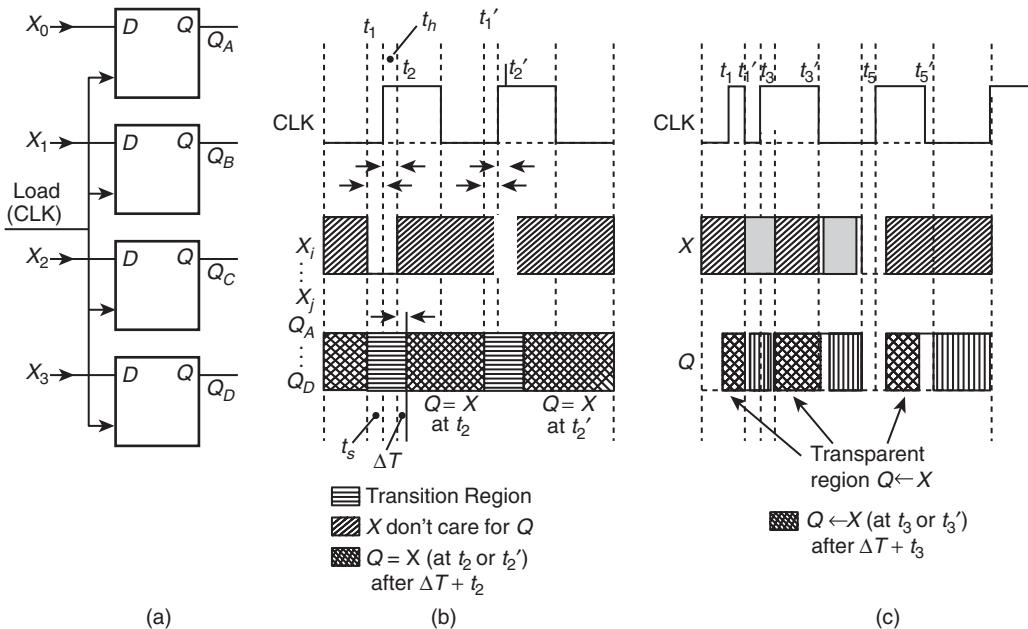


FIGURE 8.1 (a) Parallel—in Parallel—out four-bit register made from four D FFs (b) Timing diagram of a register consisting of the flipflops (c) Timing diagram of a register consisting of the latches.

outputs available as the distinct stable logic states defines the number of bits in a register.

A register is a clocked sequential circuit. The clock inputs of all D FFs are interconnected together in the register. At a clock transition at this common input, all Qs all become identical with respect to the D inputs after ΔT (propagation delay). Let a set of outputs, Q_D, Q_C, Q_B, Q_A define a binary word of four bits. Then the corresponding register shall be called a 4-bit register made from the four D FFs. [Figure 8.1(a)]

Let the CLK of all the four FFs are simultaneously activated by a load input (also called a strobe input or an enable input). The outputs Q_D, Q_C, Q_B, Q_A get the next state after a certain delay equal to propagation delay, ΔT and these simultaneously get stabilized to the input values D_D, D_C, D_B, D_A just before the clock transition from 0 to 1, if the positive edge triggered D FFs are used.

The register shown in Figure 8.1(a) can be extended to the 8-bits or 16-bits or more.

The above register is also called edge triggered load register because the data is only accepted by the register during the very short time between t_1 and t_2 during which clock rises from logic 0 to logic 1 level [refer Figure 8.2(b)].

Figure 8.1(b) shows a timing diagram. D_D, D_C, D_B, D_A must be stable before a time equal to set-up time when the write or load input (clock transition) occurs. D_D, D_C, D_B, D_A must be stable till a time equal to hold time ($= t_h$) when the write input (clock transition) occurs. Data at the inputs needs to be valid only at time equal to t_s (setup time) before a rising edge at the clock input. The data clocks (registers) at the rising edge into the register after a time equal to propagation delay ΔT . [Note: Certain edge triggered registers may write into it the inputs after a falling (trailing) edge i.e. negative edge.]

Q_s correspond to new state after ΔT from the edge. The rising edge of the CLK input thus changes the previous Q outputs in the register and places the new data instead or previous one after ΔT from t_2 .

Point to Remember

A register transfers the input D bits to next Q_s such that $Q'_i = D_i$. A register “looks upon” the data bits at $D_D D_C D_B D_A$ only at the instant of a rising edge. A register does not care (accept or clock) the data before rising edge and after t_2 and will care only again at the next rising edge.

Since the D bits must be set up before a time equal to the setup time, therefore the D s need to be set-up at the inputs before t_s (= set up time) from the clock edge. Next-state Q_s are valid when the D s hold up to t_h (= hold time) after the edge.

8.1.1 Bi-stable Latches as the Register

The bi-stable latches, for example, D latches, arranged in place of the flip flops in circuit of Figure 8.1(a), can also store the data. However, a latch differs from the register in the sense that after the activation of the clock (enable or strobe or write) input (CLK), the bi-stable latches are transparent, and allow the data bits at $D_D D_C D_B D_A$ to pass from the D inputs to the $Q_D Q_C Q_B Q_A$ outputs. However, the $Q_D Q_C Q_B Q_A$ stabilizes to the values after the CLK input inactivates. If we send a pulse at an CLK input, then the $D_D D_C D_B D_A$ should be stable during the entire duration of clock activation plus the duration for set-up for the inputs plus the duration needed t_n = hold up time of the inputs. The last Q_s become the new registered values. These appear after a time equal to ΔT from the inactivation of the CLK.

Figure 8.1(c) gives the timing diagrams for a transparent latche based register. It shows the timings for the parallel bits \underline{X} as inputs and the parallel bits \underline{Q} as outputs. Latches are used when the inputs are expected from some another source and a time interval during which those are expected is the transparent region time interval—the time for which transition enable input (CLK) is active.

Point to Remember

Registers consisting of latches accept the data during the full period of active clock.

8.1.2 Parallel-In Parallel-Out Buffer Register

The circuit of Figure 8.2(a) is also called a parallel-in parallel out (PIPO) register when there are the parallel loading mechanism for the excitation inputs to the memory section (of D FFs) and the parallel outputs mechanism for the next state outputs. Parallel inputs mean $D_i = X_i$ and parallel outputs mean $Y_i = Q_i$ where $i = 0, 1, 2$ or 3 for a 4-bit PIPO.

Point to Remember

A PIPO register transfers the input bits \underline{X} to next Q_s such that $Q'_i = X_i$. A PIPO register loads the external inputs as the excitation inputs and undergoes transition

to the next state on a clock transition. It gives parallel outputs, which are the same as the next state. Parallel inputs mean $D_i = X_i$ and parallel outputs mean $Y_i = Q_i$ where $i = 0, 1, 2$ or 3 for a 4-bit PIPO. Note: $i = 0, 1, 2 \dots n - 1$ for an n -bit register.

8.1.3 Number of Bits in a Register

Figure 8.2(a) shows a PIPO buffer register. Example 8.3 will explain the circuit of the buffer register.

8.2 SHIFT REGISTERS

Figure 8.2(b) shows the circuit of a *serial in serial out* (SISO) shift register. Figure 8.2(c) gives the different forms of the shift registers. A set of cascaded flip-flops to store the information like the register discussed in Section 8.1 along with an additional circuit gives a *shift register*.

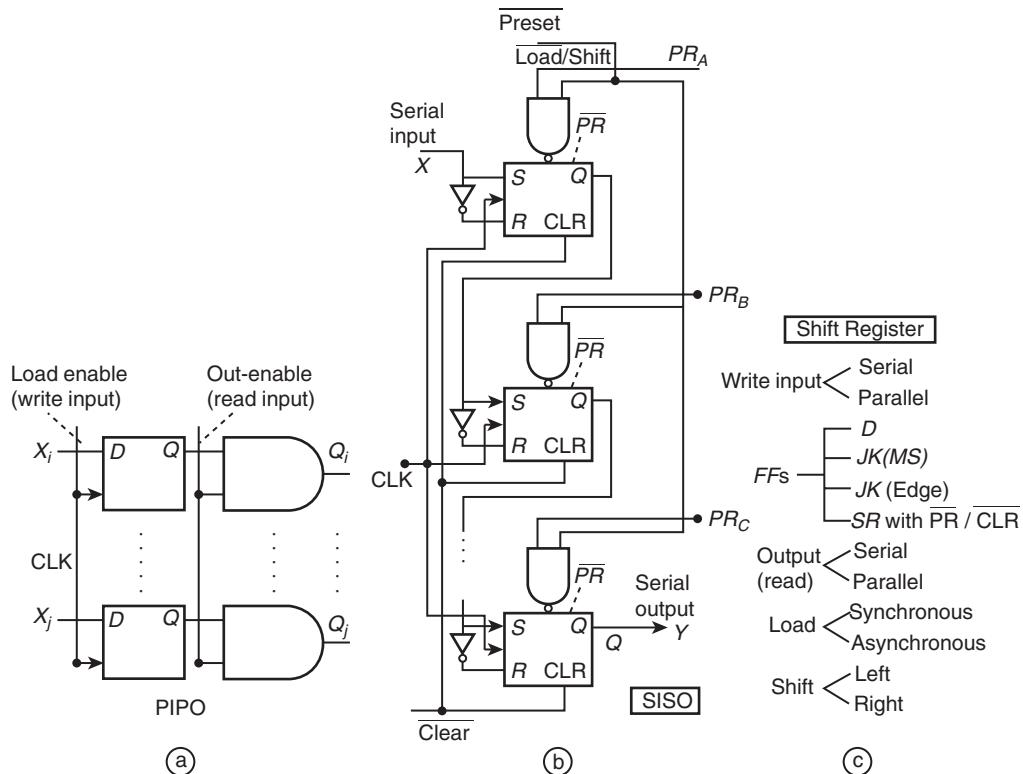


FIGURE 8.2 (a) Parallel—in Parallel—out four bit buffer register made from four D flip flops with out-enable and load-enable input (b) 4-bit serial in serial out (SISO) shift register using the S-R Flip-flops with preset and clear inputs (c) Various types of shift registers.

A shift register is a clocked sequential circuit in which stored each binary word bit shifts either towards left or towards right (towards higher place value or lower place value) on each successive clock transition.

In a parallel input shift register with a serial out, called a PISO, at the every clock pulse, one of the bit at a time of the parallel data bits from the left-most (lsb) or right-most (msb) Q is transferred to a serial output pin.

Shift registers need not be made from discrete logic gates or flip-flops due to complicated circuits of the type in Figure 8.2(b). An MSI IC 7495 (TTL) is a shift register. It is a four-bit parallel cum serial input with the parallel output. It is a left cum right (L/R) shift register. When L/R input = 1, then left-shift, else right shift. An IC 74164 TTL is an 8-bit PISO register. The 74HC95 and 74HC164 are the high-speed CMOS versions of 7595 and 74164.

Let us denote in following description a transfer from a Q to another nearby next stage Q by \rightarrow and to a previous stage Q by \leftarrow . Let us also assume that the Q_A is of greater significance than Q_B ; Q_B of greater than Q_C ; and Q_C is of greater significance than Q_D .

8.2.1 Serial-In Serial-Out (SISO) Unidirectional Shift Register

Table 8.1 gives a state table of a SISO unidirectional shift register with serial input at the flip flop with Q_A output and serial output at the flip flop with QD output. State diagram building from the table is left as an exercise for the reader. (Exercise 2).

Figure 8.2(b) gives a SISO unidirectional shift register, when weight of $Q_A > Q_B$, $Q_B > Q_C$ and $Q_C > Q_D$ and X is a serial input, it is shifting right. Note: In opposite case, it will work as a left shift SISO.

TABLE 8.1 State table for a Moore model sequential circuit for a 4-bit SISO right shift unidirectional register

Present state		Next state after transition (Q'_A, Q'_B, Q'_C, Q'_D)		Present output $Y = Q_D$
State	$Q_A Q_B Q_C Q_D$	Input $X = 0$	Input $X = 1$	
S_0	0000	S_0	S_8	0
S_1	0001	S_0	S_8	1
S_2	0010	S_1	S_9	0
S_3	0011	S_1	S_9	1
S_4	0100	S_2	S_{10}	0
S_5	0101	S_2	S_{10}	1
S_6	0110	S_3	S_{11}	0
S_7	0111	S_3	S_{11}	1
S_8	1000	S_4	S_{12}	0
S_9	1001	S_4	S_{12}	1
S_{10}	1010	S_5	S_{13}	0
S_{11}	1011	S_5	S_{13}	1
S_{12}	1100	S_6	S_{14}	0
S_{13}	1101	S_6	S_{14}	1
S_{14}	1110	S_7	S_{15}	0
S_{15}	1111	S_7	S_{15}	1

The $Q_A Q_B Q_C Q_D$ outputs up on a clock edge transition, within a time equal to ΔT_{SR} from the transition, will load the serial input into the Q_A and serial output is from Q_D .

The register changes as a $X \rightarrow Q_A$ shift register as right $Q_A \rightarrow Q_B$, $Q_B \rightarrow Q_C$, $Q_C \rightarrow Q_D$, $Q_D \rightarrow$ serial out and Q_A will become equal to 0 when serial-in = 0 and equal to 1, when serial in = 1. [Assume Q_D is of higher place (weight) value].

Serial input at right-shift register when state is 0010 (S_2) will become 0001 (S_1) when serial bit $X=0$ and serial-out Y will be = 0. Serial input when state is 0001 (S_1) will become 1000 (S_8) when serial bit = 1 and serial-out Y will also be = 1 because past $Q_A=1$.

Point to Remember

A right shift SISO loads the inputs bit X into the last highest place value stage D_A and gives the serial output = Q_D on a clock edge. Its state $Q'_j = D_i$ on the clock edge. Here and $i = 0, 1, 2, \dots, n-1$ in an n -bit register and j a nonnegative value differs from i by 1. The left shift register on the other hand, will load $Y \leftarrow Q_A$ on $Q_D \leftarrow X$.

8.2.2 Serial-In Parallel-Out (SIPO) Right Shift Register

Table 8.2 gives a state table of a right shift SIPO. Figure 8.3(a) shows a right shift SIPO register, when weight of $Q_0 > Q_1, Q_1 > Q_2$ and $Q_2 > Q_3$ and X is a serial input. Note: In opposite case, it will work as a left shift SIPO. Figure 8.3(b) shows the state diagram of the right shift SIPO register.

TABLE 8.2 State table for a Moore model sequential circuit for a 4-bit SIPO shift right register

Present state	Next state after transition (Q_0', Q_1', Q_2', Q_3')		Present outputs Y_0, Y_1, Y_2, Y_3
	Input $X = 0$	Input $X = 1$	
State	$Q_0 Q_1 Q_2 Q_3$		
S_0	0000	S_0	0000
S_1	0001	S_0	0001
S_2	0010	S_1	0010
S_3	0011	S_1	0011
S_4	0100	S_2	0100
S_5	0101	S_2	0101
S_6	0110	S_3	0110
S_7	0111	S_3	0111
S_8	1000	S_4	1000
S_9	1001	S_4	1001
S_{10}	1010	S_5	1010
S_{11}	1011	S_5	1011
S_{12}	1100	S_6	1100
S_{13}	1101	S_6	1101
S_{14}	1110	S_7	1110
S_{15}	1111	S_7	1111

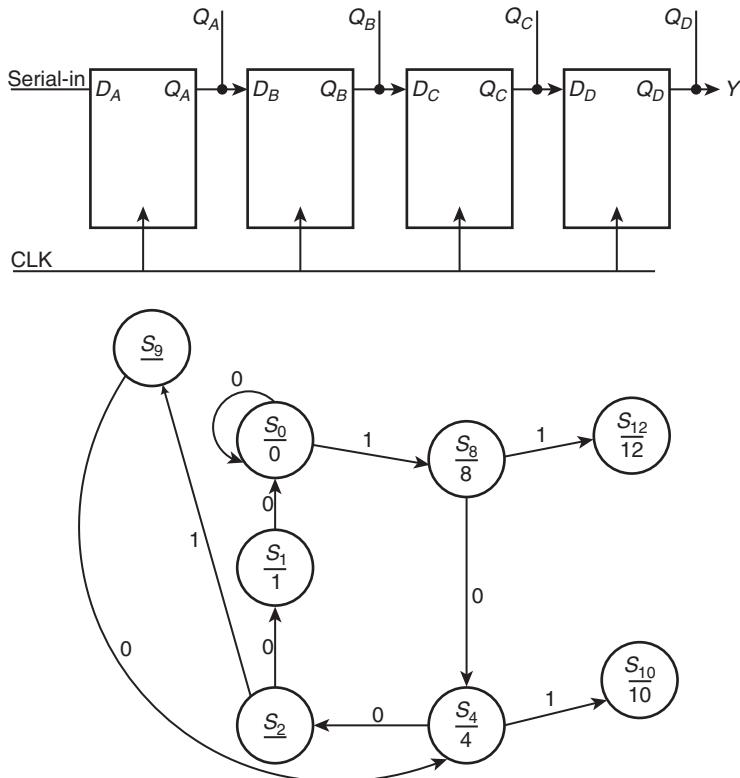


FIGURE 8.3 (a) Right shift SIPO register, when weight of $Q_0 > Q_1, Q_1 > Q_2$ and $Q_2 > Q_3$ and X is a serial input and Y is serial output (b) State diagram of the SIPO right shift register.

The Q_0, Q_1, Q_2, Q_3 outputs up on a clock edge transition, within a time equal to ΔT_{SR} from the transition, will change in a right shift register as $Q_0 \rightarrow Q_1, Q_1 \rightarrow Q_2, Q_2 \rightarrow Q_3$, and Q_0 will become equal to a serial input bit. [Assume Q_0 is of higher place (weight) value]. Figure 8.3(a) shows a 4-bit right shift register in which a serial bit propagate from Q_0 to Q_3 (higher to lower place value) and also appear as output in case of a SIPO. The shifting is done by a clock transition at the CLK input in Figure 8.3(a). Present state 0010 (S_2) will become 0001 (S_1) when serial bit = 0, and 1001, (S_9) when serial bit = 1. [Present $Y = 0010$.]

Point to Remember

A SIPO shift-right register transfers the input bits X to next Q s at each clock edge excitation so that $Q'_j = X_i$, where $j = i - 1; j \geq 0$ and $i = 0, 1, 2, \dots, n - 1$ in an n -bit register. The msb will become equal to serial bit. It gives parallel outputs, which are the same as the next state, which becomes present state for the next excitation. When serial shift input bit $X = 0$, right shift register divides a binary number by 2. [Then left shift, multiplies by 2.]

8.2.3 Parallel-In Serial-Out (PISO) Right Shift Register

Table 8.3 gives a state table of a right shift PISO; assuming Q_0 is msb (maximum significance bit) and Q_3 as lsb. [Assume that weight of $Q_0 > Q_1, Q_1 > Q_2, Q_2 > Q_3$.]

TABLE 8.3 State table for a Moore model sequential circuit for a 4-bit PISO right shift register

Present state	Next state after transition (Q'_0, Q'_1, Q'_2, Q'_3)					Present outputs Y
	\bar{L}/S	\bar{L}/S	\bar{L}/S	...	\bar{L}/S	
Serial-in, $X_0, X_1,$ X_2, X_3 = (000000)	Serial-in, $X_0, X_1,$ X_2, X_3 = (100000)	Serial-in, $X_0, X_1,$ X_2, X_3 = (110000)	...	Serial-in, $X_0, X_1,$ X_2, X_3 = (101111)	Serial-in, $X_0, X_1,$ X_2, X_3 = (111111)	
$S_i = S_0$ or S_1 or S_2 or ... S_{15}	S_0	S_0	S_8	...	S_7	S_{15}
						$Y = Q_3$

Note: There are six inputs are Load/Shift (L/S), Serial-in X_0, X_1, X_2 and X_3 . Q_3 is the lsb and Q_0 is msb.

Figure 8.4(a) gives a right shift PISO register, when weight of $Q_0 > Q_1, Q_1 > Q_2$ and $Q_2 > Q_3$ and X is a serial input. In opposite case, it will work as a left shift PISO. Figure 8.4(b) shows the state diagram.

The Q_0, Q_1, Q_2, Q_3 are outputs after a clock edge transition. Within a time equal to ΔT_{SR} from the transition, the PISO will load the inputs into the Q s when Load / Shift = 0. When the Load / Shift = 1, the register changes as a right shift register as $D_0 \rightarrow Q_1$, $D_1 \rightarrow Q_2$, $D_2 \rightarrow Q_3$, and Q_0 will become equal to 0 when serial-in = 0 and equal to 1, when serial in = 1. [Assume Q_0 is msb of (higher place (weight) value).]

Figure 8.4(a) shows a 4-bit parallel loading cum right-shift serial-out register in which a serial bit propagate from Q_0 to Q_3 (higher to lower place value).

Parallel input 0010 (X_2) will become 0001 (S_1) when serial bit = 0 and serial-out Y will be = 0. Parallel input 0001 (X_1) will become 1000 (S_8) when serial-in bit = 1 and serial-out Y will be = 1.

Point to Remember

A right shift PISO loads the parallel inputs bits X into the D s, and the $Q_i = D_i$ on the clock edge, when Load / Shift input = 0. Here which $i = 0, 1, 2, \dots, n - 1$ in an n -bit register.

When Load / Shift input = 1, the PISO transfers the D input bits X to next Q s at each clock edge excitation so that $Q'_j = X_j$, where $j = i - 1; j \geq 0$. The msb Q_0 will become equal to serial bit X and lsb Q_3 will be the one-bit output Y at the serial-out pin. The parallel Q outputs are the same as the next state. When serial shift input $X = 0$, right shift register divides the input binary number by 2. [The left shift multiplies by 2.]

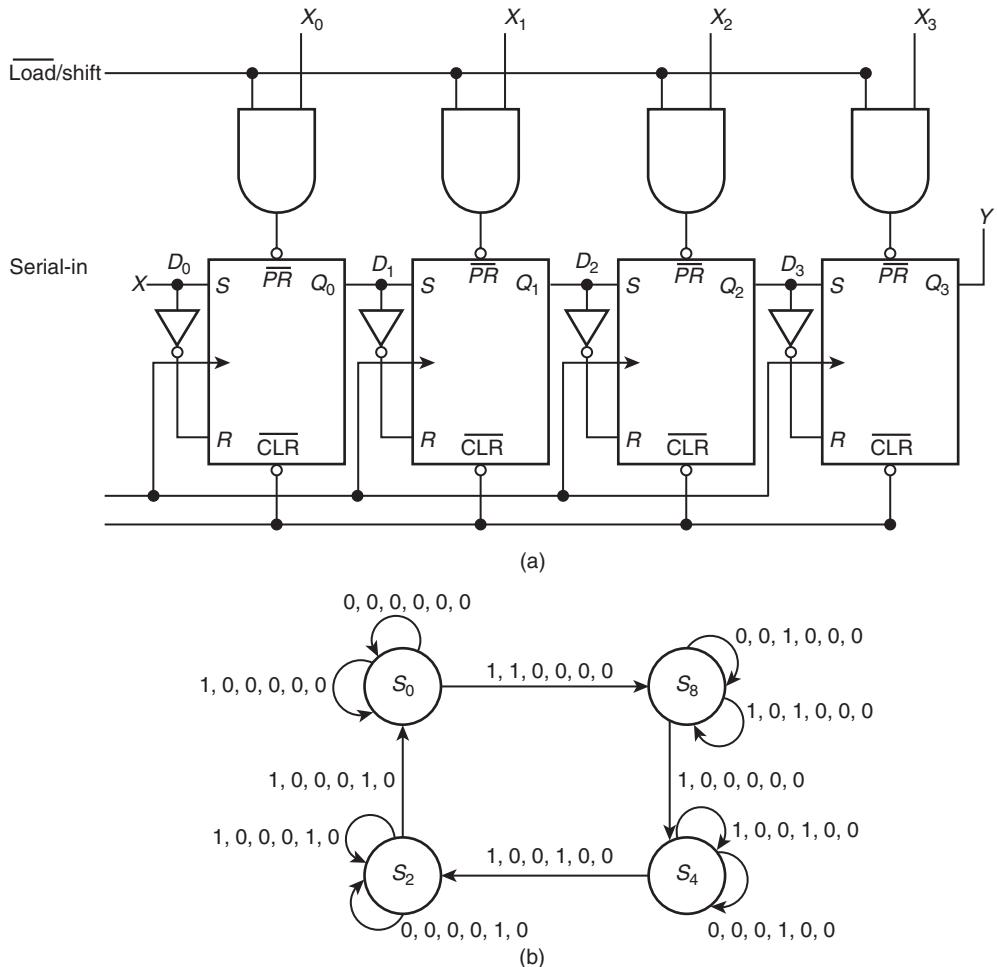


FIGURE 8.4 (a) Right shift PISO register, when weight of $Q_0 > Q_1, Q_1 > Q_2$ and $Q_2 > Q_3$ and Y is a serial output and X_0, X_1, X_2 and X_3 are the serial inputs (b) State diagram of PISO right shift register.

8.3 COUNTER

Often there is a need to count the number of pulses or triggering at an input. Counting is an essential circuit in computers. Figure 8.5(a) gives various types of counters and the various features defining a counter. Figure 8.5(b) shows a state diagram of the 16-states in the counter. It changes a state along 16 directed arcs in a cycle.

A circuit with state diagram like in Figure 8.5(b) is a counter. Figure 8.5(c) shows a binary counter. It is the counter when there is additional constraint that a state has the Q s combination corresponding to a binary number, whose output state give a binary number and cyclic changes only can occur either clock wise or anti clock wise to increase or decrease the count number for a up counter or down counter, respectively.

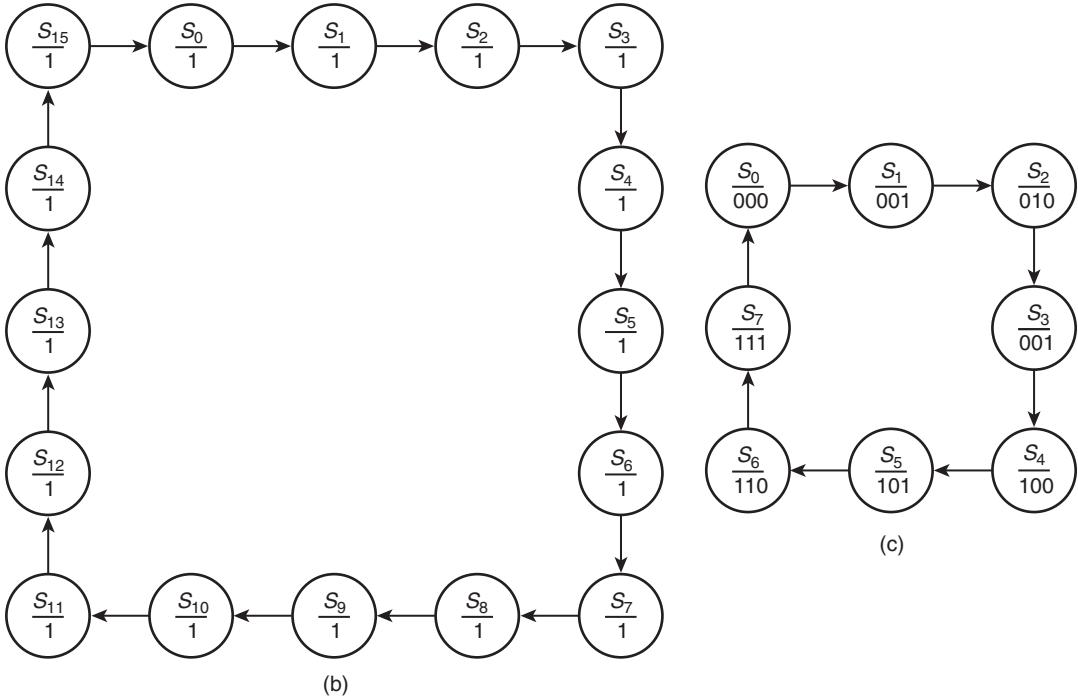
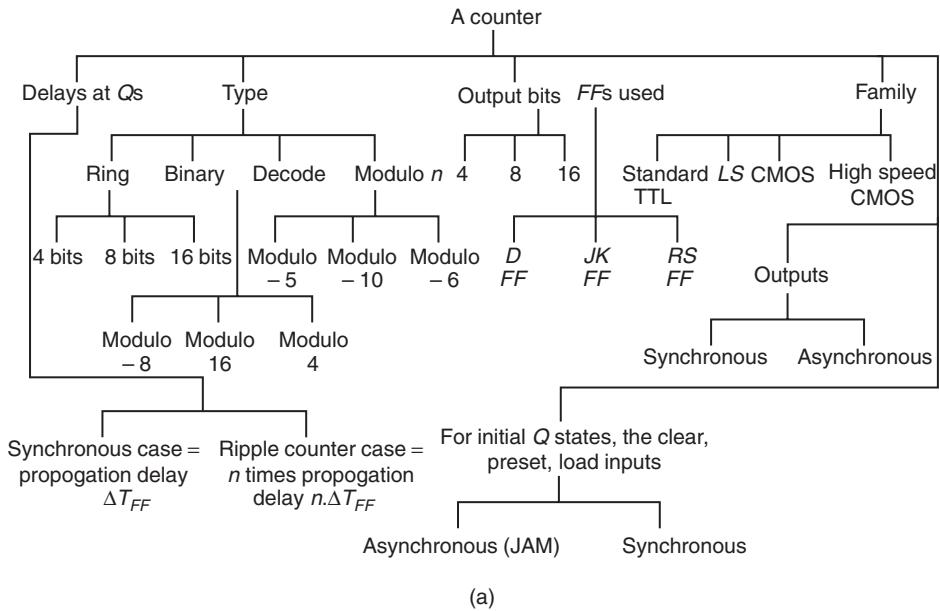


FIGURE 8.5 (a) Various types of counters and the various features defining a counter (b) State diagram of a negative pulse triggered 16-state counter. It changes state along 16 directed arcs in a cycle (c) State diagram of a count input C triggered 8-state binary 3-bit counter. It changes state along the 8 directed arcs in a cycle.

Divide-by-Two Flip-Flop as a Basic Counting Element

A divide-by-2 circuit produces one output pulse for every two pulses applied to its input. A divide-by-two circuit is made from a TFF. We learnt in Chapter 6 that one type of *FF* converts by adding some suitable circuit to another type of *FF*. For example, a *D* flip flop converts to *T FF* if *D* input = (*T* XOR Q_n). Q_{n+1} will toggle and will be Q_n when *T* = 1 before a clock edge. A *JK* *FF* converts to *T FF* if both *J* and *K* are made = 1. Q_{n+1} will toggle and will be Q_n .

Following are the ways to design a divide-by-2 circuit:

1. Use a circuit of *T*-type flip flop
2. Use a single *JK* flip flop with its *J* and *K* inputs made 1. [Refer method *i* in the Figure 8.6(a). The *T* flip flop (*FF*) is designed from *JK* or any other method to act as a divide-by-two circuit and *JK* input is now the *T*-input.]
3. Use a *D* flip flop (not *D* latch) with its \bar{Q} output feedback to the *D* input [refer method *ii* in Figure 8.6(a)]. Alternatively use a *S R* flip flop with a NOT in-between *S* and *R* to get a DFF and then convert a DFF into *T*.

T stands for the toggling. We can say the output *Q* toggles between 0 and 1. Let a *T*-flip flop is positive edge triggered. Consider first two 0 to 1 transitions. At the first clock input, *T* transition, the output is complement of the previous *Q* value, and then at next transition (0 to 1) at the clock input, the output *Q* reverts to its previous value.

Let us now consider a *T*-flip flop, which is –ve edge triggered. At the *T* input, a 1 to 0 transition is applied twice to obtain the previous value of *Q*. In other words, the output returns to original logic state for every two successive clock input at *T*.

Figure 8.6(b) shows the next state part of the transition table for a divide-by-two flip flop circuit. Two pulses at the *T* input cause the circuit to shift from one stable state to another and then back to first again.

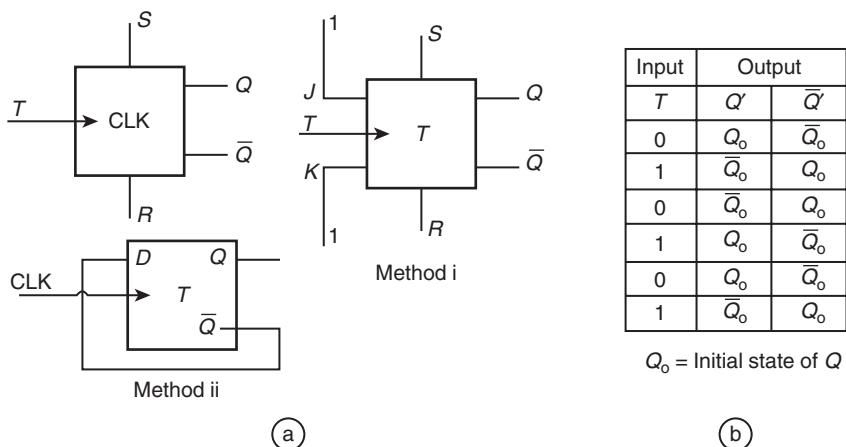
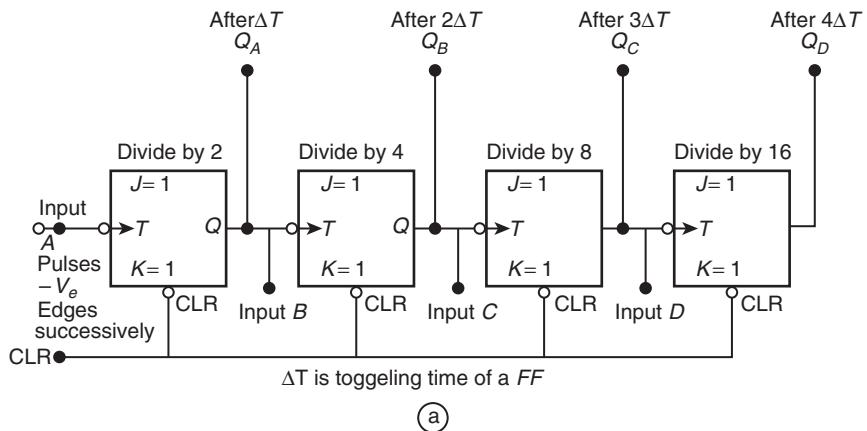


FIGURE 8.6 (a) Divide-by-two circuit basic unit using *T*-, *JK*- or *D* flip flops (b) A partial transition table for divide-by-two circuit.

8.4 RIPPLE COUNTER

8.4.1 Cascaded Divide-By- 2^n Circuit as a Ripple Counter

We have seen above that a T FF acts as a divide-by-2 circuit. If Q output of the FF connects to the T input of the second FF , and the Q output from the second FF connects to T -input of the third flip flop and so on, the FF s are said to be in a cascade (Figure 8.7(a)). The output transition partial table of this cascade asynchronous circuit for a 4-bit –ve going input pulse triggered ripple counter is given in Figure 8.7(b).



		First make $CLR = 0$ get $Q_A = Q_B = Q_C = Q_D = 0$			
		Outputs after different delays			
		Q_D	Q_C	Q_B	Q_A
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

(b)

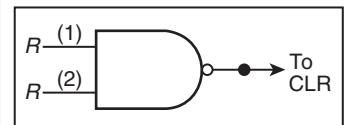


FIGURE 8.7 (a) Cascaded divide-by-two circuits using T –ve edge triggering pulse to get a 4-bit binary asynchronous (ripple) counter (b) Partial transition table (c) Additional feature in ripple counter in IC 7493 to enable its conversion to modulo-6 counter.

A cascade of n flip flops gives a divided by 2^n circuit. It is also called a ripple counter because a state is transferred from one stage to next stage after some delay like a ripple in water.

A cascade of four flip flops A, B, C, D is there in Figure 8.7(a) circuit. The outputs are Q_A, Q_B, Q_C and Q_D after each negative edged successive transition at T . A 0 to 1 followed by an 1 to 0 input at a T input forms one negative edged trigger. By convention, a sequence of writing in a transition table is Q_D, Q_C, Q_B and Q_A . This is so because in decimal system, we write a bigger placed value digit on the left. In one hundred, the place value of 1 is biggest. So 1 is written as left most when we write 100 in decimal system. Similarly, in the FFs shown in Figure 8.7(a), the Q_D changes after 8 pulses and return to original state after 16 pulses. Therefore, Q_D is written on the left-most side by convention. The output Q_A is obtained after a delay, called propagation delay, ΔT , of a T FF. Q_B is obtained after delay of $2\Delta T$, Q_C is obtained after a delay of $3\Delta T$ and Q_D is obtained after a delay of $4\Delta T$.

A combination of the flip flops A and B gives a divide by four circuit, combination of B, C and D with input given at input B with Q_A disconnected gives a divide by eight circuit, and combination of A, B, C and D FFs gives a divide by 16 circuit when Q_A not disconnected to input B . Q_B to C and Q_C to D .

The cascade FFs act as registers of the number of pulses, act as a ripple counter, and also can manipulate digital logical inputs, which represent the binary number.

Table 8.4 gives state table for a 4-bit –ve going input pulse triggered ripple counter. Figure 8.5(b) showed the state diagram of the counter. Note that it is table for asynchronous circuit. State is also as one of the input.

TABLE 8.4 State Table for a 4-bit –ve going input pulse triggered ripple (asynchronous) counter

Present state	Next state after transition (Q_3', Q_2', Q_1', Q_0')		Present output $Y = Q_3 Q_2 Q_1 Q_0$
	For an Input $Q_1 Q_0$	For an Input State with $T = 1$	
S_0	0000	S_0	0000
S_1	0001	S_1	0001
S_2	0010	S_2	0010
S_3	0011	S_3	0011
S_4	0100	S_4	0100
S_5	0101	S_5	0101
S_6	0110	S_6	0110
S_7	0111	S_7	0111
S_8	1000	S_8	1000
S_9	1001	S_9	1001
S_{10}	1010	S_{10}	1010
S_{11}	1011	S_{11}	1011
S_{12}	1100	S_{12}	1100
S_{13}	1101	S_{13}	1101
S_{14}	1110	S_{14}	1110
S_{15}	1111	S_{15}	1111

The counting circuit of Figure 8.7(a) is also called asynchronous up counter. Asynchronous behavior is because of its design as a ripple counter. (Up counter is because its next state binary number increases from all 0s to all 1s). Q s change from 0000 to Q s 1111. Figure 8.7(b) gives its transition partial state table.

Figure 8.7(c) shows the two additional features in a standard TTL IC 7493 or CMOS IC 7493, which has configuration of the type of Figure 8.7(a). This enables modulo 6 or 7 or 10 designs from 16 state counter.

8.4.2 Modulo-6, Modulo-7 and Modulo-10 Counters

If a counter returns to original state after $Q_B = 0$, $Q_A = 1$ and $Q_C = 1$, we say it is modulo-6 counter. Such counter is useful in watches/clocks due to fact that 60s = 1 minute and 60 minute = 1 hour. [Figure 8.8(a)]

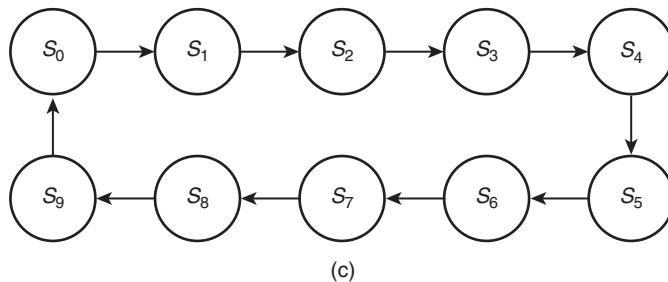
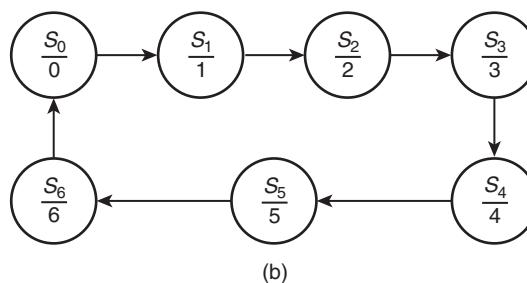
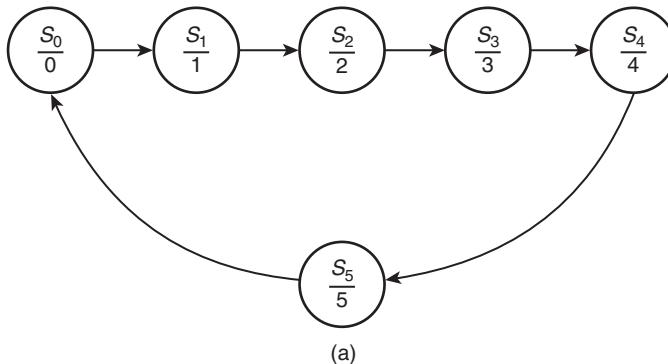


FIGURE 8.8 (a) State diagram of modulo-6 binary counter (b) State diagram of modulo-7 binary counter (c) State diagram of modulo-10 binary counter from a 4-bit binary counter.

If a counter returns to original state after $Q_B = 1$, $Q_A = 0$ and $Q_C = 1$ and has state diagram as shown in Figure 8.8(b), we say it is modulo-7 counter. Such counter is useful in triggering actions on odd number of sequences (action after seven in place of six).

If a counter returns to original state after $Q_D = 1$, $Q_B = 0$, $Q_A = 1$ and $Q_C = 0$, we say it is modulo-10 counter or decade counter. [Figure 8.8(c)]

8.4.3 Ring Counter

A ring counter is shown in Figure 8.9(a). It is a shift register based counter. It is a shift register with serial-input bit permanently from Q_D (msb). It has clear and preset facilities and with no serial output facility.

Its partial transition state stable is given in Figure 8.9(b). The ring counter shifts state 1 from lower significant bit (lsb) end to maximum significant bit (msb) end. The msb is feedback to $D\text{ FF-A}$ to form a ring. It acts like a cyclic rotate right of bit 1 toward left (msb place).

When the outputs of a ring counter connect a buffer of n-inputs, the n-sequences activate cyclically. Figure 8.10(b) shows how a 4-bit ring counter output to a buffer that activates four sequences cyclically.

The output of the counter is not binary. From the table, output sequences are 0001, 0010, 0100, 1000, 0001, 0010, ... [Refer Figure 8.10(c) state diagram]

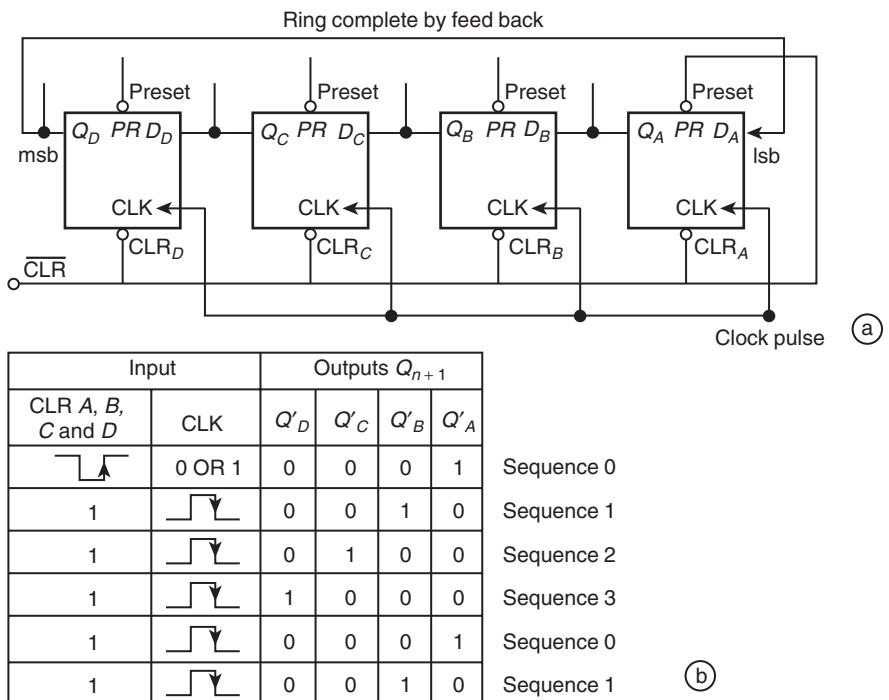


FIGURE 8.9 (a) Circuit of a ring counter (b) Partial transition table of ring counter.

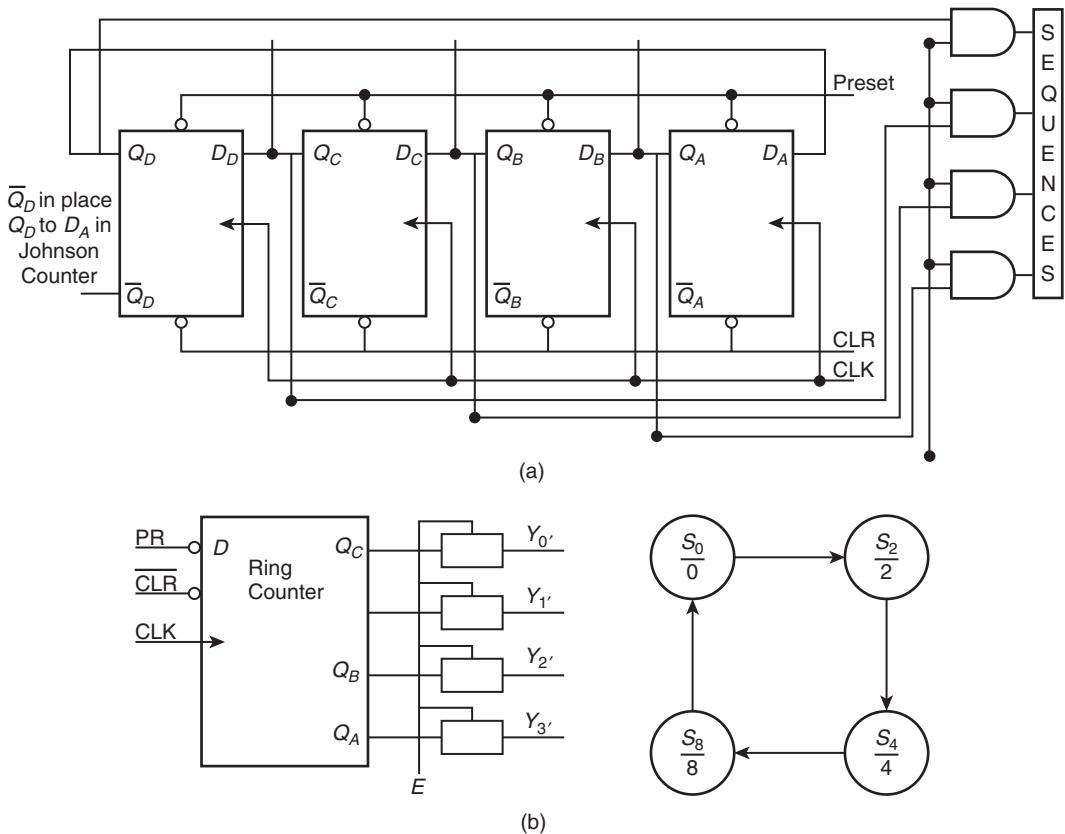


FIGURE 8.10 (a) Activation of four sequences cyclically using a four bit ring counter and method to change it into Johnson counter for activation of eight sequences cyclically using four bits (b) Ring counter block diagram and four sequences.

Such a counter is useful in a computer where several circuits are sequentially enabled (or activated) from 1st stage to the n th stage using an n -bit ring counter.

8.4.4 Johnson Counter (Even Sequences Switch Tail or Twisted Ring Counter)

A Johnson counter connection changes are shown in Figure 8.10(a). It is a shift register with serial-input bit permanently from last msb state \bar{Q} . It has clear and preset facilities and with no serial output facility. Figure 8.11(a) shows the changes in ring counter of Figure 8.10(a) to activate 8 sequences.

Table 8.5 gives the state table. The Johnson counter shifts state 1 from lower significant bit (lsb) end to maximum significant bit (msb) end. The complement of msb \bar{Q}_D is feedback to D_A input of the first stage D FF to form a ring. It acts like a shift left of the bit and also rotate the complement of msb.

The output of the counter is not binary. From the table, output sequences are 0001, 0011, 0111, 1111, 1110, 1100, 1000, 0000, 0001, 0011 ...

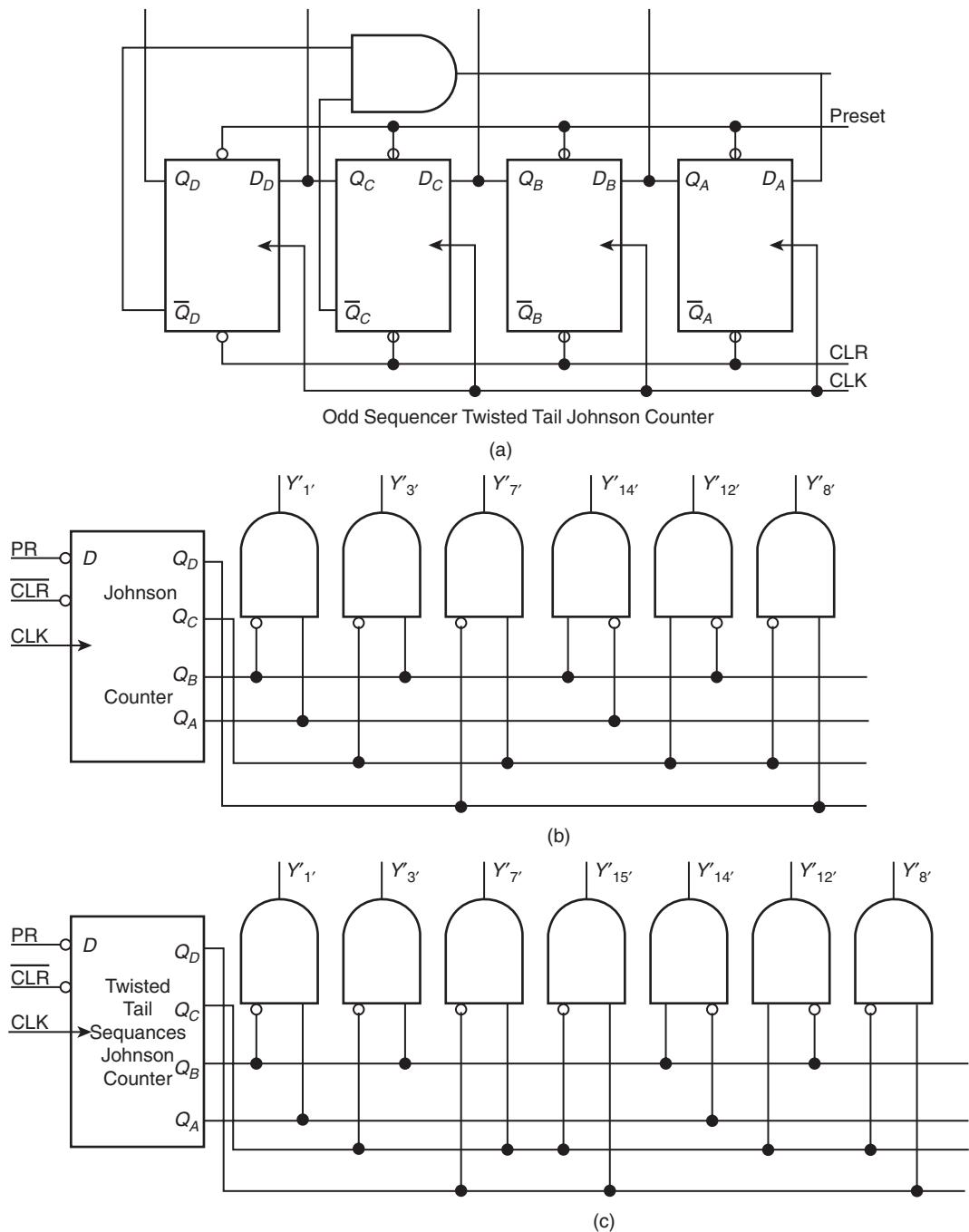


FIGURE 8.11 (a) Activation of eight sequences (state table of Table 8.5 in a twisted ring counter) (b) Cyclically 6 sequence activator using 6-AND array and a four bit Johnson counter (c) A four bit twisted tail odd sequencer Johnson counter—activator of seven sequences cyclically.

TABLE 8.5 State table for a 4-bit Johnson counter to activate eight sequences ($(Y'_0, Y'_1, Y'_3, Y'_7, Y'_{15}, Y'_{14}, Y'_{12}, Y'_8)$)

Present state		Next state after transition (Q_3', Q_2', Q_1', Q_0')	Present output Y
State	$Q_3 Q_2 Q_2 Q_1$		
S_0	0000	<u>S_1</u>	<u>0001</u> (Y'_1)
S_1	0001	<u>S_3</u>	<u>0011</u> (Y'_3)
S_3	0011	<u>S_7</u>	<u>0111</u> (Y'_7)
S_7	0111	<u>S_{15}</u>	<u>1111</u> (Y'_{15})
S_{15}	1111	<u>S_{14}</u>	<u>1110</u> (Y'_{14})
S_8	1000	<u>S_{12}</u>	<u>1100</u> (Y'_{12})
S_{12}	1100	<u>S_8</u>	<u>1000</u> (Y'_8)
S_{14}	1110	<u>S_0</u>	<u>0000</u> (Y'_0)

Note: Q outputs which define a unique sequence are connected to AND gate of the AND array are underlined in the last column to activate outputs in circuit of Figure 8.11(b).

When pair of Q outputs of a Johnson counter connects to a 2– input enable buffer (Figure 8.11(b)), the $2n$ -sequences activate cyclically. Figure 8.11(b) shows how a 4-bit Johnson counter pair of FF outputs to the buffers activates 6 sequences $Y'_1, Y'_3, Y'_7, Y'_8, Y'_{14}, Y'_{12}$ cyclically. [Maximum eight outputs $Y'_1, Y'_3, Y'_7, Y'_{15}, Y'_{14}, Y'_{12}, Y'_8$ and Y'_0 can be activated].

Such a counter is useful in a computer where $2n$ circuits are sequentially enabled (or activated) from 1st stage to the $2n^{\text{th}}$ stage using an n -bit Johnson counter.

8.4.5 Odd Sequencer Johnson Counter (Odd Sequencer Switch Tail or Twisted Ring Counter)

An odd sequencer (seven numbers) Johnson counter (switch tail or twisted ring counter) is shown in Figure 8.11(c). It is a shift register with serial-input bit permanently to an AND gate output, which has inputs \bar{Q}_D and \bar{Q}_C . It has clear and preset facilities and with no serial output facility.

Table 8.6 gives the state table. The twisted tail odd sequencer counter shifts state 1 from lower significant bit (lsb) end to maximum significant bit (msb) end. The msb Q_D complement and its previous stage msb complement are feedback through a two-input AND gate to D input of the first stage FF_A to form a ring [Figure 8.11(a)]. It acts like a shift left of bit and rotates two stages complement after ANDing.

The output of the counter is not binary. From the table, output sequences are 0001, 0011, 0111, 1111, 1110, 1100, 1000, 0001, 0011...

When pair of Q outputs of a Johnson counter connects to a 2– input enable buffer, the $(2n - 1)$ sequences activate cyclically. Figure 8.11(c) shows how a 4-bit twisted tail modulo-7 counter connect 7 pairs of FF outputs to the seven number two-input AND arrays and activates 7 clock sequences $Y'_1, Y'_3, Y'_7, Y'_{15}, Y'_{14}, Y'_{12}$ and Y'_8 cyclically.

Such a counter is useful in a computer where 6 or 7 circuits are sequentially enabled (or activated) using an 4-bit Johnson counter.

TABLE 8.6 State table for a 4-bit Johnson counter to activate seven sequences from circuit of Figure 8.11(c)

Present state		Next state after transition (Q_A' , Q_B' , Q_C' , Q_D')	Present output Y
State	$Q_3 Q_2 Q_1 Q_0$		
S_1	0001	S_1	0 0 <u>0</u> 1 (Y'_1)
S_3	0011	S_3	0 0 <u>1</u> 1 (Y'_3)
S_7	0111	S_7	<u>0</u> 1 1 1 (Y'_7)
S_8	1000	S_{15}	<u>1</u> 1 1 1 (Y'_{15})
S_{12}	1100	S_{14}	1 1 <u>1</u> 0 (Y'_{14})
S_{14}	1110	S_{12}	1 <u>1</u> 0 0 (Y'_{12})
S_{15}	1111	S_8	<u>1</u> 0 0 0 (Y'_0)

Note: Q outputs which define a unique sequence are connected to AND gate of the AND array are underlined in the last column which activate outputs in circuit of Figure 8.11(c).

8.5 SYNCHRONOUS COUNTER

8.5.1 Synchronous Counter Using Additional Logic Circuit

Let us consider a typical design shown in Figure 8.12. The J and K inputs connect together in each flip flop, and are connected to a logic combinational circuit. J and K inputs are held 0 so that Q s don't change up to the final stage gets the counting

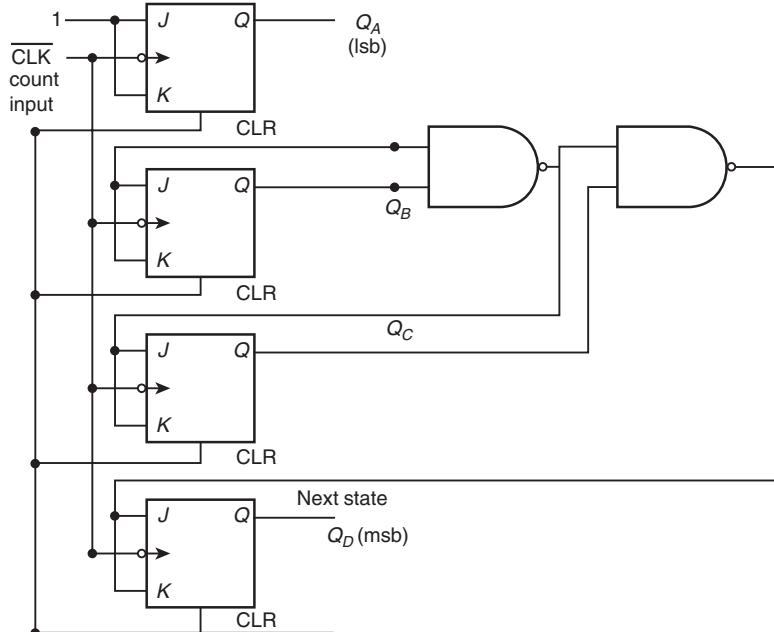


FIGURE 8.12 A synchronous counter using J - K FFs and circuit to force state change at all Q s.

input. As soon as final stage gets the input, the J and K of all FFs will simultaneously equal to 1 and toggle as per the inputs. Except first Q_A , all Q outputs are also inputs to this combinational logic circuit. This logic circuit checks the states of Q_A , Q_B , Q_C and Q_D and gives the different JK FFs a pair of inputs 0 or 1. When both JK inputs to a FF are 0 the output is unaffected but if both JK inputs to a FF are 1, the output is affected, and its output Q_n is complemented.

The combination logic circuit ensures that all the FFs change the Q outputs precisely at the same time after the clock edge transition.

Whenever the clock input makes a desired transition, that all T inputs are interconnected together. Such synchronous counters are useful for fast counting applications as these provide output valid states fast, and for applications when all Q_D , Q_C , Q_B , Q_A are to be read simultaneously. False intermediate states are also now not obtained in the output.

8.6 ASYNCHRONOUS CLEAR, PRESET AND LOAD (JAM) IN A COUNTER

1. There may be a need to reset all Q s as 0s at the start of the counter. For example, in an up counter.
2. There may be a need to reset all Q s as 1s at the start of the counter. For example, in a down counter.
3. There may be a need to set certain Q s as 1s and remaining as 0s at the start of the counter. For example, in a timer. Timer is a counter getting counting inputs at regular intervals after presetting a certain value.

How do we do these three operations in a counter? Following description explains these operations.

When we activate CLR input at a counter, without waiting for a clock edge all the output Q s may become 0 after the propagation delay ΔT . This procedure-adopting counter is known as the asynchronous clearing counter. If all output Q s can be preset to 1s without waiting for a clock edge (but of course after a time ΔT). Such a procedure-adopting counter is known as an asynchronous presetting counter.

If without waiting for a clock edge, we can clear certain Q s and preset remaining Q s in a counter simultaneously, then individual FF clearing and presetting inputs are called JAM inputs. JAM inputs asynchronously load without waiting for a clock edge at the input the Q s of the counter.

Figure 8.13(a) shows resetting = 0s of all Q s by a common CLR input. Figure 8.13(b) shows setting of all Q s = 1s by a common PR input. Figure 8.13(c) shows setting and resetting of all Q s as desired by the JAM inputs.

8.7 SYNCHRONOUS CLEAR, PRESET AND LOAD FACILITIES IN A COUNTER

Figure 8.13(d) shows the circuit for synchronous clear, preset and load.

TTL 74163 or CMOS 74HC163 has synchronous clear and synchronous load facility. In that case, the clear, preset or load inputs must be properly defined

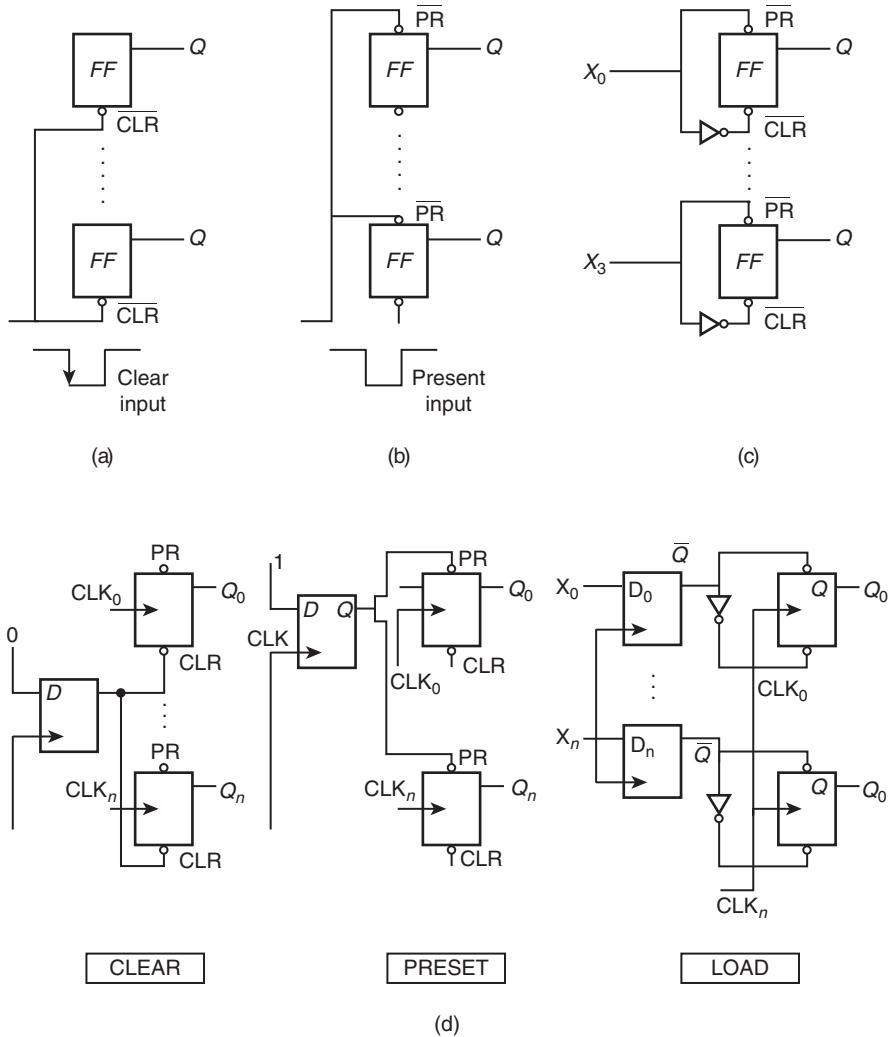


FIGURE 8.13 (a) Clearing all Q s (b) Presetting all Q s (c) Setting and clearing of Q s using the JAM inputs (d) Synchronous clear, preset and load.

at a time, t_s , called setup time, before a clock edge at the input is activated. The effects of the CLR (clear) or PR (preset) or load inputs appear in the outputs only after a time equal to ΔT_{FF} from the clock input transition i.e. activation (net time taken $> t_s + \Delta T_{FF}$). Here, the ΔT_{FF} is propagation delay within a FF of the counter.

8.8 TIMING DIAGRAMS

Figures 8.14, 8.15, 8.16 and 8.17 show the timing diagrams in a SISO, in a left-shift SIPO register, in a ripple counter and in a synchronous counter, respectively.

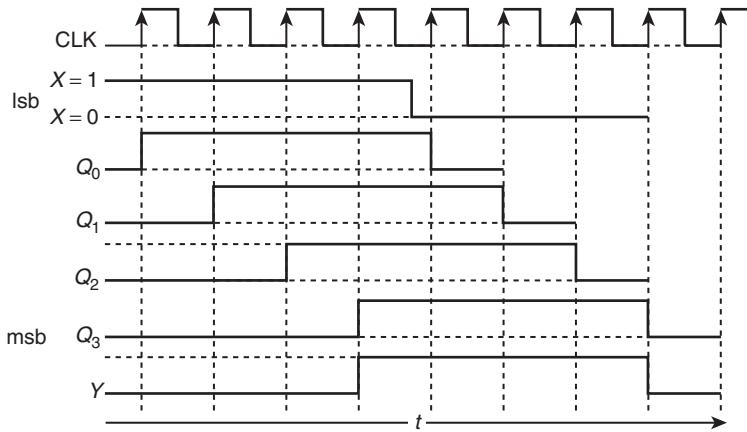
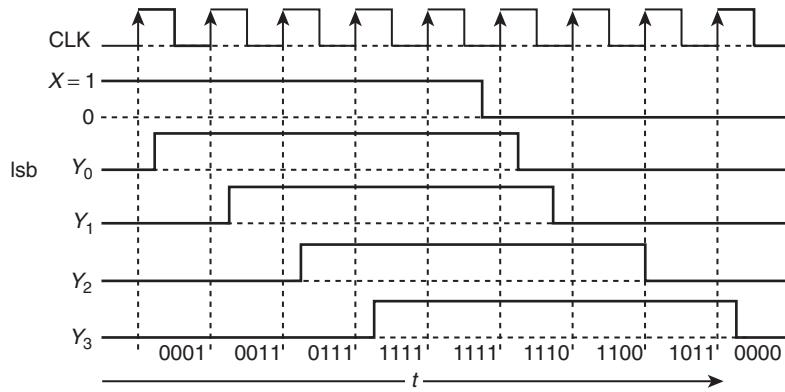
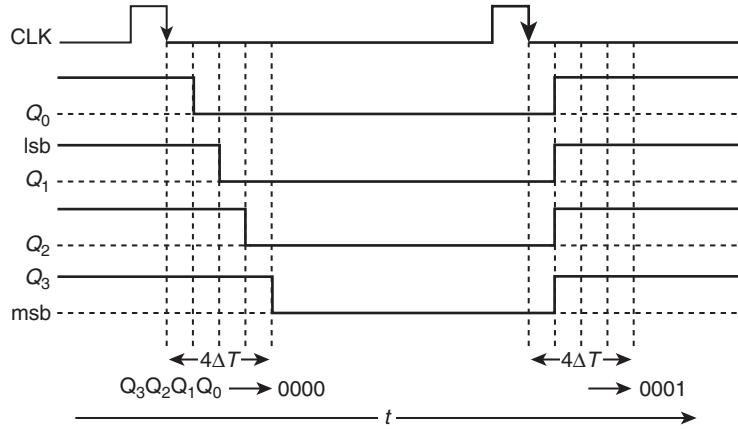


FIGURE 8.14 Timing diagrams in a SISO.

FIGURE 8.15 Timing diagrams in a SIPO left-shift shift register. (ΔT_{FF} delay effects also shown)FIGURE 8.16 Timing diagrams in a $-ve$ edge triggered ripple counter. (ΔT_{FF} effects also shown)

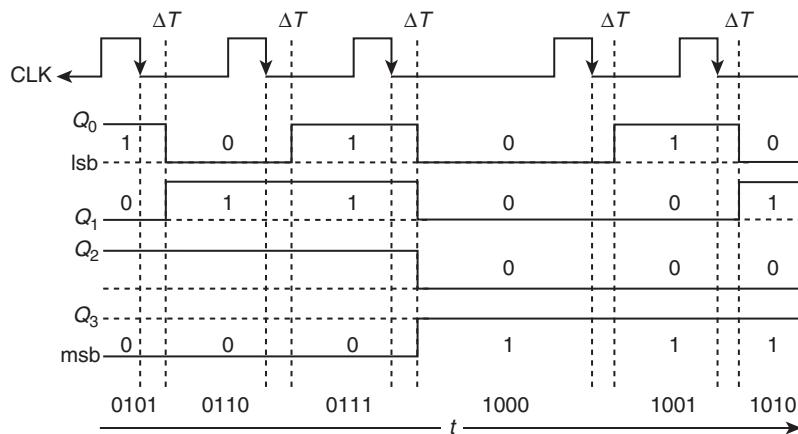


FIGURE 8.17 Timing diagrams in synchronous counter. (ΔT_{FF} effects also shown)

■ EXAMPLES

Example 8.1

Consider the circuit-a shown in Figure 8.18 A is a 4-bit parallel-in parallel-out register, which loads at each rising edge at the clock input C . The input to it is from a four-bit bus, W . The output from it acts as an input to a 16×4 ROM, whose output is floating when the enable input E is 0. A partial table of the contents at the ROM is as follows:

Address	0_d	2_d	4_d	6_d	8_d	10_d	11_d	14_d
data⁺	0011	1111	0100	1010	1011	1000	0010	1000

+ Four bits

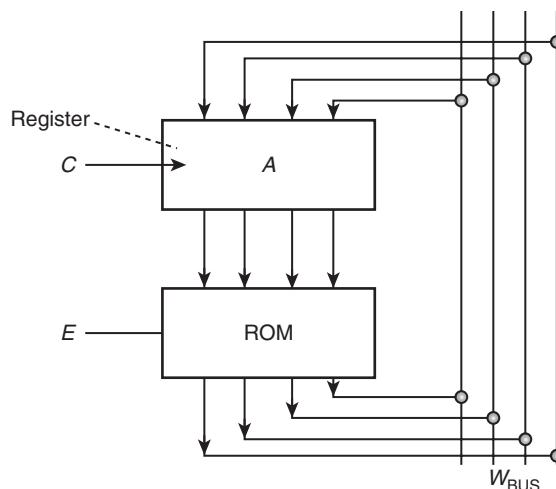


FIGURE 8.18 Circuit-a consisting of a register A and a ROM with register input and ROM output on a common set of four lines (bus W).

The clock to the register is shown, and the data on the W bus at first edge time t_1 is 0110. The data on the bus at second edge time t_2 is (A) 16.71 (B) 1011 (C) 1000 (D) 0010.

Solution

A common set of lines, used by a part of a digital circuit as inputs or outputs or both forms a bus. Usually these are four or eight or 16 or 32 lines and carry data or address bits. In present problem, there are four-lines for the 4-bit data.

Consider the time t_1 . $C = 0$. Register input is given to us as 0110 ($= 6_d$).

After the first +ve edge at the clock, the register output will become 6_d . Therefore the ROM output will be for address $= 6_d$. From the table given to us, it will be 1010 ($= 10_d$) at the W bus. This now becomes the input to the register.

After the second edge, register output will also become 10_d . Therefore ROM output for address $= 10_d$ from the table given to us will be 1000_b on the W bus.

Between t_2 and the second edge, there is no other positive clock edge. Hence the data on W bus at t_2 is also 1000_b . Therefore, answer (C) is correct.

Example 8.2

Study the problem in Example 8.1 once again. Suppose you use parallel loading serial right shift register, what will be the output on W bus at t_2 now. Assume that output for the next adjacent addresses of 0, 2, 4, 6, and 8 is complement of the one given in the table.

Solution

After the first +ve edge at the clock, the shift register output will become 0110 ($= 6_d$) shifted right, which will now equal 0011 ($= 3_d$). Therefore, ROM output for address $= 3$ from the table given to us will be 0000 (complement of the output at address $= 2$). W bus now becomes the input to the register. The register input will now be 0.

After the second edge, register output will also become 0011 shifted right, which will now equal 0001. Therefore, ROM output for address $= 1$ from the table given to us will be complement of 0011. Therefore, the output after the edge will be 1100.

Between t_2 and the second edge, there is no other positive clock edge. Hence the data on W bus at t_2 will be also be 1100.

Example 8.3

Design a 4-Bit buffer Register with parallel output after storing.

Solution

A buffer register has an additional feature. The data is clocked into the register by a CLK or strobe input. However, the data is actually placed on the output lines after the receipt of an input at the output enable (or control) pin. The buffer register is often made by placing an AND gate each between a Q output and a datum output (Refer Figure 8.2(a)). The one input terminal of each AND is connected to each Q output and the other terminal of all the ANDs coupled together and connected to an output (read) enable (or control or out) pin. If at the control pin, input logic is 0, the outputs from the register are 0s. If 1, then the Q s of the internal FFs appear at the register outputs. The buffer register is like a small memory unit in which the data bits can be written upon the activation of *store* or *load* or *write* signal at its CLK input, and the same can be read upon the activation of a output enable (or strobe or read or control) signal. Figure 8.2(a) showed a buffer register.

Example 8.4

Give the state table of a 4-bit PIPO register and then draw a state diagram of the PIPO when parallel-loading parallel-output register no shift.

Solution

Table 8.7 shows the state table of a PIPO and Figure 8.19(a) shows the state diagram.

TABLE 8.7 State table for a Moore model sequential circuit for a 4-bit PIPO register

Present state	Next state after transition (Q_1' , Q_2' , Q_3' , Q_4')				Present outputs Y_1 , Y_2 , Y_3 , Y_4
	Input (X_1 , X_2 , X_3 , X_4) = (0, 0, 0, 0)	Input (X_1 , X_2 , X_3 , X_4) = (0, 0, 0, 1)	Input (X_1 , X_2 , X_3 , X_4) = (1, 1 1, 0)
$S_i = S_0$ or S_1 or S_2 or ... S_{15}	S_0	S_1	S_{14}

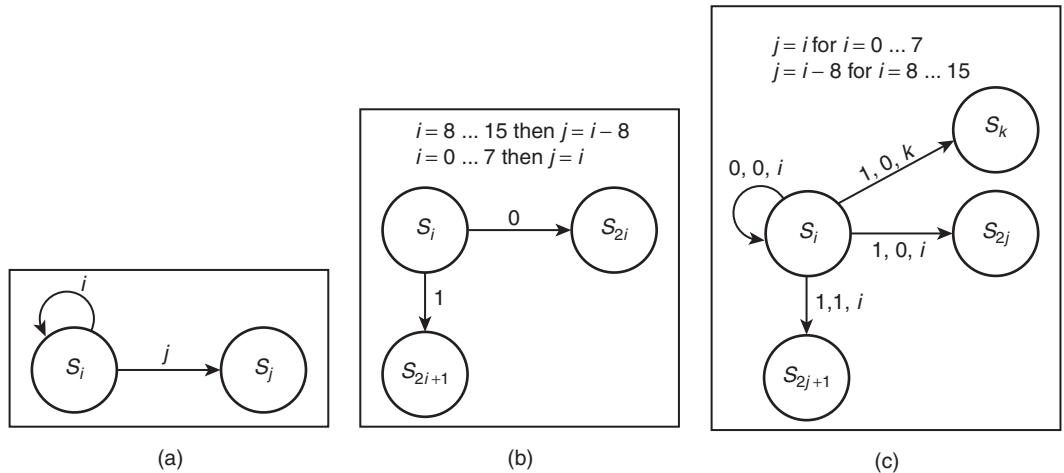


FIGURE 8.19 (a) State diagram of a Parallel-In Parallel-Out (PIPO) no shift register (b) State diagram of a SIPO left shift register (c) State diagram of a PISO left shift register.

Example 8.5

Explain Serial-In Parallel-Out (SIPO) left shift register by a state table.

Solution

Table 8.8 shows the state table for a left shift SIPO and Figure 8.19(b) shows the state diagram.

TABLE 8.8 State table for a Moore model sequential circuit for a 4-bit SIPO shift left register

Present state		Next state after transition (Q_0' , Q_2' , Q_3' , Q_4')		Present outputs Y_0 , Y_1 , Y_2 , Y_3
State	$Q_0 Q_1 Q_2 Q_3$	Input $X = 0$	Input $X = 1$	
S_0	0000	S_0	S_1	0000
S_1	0001	S_2	S_3	0001
S_2	0010	S_4	S_5	0010
S_3	0011	S_6	S_7	0011
S_4	0100	S_8	S_9	0100
S_5	0101	S_{10}	S_{11}	0101
S_6	0110	S_{12}	S_{13}	0110
S_7	0111	S_{14}	S_{15}	0111
S_8	1000	S_0	S_1	1000
S_9	1001	S_2	S_3	1001
S_{10}	1010	S_4	S_5	1010
S_{11}	1011	S_6	S_7	1011
S_{12}	1100	S_8	S_9	1100
S_{13}	1101	S_{10}	S_{11}	1101
S_{14}	1110	S_{12}	S_{13}	1110
S_{15}	1111	S_{14}	S_{15}	1111

The $Q_0 Q_1 Q_2 Q_3$ outputs will change after ΔT_{SR} as $Q_0 \leftarrow Q_1$, $Q_1 \leftarrow Q_2$, and $Q_2 \leftarrow Q_3$ in a left shift register upon a clock transition and serial datum bit is placed at Q_3 . Present state 0001 (S_1) will become 0010 (S_2) when serial bit = 0 and 0011 when serial bit = 1.

A SIPO shift-left register transfers the input bits X to next Q s at each clock edge excitation so that $Q'_j = X_p$ where $j = i + 1, j \leq n - 1$ and $j > 0$ and $i = 0, 1, 2, \dots, n - 1$ in an n -bit register. The lsb Q'_0 will become equal to serial bit. It gives parallel outputs, which are the same as the next state. When serial shift input $X = 0$, left shift register multiplies a binary number by 2. Figure 8.19(b) shows that state $S_i \rightarrow S_{2i}$ when serial in at $Q_3 = 0$.

Example 8.6 Explain a Parallel In Serial Out (PISO) left shift register using a state table.

Solution

Assume Q_0 is msb and Q_3 as lsb. Assume that weight of $Q_0 > Q_1, Q_1 > Q_2, Q_2 > Q_3$.

Table 8.9 shows the state table for a left shift PISO and Figure 8.19(c) shows the state diagram. When serial-in at $Q_3 = 1$ and load / shift = 0 then $S_i \rightarrow S_k$ where $k = i$ if i is even and $k = i + 1$ if i is odd. $Y \leftarrow Q_0$.

When serial-in at Q_3 and no shift then S_i does not change. When serial-in = 1 and shift = 0 then $S_i \rightarrow S_{2i}$. When serial-in = 1, shift 1 $S_i \rightarrow S_{2i+1}$. The $Q_0 Q_1 Q_2 Q_3$ outputs up on a clock edge transition, within a time equal to ΔT_{SR} from the transition,

TABLE 8.9 State table for a Moore model sequential circuit for a 4-bit PISO left shift register

Present state	Next state after transition (Q'_0, Q'_1, Q'_2, Q'_3)					Present outputs Y
	$\overline{\text{Load}}/\text{Shift},$ Serial-in	$\overline{\text{Load}}/\text{Shift},$ Serial-in at Q_r	$\overline{\text{Load}}/\text{Shift},$ Serial-in at Q_0	...	$\overline{\text{Load}}/\text{Shift},$ Serial-in	
$S_i = S_0$ or S_1 or S_2 or $\dots S_{15}$	$X_0, X_1,$ X_2, X_3 $= 000000$	$X_0, X_1,$ X_2, X_3 $= 100000$	$(X_0, X_1,$ $X_2, X_3)$ $= 110000$...	$(X_0, X_1,$ $X_2, X_3)$ $= 101111$	$\overline{\text{Load}}/\text{Shift},$ Serial-in $(X_0, X_1,$ $X_2, X_3)$ $= 111111$
S_0	S_0	S_1	...	S_{14}	S_{15}	$Y = Q_0$

Note: There are 6 inputs— $\overline{\text{Load}}/\text{Shift}$, serial-in from Q_3 and parallel inputs (X_0, X_1, X_2 and X_3). Q_3 is the lsb and Q_0 is msb.

will load the inputs into the Q_s when $\overline{\text{load}}/\text{Shift} = 0$. When the $\overline{\text{load}}/\text{Shift} = 1$, the register changes as a left shift register as $Q_0 \leftarrow Q_1, Q_1 \leftarrow Q_2, Q_2 \leftarrow Q_3$, and Q_3 will become equal to 0 when serial-in = 0 and equal to 1, when serial in = 1. Serial output will be from Q_0 assume Q_0 is of higher place (weight) value.

Parallel input 1010 (X_{10}) will become 0100 (S_4) when serial bit = 0 and serial-out Y will be = 1. Parallel input 0001 (X_1) will become 0010 (S_2) when serial-in bit = 0 and serial-out Y will be = 0. Parallel input 0001 ($X1$) will become 0011 (S_3) when serial-in bit = 1 and serial-out Y will be = 0.

Point to Remember

A PISO loads the parallel inputs bits X into the D s and gives the output $Q_j = D_i$ on the clock edge, when Load / Shift input = 0. Here j and $i = 0, 1, 2, \dots, n - 1$ in an n -bit register.

When $\overline{\text{Load}}/\text{Shift}$ input = 1, a PISO shift-left register transfers the D input bits X to next Q_s at each clock edge excitation so that $Q'_j = X_p$, where $j = i + 1; j \leq n - 1$. The lsb Q_3 will become equal to serial bit and post excitation and transition msb Q_3 will become the one-bit output Y at the serial-out pin. The parallel Q outputs are the same as the next state. When serial shift input = 0, left shift register multiplies the input binary number by 2.

Example 8.7

Explain a serial in serial out (SISO) circular right shift register.

Solution

A circular shift register feeds back the msb (last) stage Q_{n-1} input to the output of (next) stage Q_A . Table 8.10 gives a state table of a SISO circular shift register with serial input at the flip-flop with Q_{n-1} output and serial output at the flip-flop with Q_0 output. Q_0 is also the input S_{n-1} .

Figure 8.20(a) a n -bit SISO circular right shift register and Figure 8.20(b) shows its state diagram of a 4-bit circuit.

The $Q_3 Q_2 Q_1 Q_0$ outputs up on a clock edge transition, within a time equal to ΔT_{SR} from the transition, will load the serial input from Q_{i+1} into the Q_i and give serial output to Q_{i-1} stage.

TABLE 8.10 State table for a Moore model sequential circuit for a 4-bit SISO right shift circular register

Present state		Next state after transition (Q_1' , Q_2' , Q_3' , Q_4')		Present output $Y = Q_0$
State	$Q_3 Q_2 Q_1 Q_0$	Input $\bar{i}/s = 1$	Input $i/s = 0$	
S_0	0000	S_0	S_0	0
S_1	0001	S_8	S_1	1
S_2	0010	S_1	S_2	0
S_3	0011	S_9	S_3	1
S_4	0100	S_2	S_4	0
S_5	0101	S_{10}	S_5	1
S_6	0110	S_3	S_6	0
S_7	0111	S_{11}	S_7	1
S_8	1000	S_4	S_8	0
S_9	1001	S_{12}	S_9	1
S_{10}	1010	S_5	S_{10}	0
S_{11}	1011	S_{13}	S_{11}	1
S_{12}	1100	S_6	S_{12}	0
S_{13}	1101	S_{14}	S_{13}	1
S_{14}	1110	S_7	S_{14}	0
S_{15}	1111	S_{15}	S_{15}	1

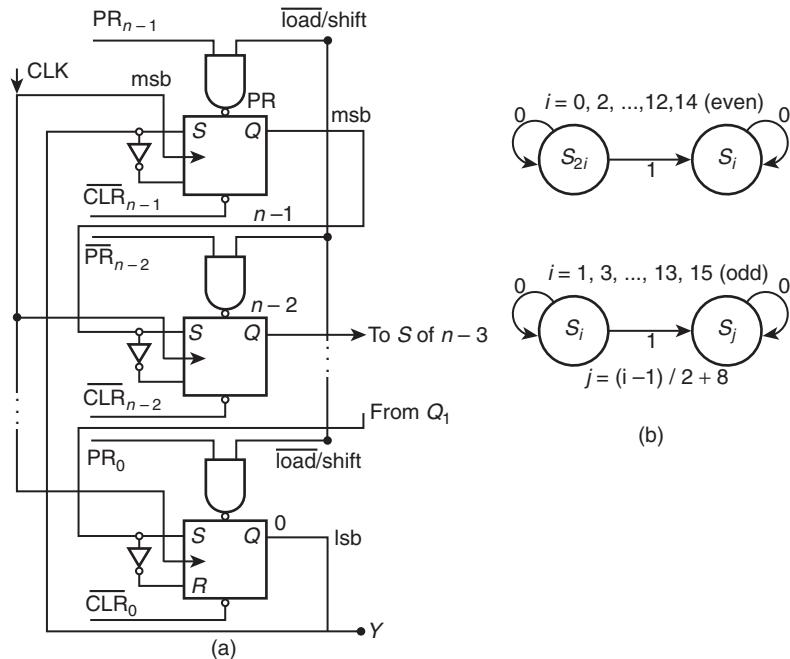


FIGURE 8.20 (a) n-bit SISO circular right shift register (b) 4-bit circuit State diagram.

The register changes as a shift register as $Q_3 \rightarrow Q_2, Q_2 \rightarrow Q_1, Q_1 \rightarrow Q_0, Q_0 \rightarrow$ serial out as well as Q_3 .

Serial input when state is 0010 (S_2) will become 0001 (S_1). Serial-out Y will be = 0. Serial input when state is 0001 (S_1) will become 1000 (S_8). Serial-out Y became 0 because after the transition $Q_0 = 0$.

When Q_{n-1} has a higher weight than other Q s in the binary number representation of output. A circular right shift SISO loads the inputs bit from Q_1 into the first stage D_0 and gives the serial output at Q_0 on a clock edge. Further, first stage gives the input to the last stage on a clock edge.

Example 8.8

Give the circuit of a left cum right shift cum parallel loading and parallel output universal register using the four multiplexers.

Solution

Figure 8.21 shows the circuit of a universal register using the four multiplexers. Each D flip-flop gets the D -inputs from a multiplexer output. We use four channel multiplexers. Each multiplexer has a four channel-input lines, one of which is selected at an instant for the output bit connected to D -input of a FF . Each multiplexer has two

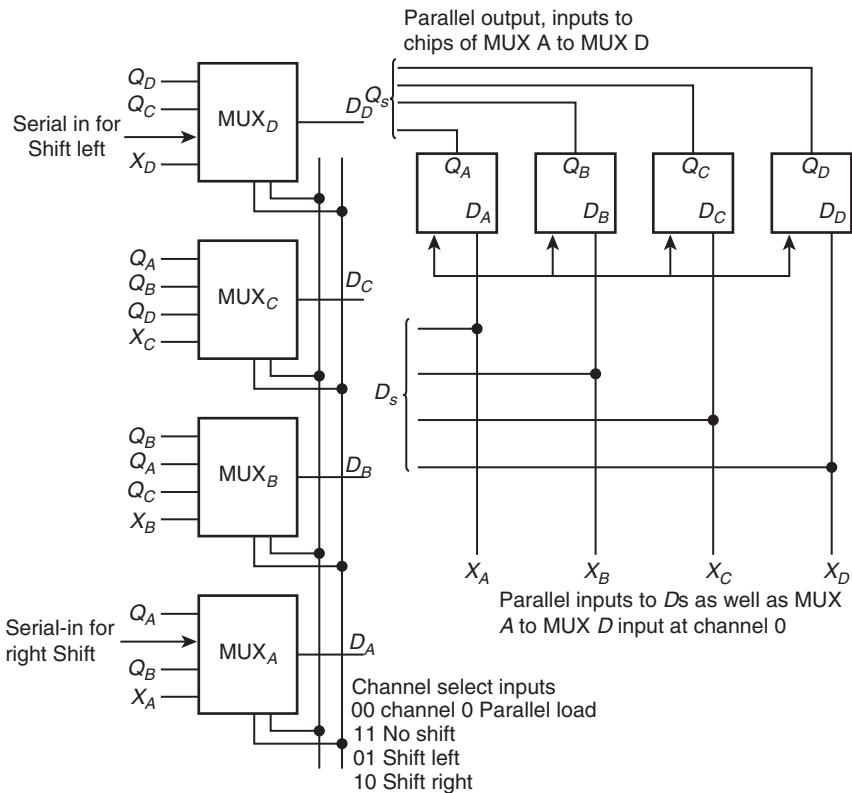


FIGURE 8.21 Circuit of a left cum right shift serial cum parallel loading and serial cum parallel output universal register using four number of multiplexers.

channel-address select bits for the four input channels. 00 selects channel A , 01 channel B , 10 channel C and 11 channel D .

1. MUX_A, MUX_B, MUX_C and MUX_D are given the parallel inputs X_A , X_B , X_C and X_D , respectively. These inputs are at the channel-A inputs. Therefore, when the channel address select bits are 00, the output on next clock transition will be the Q_A , Q_B , Q_C and Q_D as per D_A , D_B , D_C and D_D and these are equal to X_A , X_B , X_C and X_D , respectively.
2. MUX_A, MUX_B, MUX_C and MUX_D are given the inputs Q_B , Q_C , Q_D and serial-in bit for left-shift needs, respectively. These inputs are at the channel-B inputs. Therefore, when the channel address select bits are 01, the output on next clock transition will be as per the Q_B , Q_C , Q_D and serial-in bit at the D -inputs, D_A , D_B , D_C and D_D , which register at the Q_A , Q_B , Q_C and Q_D lines, respectively. Therefore, the next states are Q'_D = serial-in left shift bit, $Q'_C = Q_D$, $Q'_B = Q_C$ and $Q'_A = Q_B$, respectively.
3. MUX_A, MUX_B, MUX_C and MUX_D are given the inputs serial-in bit for right shift, Q_A , Q_B and Q_C , respectively. These inputs are at the channel C inputs. Therefore, when the channel address select bits are 10, for the output on next clock transition the serial-in bit for right shift, Q_A , Q_B and Q_C respectively, will be the D -inputs, D_A , D_B , D_C and D_D , which register at the Q_A , Q_B , Q_C and Q_D output state lines. Therefore, the next states are $Q'_D = Q_C$, $Q'_C = Q_B$, $Q'_B = Q_A$ and Q'_A = serial-in bit for the right-shift, respectively.
4. MUX_A, MUX_B, MUX_C and MUX_D are given the parallel inputs Q_A , Q_B , Q_C and Q_D , respectively. These inputs are at the channel-D inputs. Therefore, when the channel address select bits are 11, the output on net clock transition will remain as the Q_A , Q_B , Q_C and Q_D . This D_A , D_B , D_C and D_D for the next state Q'_S , Q'_A , Q'_B , Q'_C and Q'_D , are same as Q_A , Q_B , Q_C and Q_D , respectively.

Therefore we get a universal shift register, which is a parallel load PIPO when address bits to multiplexers are 00, left-shift PIPO when 01, right shift PIPO when 10 and holds (no-shift) when 11.

Example 8.9 Give a circuit to get a binary synchronous counter.

Solution

Refer the circuit of Figure 8.12. It gives a synchronous divide by 2^n . The Q outputs from the each FF available at the external output Q_S , as in us a binary counter. Each output of a FF has two states 0 or 1. The binary counter can count from 0 to $(2^n - 1)$ pulses. After every $(2^n - 1)$ pulse, at the next pulse the counter returns to the original state. The original state of all the outputs of the FFs can be made 0 by the inputs at the CLR.

Example 8.10 How will you design a binary counter using the ICs?

Solution

7493 is a standard TTL four stage binary ripple counter. Examples of various binary counters are 4040B (a 12 stage CMOS binary counter) and 4060 (a 14 stage CMOS

binary counter). 74HC93 or 74HCT93 are high speed CMOS variation of a 7493 counter of four stages.

Example 8.11 How will you make circuit of binary asynchronous up and down (U / \bar{D}) counters?

Solution

The up counter has the flip-flops which toggle or change on the negative (1 to 0) transition. When a JK MS FF is used for the UP counter, 0 to 1 transition at a clock input simply effects the master section of the FF and 1 to 0 transition in the slave section of that FF. The flip flop is, therefore, said to trigger at the negative edge of the clock pulse. Using a negative edge triggered D FF in method (ii) of Figure 8.6(a) also gives the identical effects.

While cascading FFs (i) if T FFs toggle on positive edge transition (0 and 1), (ii) if ' \bar{Q} ' outputs are connected to the T inputs, and (iii) if ' \bar{Q} ' outputs be used to indicate the states of the counter instead of Q, then we obtain a down counter. When PR input is activated, all ' \bar{Q} ' outputs become 0s. It is also possible that in an integrated circuit, we may call ' \bar{Q} ' as Q, CLR as PR and may offer it straight away as a down counter. 74HC469 and 744469 are the examples of ICs for the 8 bit (stage) up down counters.

Figure 8.22 shows two counters up and down using T FFs.

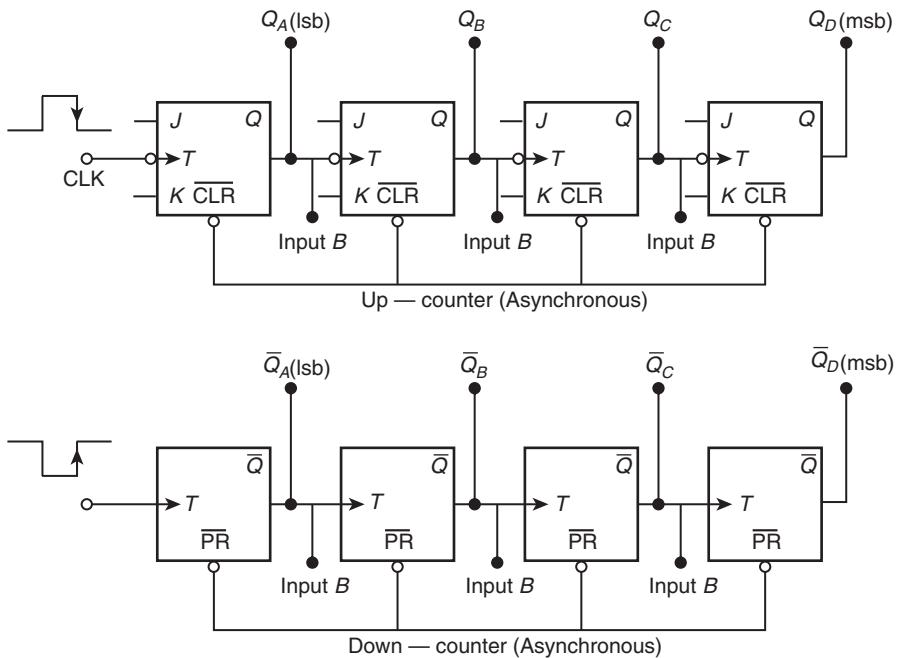


FIGURE 8.22 The circuits of the up and down asynchronous binary counters.

Example 8.12

Describe an asynchronous counter. What are the asynchronous counter disadvantages and advantages?

Solution

Consider the circuit in Figure 8.22(a). Upon the receipt of 16th input pulse, the flip flops A to D have to change the state. First Q_A is reset to 0, then after a while Q_B is 0, then after a while Q_C is reset to 0 and then after a while Q_D is 0. Let a flip flop needs time ' ΔT_{FF} ' to change the Q output. When a T -input is applied, the total time in changes of Q s at the sixteenth pulse at T -input is $4\Delta T_{FF}$. The time for the final correct state changes at Q s upon the eight pulses is $3\Delta T_{FF}$. The time for the Q s final change at the 4th pulse is $2\Delta T_{FF}$.

Disadvantage of using an asynchronous counter is that there is a variable time at which correct Q outputs of a counter appear. The Q_A changes at the faster rate than Q_B , the Q_B changes at the faster rate than Q_C , and so on. Readings of the combination $Q_D Q_C Q_B Q_A$ may differ during the period $4\Delta T_{FF}$. For fast changing pulses, the false readings during $4\Delta T_{FF}$ is obtained by such a counter.

An asynchronous counter does not have all clock inputs common in same logic state as in a synchronous counter.

Advantage of an asynchronous counter is that for multi-stage counting (for example, 16-bit) the fewer gates suffice and power dissipation in asynchronous counter is also small.

Example 8.13

How will you make circuit of modulo-10 decade counter?

Solution

IC 7490 or 74HC90 is the decade asynchronous modulo-10 ripple counter, which is very popular. Figure 8.23 shows its circuit diagram. Let Q_A and Q_D are 1, and Q_B and Q_C are 0. This counter has a within it a combinational logic such that when these outputs are present at the Q s at a next clock input pulse, the counter is cleared, and all output Q s are reset to 0. This counter has interesting facilities for changing its modulus, as we shall learn while performing an experiment with this counter.

Working of the Modulo – 10 (Decade) Counter

Let us refer to Figures 8.22(a) and (b). Let us join Q_A and input-B (CLK of next to lsb FFs). Let us also set inputs, R_0 (1) and R_0 (2) at 1. Let us also make at least one of the inputs Rg (1) and Rg (2) as 0. CLR becomes 0 because of a NAND gate shown at lower end of left hand side and PR moves to 1 because of a NAND on the lower end of the left hand side in circuit of Figure 8.22(a). This means that CLR inputs of all JK flip flops are activated, and this makes $Q_A = Q_B = Q_C = Q_D = 0$. Now let us make at least one of the R_0 and one of the Rg inputs at 0 logic levels. The CLR and PR inputs to all four FFs are therefore 1 (inactivated). So the J , K and CLK inputs are now effective. A -ve edge at the input A is now applied which is also the CLK input of FF_A . This will set Q_A to 1. As Q_A is connected to CLK of FF_B , at next input pulse at A , (i) the Q_A toggles to 0 and therefore a -ve edge occurs at CLK of

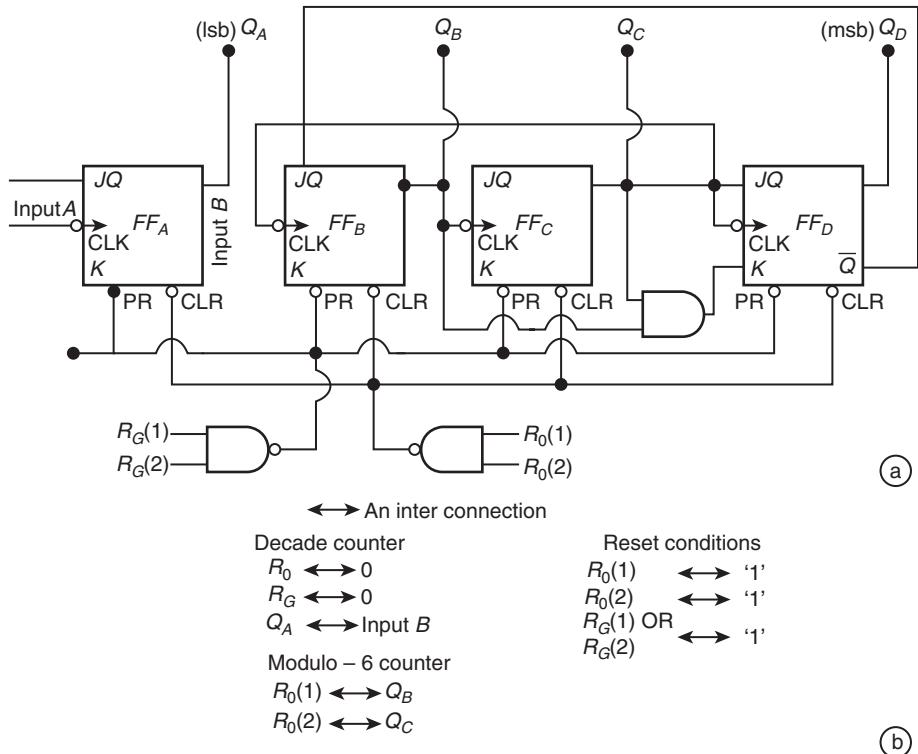


FIGURE 8.23 (a) The circuits of the decade counter using up and down asynchronous binary counters IC 7490
 (b) Reset conditions and uses of circuit as a decade counter, modulo-6 counter.

FF_B . At next -ve edge at input A , the Q_A will again change its state and does not clock the FF_B . At next -ve edge at input A , the Q_A will again change its state, but now clocks the FF_B , which toggles to 0. This feeds a -ve edge at FF_C which gives $Q_C = 1$. The outputs Qs indicate the number of -ve edges at input A . This continues up to nine -ve edges at input A , which means till $Q_A = 1$, $Q_B = 0$, $Q_C = 0$ and $Q_D = 1$. Q_B and Q_C are connected to a AND gate, the output of which is connected to K input of FF_D . As Q_C is connected to J input of FF_D , upon receiving 10th edge at input A , all outputs will become 0. This means that the configuration shown in Figure 8.22(a) acts as a decade counter.

Example 8.14 How will you make circuit of modulo-6 counter?

Solution

A modulo-6 counter can be built from IC-74HC90 as follows due to a special logic in it. For modulo-6 counter, we disconnect $R_0(1)$ and $R_0(2)$ from level 0s and connect it to Q_B and Q_C [refer Figure 8.22(b)]. As soon as both Q_B and Q_C goes 1 i.e. six

clock pulses are completed at the input A from the initial Q_s clear condition, the R_0 (1) and R_0 (2) will go 1. So the counter outputs Q_A , Q_B and Q_C are cleared and reset to 0 levels. The Q_D is immaterial in modulo-6-counter.

Example 8.15 How will you make a synchronous counter using MSI ICs?

Solution

TTL 74163 or 74HC163 is synchronous pre-settable binary up counter. 74HC193 or 74LS193 is up-down binary synchronous up-down counter.

Synchronous counter like 40161(b), 40162(b), 40163(b), 74HC161(b), 74HC193(b) are designed such that it has all of its T flip-flops change its Q outputs simultaneously. 74HC590 and 74HC592 are 8-bit synchronous counters (each of eight FF stages).

■ EXERCISES

1. Design a PIPO, which is a 4-bit buffer register with parallel in (loading) and parallel output (storing). [Hint: refer figure 8.2(a), use the AND buffers at both the ends.]
2. Show the state diagram from a state table corresponding to a SIPO. (Hint: Refer state table at Table 8.2).
3. Design a multipurpose behavior of a left shift register, which shift left when channel inputs, are 01 and circularly shift left when 10, synchronous clear when 11 and no-shift when 00. (Hint: Refer Example 8.8).
4. Solve the problem in Example 8.2 once again for a left-shift register.
5. Design a SISO circular left shift register.
6. Draw and compare the state diagrams of (i) a SIPO of 4 bits (ii) a 4-stage ripple counter.
7. Give a state table and state diagram of a binary synchronous counter.
8. Give a state table and state diagram of a asynchronous down modulo 6 counter.
9. How much is the time interval for Q_s to stabilize in a sixteen bit asynchronous counter? Assume flip flop propagation delays = 10 ns.
10. How will you make circuit of modulo-7 counter?
11. Show state diagram of a modulo-4 counter?
12. What should be the count value for a 8-bit counter to time out in 1 ms if input clock pulses at 100 μ s intervals?
13. Show the timing diagram of a ring counter.
14. Show the circuit of a 3-bit Johnson counter.
15. Show the timing diagram of a Johnson counter modified for maximum 7-sequence clock generator.

■ QUESTIONS

1. For a combinational circuit, the all-possible outputs are given in a truth table. For the clocked sequential circuits like counters, and registers. A state table describes the all-possible outputs. When do we use truth table and when do we use a state table? Explain by examples of the multiplexer and shift register.
2. What do we mean by state machines? Can we explain an exemplary state machine in which D FFs plus several logic gates put together gives us the various states at the FF outputs? (Hint: Shift register at every clock edge generates another state. Similarly a counter does so.)
3. What do we mean by a register?
4. How do we clear (erase) in the register? (Hint: Using \overline{CLR} input at the SR latches or FFs).
5. How do we store in the register?
6. What do we get at the outputs from the register when the outputs are in the tristates?
7. A D FF is a 1 bit register. Explain, Why can it be said so?
8. Explain circuit of a shift register, say, 74 HC175 or 74HC75.
9. What do we mean by shifting left (i) a binary word 1001 0101 (ii) word 0010 0101 1001 0100? If we shift twice then what will be the result.
10. Can we consider shift left four times as equivalent of multiplication by 16?
11. What do we mean by shift right? If we shift on right twice the following words, what will be resulting number? (i) 0100 0100 (ii) 0000 0011 1100 0011.
12. Can we consider shift right once means division by 2?
13. What is the application of 8-bit shift register? (Hint: In accepting serial data bits over a telephone cable and storing these in a computer or in transmitting a binary word digitally over a cable like a telephone cable).
14. What do we mean by clear input in a register?
15. What do we mean by SIPO, PISO, PIPO and SISO Shift registers? Explain with timing diagram (i) shift left in each (ii) shift right in each.
16. A ripple counter is very valuable as circuit for a frequency divider and when a slow response is tolerable outputs like in a digital clock. Why?
17. What is the main disadvantage in a ripple counter?
18. What are the differences between an asynchronous and a synchronous counter?
19. A synchronous counter provides output after, say, ΔT . An asynchronous counter like ripple counter of n FF stages will give a correct output after how much time.
20. What is the difference between an up counter and a down counter?

21. How is the different T flip-flops (or JK FF with $J = 1$ and $K = 1$) cascaded to obtain a counter?
22. What is a binary counter? Explain its circuit
23. What are the applications of a binary counter?
24. What is a decade counter? Explain its circuit.
25. What are the applications of a decade counter?
26. How do we convert a binary counter into a decade counter and into a divide-by-6 counter? How can such circuits be used in a digital clock?
27. What do we mean by (i) a ripple counter and (ii) a ring counter? Explain their circuits.
28. How will we divide input pulses by four?
29. What do we mean by the JAM inputs to a counter or a register?
30. Suppose we wish to connect a mechanical switch undergoing up down transition several times. Why do we connect a debouncing circuit in between contact switch and a counter? What will be advantage of a monostable circuit in-between? Can an Schmitt Trigger be also used in-between?
31. A counter can also be said to act as a register, which holds the number of input pulses with respect to the contents at the beginning. In a computer, a program counter is such a register or a stack pointer is another such register why?
32. What is the use of carry in and carry out pins in an IC of a synchronous counter?
33. Why do we sometimes incorporate a lockout time (dead time) count inhibit circuit at the count pulses at the counter input by using a monostable?
34. A counter can also be considered as a state machine (a sequential device which provides at an instant a set of outputs followed by another instant another set of outputs). Explain, why can we consider so?
35. Why can't a D latch with \bar{Q} feedback to its D input be used as a T FF? In fact, we get oscillations of several tens of MHz at Q in such a circuit. Why? (Hint: After each time interval equal to propagation delay between D input and Q output, we get a toggling till such time when clock input level is active).
36. Why are asynchronous counters made from -ve edge triggered FFs and synchronous counters for +ve edge triggered FFs? (Hint: Easy interconnectivity between two FFs in an asynchronous between two FFs).
37. A ripple counter can be used to generate long duration pulses instead of using a monostable. How does it happen? Explain it. (Hint: For example, 14 stage binary counter like CMOS 4060 will give $2^{14} T$ duration pulse at its last stage if T is time interval between two successive pulses at its T input).
38. Describe synchronous counter advantages.
39. Describe ring counter applications.

CHAPTER 9

Fundamental Mode Sequential Circuits

OBJECTIVE

In this chapter, we shall learn the asynchronous sequential circuit operations in a mode called fundamental mode—their stable and unstable states, analysis, output specifications, feedback cycles, races, race-free assignments in their design and circuit implementation.

We learnt in Chapter 7 the following concepts:

1. A sequential circuit is a circuit made up by combining the logic gates such that the required logic at the output(s) depends not only on the current input logic conditions but also on the past inputs, outputs and sequences.
2. A sequential circuit has the memory elements like flip flops and a combinational circuit(s) has no memory elements.
3. A general sequential circuit network has a memory section and the combination circuits at the memory inputs and outputs.

We learnt in Chapter 7 two classifications of the sequential circuits:

- (a) **Synchronous sequential circuit** is a circuit in which the output \underline{Y} depends on present state \underline{Q} and present inputs \underline{X} at the clocked \uparrow or \downarrow or $\overline{\uparrow}\overline{\downarrow}$ (MS) or $\overline{\uparrow}\overline{\downarrow}$ (MS) instance(s) only. Memory section activates a transition as per the excitation inputs to the next state \underline{Q}' . A clock input (or a set of inputs) is used, which activates the transition to next state. [There is a special case, called clocked sequential circuit. There is a master clock in the circuit,

which activates the memory section.] *Synchronous sequential circuit clock input (s) controls the instance(s) at which an output(s) changes.*

- (b) **Asynchronous sequential circuit** is a circuit in which not only the present inputs \underline{X} and present state \underline{Q} but also the sequences of changes affect the output \underline{Y} . Asynchronous means that the changes can be at the undefined instances of time. *There are no controls for the instance at which the output(s) change.*

9.1 GENERAL ASYNCHRONOUS SEQUENTIAL CIRCUIT

A general asynchronous sequential-circuit has a section consisting of the (i) latches without clock inputs (or an equivalent delay device) and the (ii) combinational circuits at the input and output stages. It means a memory section of the clocked sequential circuit replaces the latches without the clock inputs (or an equivalent delay device). Assume the followings structure of a general sequential circuit.

1. There is a set \underline{X}_i of m present input variables X_0, X_1, \dots, X_{m-1} . These are applied along with the present \underline{x}_q to a combinational circuit. The output of \underline{X}_i along with \underline{x}_q is a set of present state outputs \underline{Y} of which \underline{y}_q is a subset. (\underline{Y} consists of two subsets; \underline{Y}_j with outputs $Y_0, Y_1 \dots Y_{j-1}$ with no feedback and \underline{y}_{q-} with feedback to the input via the memory section).

The circuit memory section consists of the m latches without any clock edges or pulse inputs to control the instance or period of their operations. These have a set \underline{x}_q of variables from the m present state outputs. The $\underline{x}_q = x_{q0}, x_{q1} \dots x_{qn-1}$ from the \underline{y}_{q-} inputs after a delay.

2. The present state of \underline{Y} and therefore, its subset \underline{y}_q changes by the change in \underline{x}_q after a delay at the memory section. Changed \underline{Y} can change again after the next feedback-cycle of delay (in the second feedback-cycle at the memory section). The feedback-cycles after first feedback-cycle may continue or may not continue; there may be unstable or stable \underline{Y} due to unstable or stable \underline{y}_q , respectively.

Figures 9.1(a) and (b) show (i) a representation for the general asynchronous sequential circuits, which has above features and a special case of fundamental mode operation (ii) A sequential asynchronous circuit is also like a machine producing the states without a clock controlling that.

Point to Remember

Asynchronous sequential circuit is circuit in which there are no control(s) for the instance(s) at which the output(s) change like a control by a clocked instance in synchronous sequential circuit.

9.2 UNSTABLE CIRCUIT OPERATION

Step 1: Let us also assume that the input X change applied only at stable \underline{x}_q . Let the present state \underline{Y}_0 and therefore its subset \underline{y}_q changes when a bit changes

\underline{X}_i to \underline{X}'_i . Therefore, \underline{Y}_0 changes to \underline{Y} after a normal gate propagation delays (not in consideration).

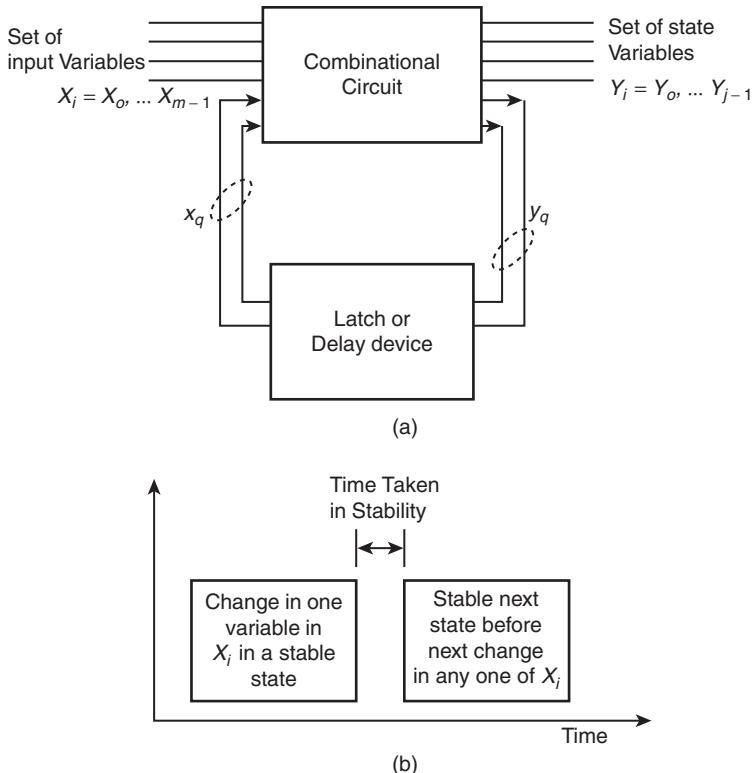


FIGURE 9.1 (a) General asynchronous sequential circuit with a section consisting of the (i) Latches without clock inputs and the (ii) Combinational circuits at the input and output stages (b) Above circuit fundamental mode operation with an assumption that one variable changes at an instance.

Step 2: However, in the first feedback-cycle, a delayed change in x'_q occurs due to change in y_q to y'_q , being a subset of \underline{Y} . The feedback x'_q after a memory-section delay combined with \underline{X}'_i changes \underline{Y} to next state \underline{Y}' .

Step 3: However, in next feedback-cycle a delayed change in x''_q may also occur due to change to y''_q , a subset of \underline{Y}' . The feedback x''_q after the feedback-cycle of delay combined with \underline{X}'_i may therefore change \underline{Y}' to next state \underline{Y}'' .

After the step 2 feedback-cycle if there are changes to the next state(s), the \underline{Y} is said to be unstable till after in any further feedback-cycle there is no change.

Figures 9.2(a) shows a circuit one D latch and two buffer gates that remains unstable in successive feedback-cycles, when X_0 change from 0 to 1.

Let instance when X_0 changes from 0 to 1 is after stable $x_0 = 1$. The buffer gives the output $Y_0 = 1$ and $y_1 = 1$. The y_1 in first feedback-cycle after a delay gives the output $= x'_0 = 0$. Therefore, now y_1 now becomes y'_1 and is 0. Now when $y_1 = 0$, the x''_0 will become = 1. The x_0 continues to toggle after successive delays. The circuit is unstable till infinity.

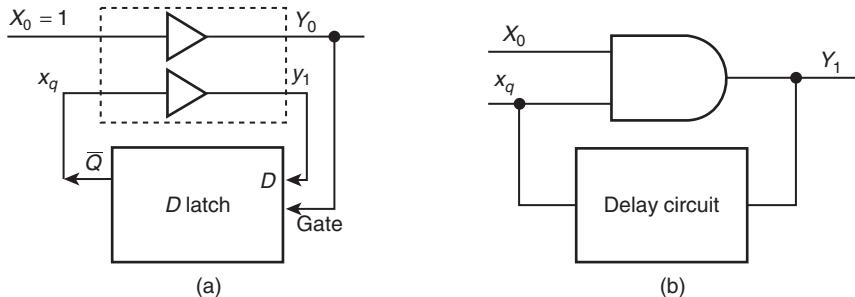


FIGURE 9.2 (a) Unstable asynchronous sequential circuit (b) Stable asynchronous sequential circuit.

If after first feedback-cycle or an $(n + 1)^{\text{th}}$ feedback-cycle, if the Y does not change, then the circuit is said to be stable. The n can be 0 to infinity—the circuit can be permanently unstable.

Point to Remember

Asynchronous sequential circuit is called unstable when the next state changes repeatedly without settling to a stable state.

9.3 STABLE CIRCUIT ASYNCHRONOUS MODE OPERATION

Figures 9.2(b) shows a circuit that becomes stable after first feedback-cycle itself. Let when X_0 changes from 0 to 1 when there is stable $x_q = 1$. The AND gives the output $y'_1 = 1$. The y'_1 in first feedback-cycle after a delay gives the output = $x'_q = 1$. Therefore, now y''_1 will be 1 after the first feedback-cycle. In subsequent feedback-cycles also, there is no change unless the input X_0 again changes to 0 again.

Point to Remember

Asynchronous sequential circuit is called a stable circuit when after an input change the next settles to a stable state in first or known number of memory section (latches) feedback-cycles.

9.4 FUNDAMENTAL MODE ASYNCHRONOUS CIRCUIT

Asynchronous sequential circuit operates as a fundamental mode operating circuit when no input variable(s) \underline{X}_i is changed unless the \underline{x}_q and \underline{y}_q are stable. An assumption useful for an analysis of fundamental mode circuit is that only one input variable (one input bit) change at an instance for changing one state of \underline{Y}_q to the next state of \underline{Y} . Figure 9.1(b) showed a fundamental mode circuit operation with this assumption.

Point to Remember

Fundamental mode of an asynchronous sequential circuit is a mode in which the input(s) changes only when the circuit is in stable state.

An assumption useful for an analysis of fundamental mode circuit is that only one input variable (one input bit) change at an instance for changing one state to the next state.

9.4.1 Tabular Representation of Excitation-cum-Transitions of States and Outputs

Consider the transition table for a clocked sequential circuit in Table 7.8 once again for another exemplary asynchronous circuit operating in fundamental mode. Let us forget about the actual circuit, and the values shown in the table. We just first learn how to show a stable state, transition to an internal state(s) and transition to next stable state. Let us just also learn how to show a stable output undergoing transition to an internal output(s), and transition to next stable output. Table 9.1 shows the excitation-cum-transitions by markings by squares or box and directed arcs to show the changes from one stable state S (and output Y) to another though the intermediate instability feedback-cycles.

TABLE 9.1 Excitation-cum-transition table for an exemplary asynchronous sequential circuit operating in fundamental mode

Present state (x_{q0}, x_{q1})	Next state after transition (x'_{q0}, x'_{q1})		Present output Y_0	
(x_{q0}, x_{q1})	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$
0, 0				
0, 1	0, 1	1, 0	1	0
1, 0	0, 1	1, 1	1	1
1, 1	0, 1	0, 0	1	1

A stable state is marked by a square or circle or box over the state variables or state names. Horizontal directed arc(s) with arrow shows the action at the combinational circuit. Vertical directed arc(s) with arrow shows the action at the memory section by the latches. Number of vertical arcs equal the number of feedback-cycles after which stability is achieved.

A stable state shows by an square or box or circle surrounding the (x'_{q0}, x'_{q1}) in a column. The table shows that $(0, 0)$ is an initial stable state. On changing $X = 0$ to $X = 1$, it changes to state $(1, 0)$ at first feedback-cycle combinational

circuit action shown by horizontal arc. Assume that state $1,0$ is unstable after the memory section first feedback-cycle transition to next state (x'_{q_0}, x'_{q_1}) . It changes in the feedback-cycle to $(1, 0)$ unstable (intermediate or internal) state. Now after the second feedback-cycle transition to another next state $(1, 1)$, the state is stable as shown by the box or circle over it.

A stable output Y_k ($k = 0$) shows by a box (or circle) surrounding the (Y_k) . The table row 1 shows that (0) is an initial stable Y_0 . On changing $X = 0$ to $X = 1$, it changes to state (0) at first feedback-cycle combinational circuit action by horizontal arc. Assume that output Y_0 is unstable after the memory section first feedback-cycle transition to next state. The Y_0 changes in the feedback-cycle to (0) unstable (intermediate or internal) case output in row 2. Now after the second feedback-cycle transition to another next output (1)—the output is stable as shown by the box or circle over it in row 3 last column.

Table 9.1 showed that in the given circuit whose table it is, the stability in the state or output is shown to reach in two feedback-cycles of actions at the combinational circuit (horizontal arc) followed by memory section in each feedback-cycle (vertical arc). Starting point is always a stable state. Therefore, the operation is the fundamental mode operation because in fundamental mode an input is changed when the circuit is in stable state. It is also seen that the end-state and end-output in the present case one also stable after the excitation cum transitions.

Recall the sequential circuit-4 in [Figure 7.3(b)]. Now, if the control by the clock edge inputs are removed, the circuit will be asynchronous mode circuit. Let J - K in Figure 9.3(a) become a J - K latch now. Figure 9.3(a) shows circuit-4 for

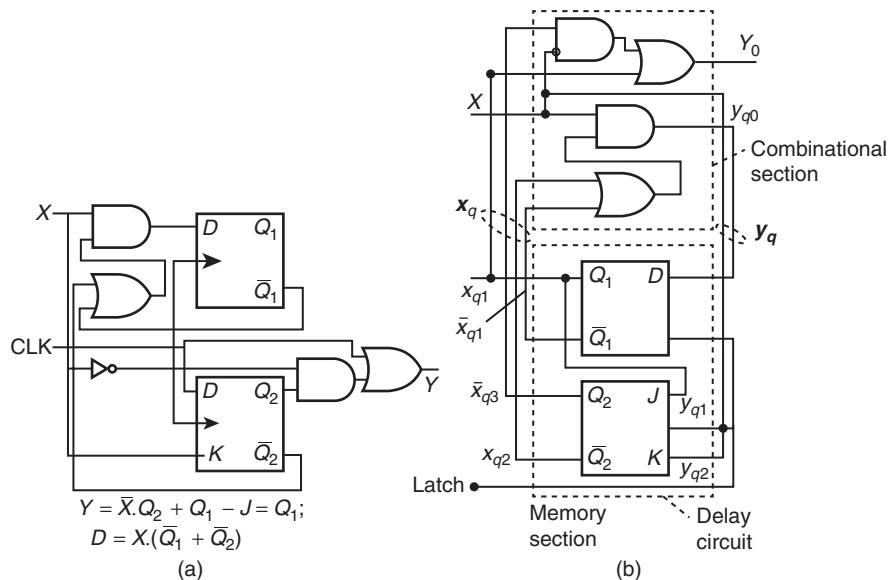


FIGURE 9.3 (a) Circuit-4 of Figure 7.3 (b) Redrawn as asynchronous sequential circuit-5 withdrawing the clock input and replacing JK FF by a JK latch.

an analysis later in fundamental mode. Figures 9.3(b) shows using one D latch and one $J-K$ latch an exemplary un-clocked sequential circuit-5.

Following are the new inputs, present states, outputs, input functions and output function for the circuit-5:

$$\text{Four outputs: } Y = \bar{X} \cdot \bar{x}_{q2} + x_{q1}; y_{q0} = (\bar{x}_{q1} + x_{q2}) \cdot X; y_{q1} = \bar{x}_{q1}; y_{q2} = X \quad \dots(9.1)$$

$$\text{Three inputs: } X, x_{q1} = D \text{ and } \bar{x}_{q2} = J \cdot \bar{x}_{q2} + \bar{K} \cdot \bar{x}_{q2}, \quad \dots(9.2)$$

$$\text{Delay section: } D = y_{q0}; J = y_{q1}; K = y_{q2} \quad \dots(9.3)$$

There are four outputs Y, y_{q0}, y_{q1} and y_{q2} . There are three inputs; one external input and two state-output as feedback inputs \bar{x}_{q1} and \bar{x}_{q2} .

9.5 ANALYSIS PROCEDURE

An analysis is important for understanding the behavior of an asynchronous sequential circuit. Analysis also provides a tabular representation of the circuit. A seven step method for analysis is as follows:

1. Draw logic circuit diagram to be analyzed. [For example, circuit in Figure 9.3(a)].
2. Circuit should be drawn such that the feedback-loop is clear and feedback loop should be minimized to have minimum feedback inputs and outputs. This helps in simplification of the analysis. [For example, circuit in Figure 9.3(b) only the x_{q1} and \bar{x}_{q2} are the state representing inputs and only three y_{q0}, y_{q1} and y_{q2} inputs to memory section are there. See the contrast to this circuit with the circuit in Figure 9.3(a), which has more state outputs $Q_1, \bar{Q}_1, Q_2, \bar{Q}_2$].
3. Perform state variables \underline{x}_Q assignments and excitation variables \underline{y}_Q (means latche and other delay-circuit element inputs) assignments. [For example state variable assignments x_{q1} and \bar{x}_{q2} and excitation variables y_{q0}, y_{q1} and y_{q2} in Figure 9.3(b)].
4. (i) Find the expressions for the excitations from the flip flop characteristics equations as per the excitations. In other words, find $\underline{Y} = F_o(\underline{X}_i, \underline{x}_Q)$ and thus \underline{Y}_i and \underline{y}_Q . [For example, find expressions given Equation 9.1 for circuit-5.].
(ii) Make an excitation table.
5. Make an excitation-cum-transition table (or simply a transition table) from the expressions for next state inputs $\underline{x}_Q' = F_o(\underline{X}_i, \underline{x}_Q, \underline{y}_Q)$ (For example, Equation 9.2 for circuit-5).
6. Draw a flow diagram by making the state, flow and primitive flow tables.
7. Find reduced flow table for the restricted inputs.

Examples 9.1 to 9.3 will explain the analysis procedure for circuit of Figure 9.3 – how to make the excitation table, transition table and state table, respectively. Example 9.4 will explain why is it that the flow table can't be constructed for this circuit.

Definitions for the different terms used during the analysis of the circuit or during the design and synthesis for implementing the circuit mentioned above are as follows:

9.5.1 Excitation Table

An excitation table is a tabular representation of the present state \underline{x}_q , \underline{y}_q and \underline{Y}_i at the delay and combinational sections. The table is a tabular representation of present state functions F_Q and output function F_O . Present state defines by a specific combination of state variables \underline{x}_q . It gives present state inputs given from the memory section as per column 1.

First column of excitation table gives a present state (x_{q0}, x_{q1}) in its each row. Number of rows in each column equals 2^n where n is the number of delay-section outputs. Delay section may consist of either latches (FFs without clock for synchronizing) or simple set of gates to create propagation delays for \underline{x}_q longer than the combinational circuit section \underline{y}_Q and \underline{Y}_i in Figure 9.1. For example, if (x_{q0}, x_{q1}) are the states at the memory section, then $(x_{q0}, x_{q1}) = (0, 0), (0, 1), (1, 0)$ and $(1, 1)$ are the four combinations possible for the four different states of the memory section.

For each state at a row, the excitation inputs \underline{y}_Q are at the columns 2 to column $(1 + 2^m)$, where 2^m is the number of possible combinations in set of inputs \underline{X}_i . For each state at a row, the outputs \underline{Y}_i are at the columns $(2 + 2^m)$ to column $(1 + 2^{m+1})$. It also gives the outputs that precede the excitations at the memory section.

Example 9.1 will give an exemplary excitation table.

9.5.2 Transition Table

A transition table is a tabular representation of next state output as per functions F_Q and F_O . It differs from excitation table in the middle section columns; columns 2 to column $(1 + 2^m)$, where 2^m is the number of possible combinations in set of inputs \underline{X}_i . It shows how the delay circuit or latches will respond to all the present inputs \underline{y}_q and will generate \underline{x}'_q .

Number of rows in each column equals 2^n for the n -delay section outputs.

First and columns $(2 + 2^m)$ to column $(1 + 2^{m+1})$ of transition table are same as excitation table.

The number of columns for the next state equals the number of possible combinations of external inputs in the set \underline{X}_i . It equals 2^m . For each set of inputs, there is a set of memory section next state variables after the transition at the memory section. For example, if $(x_{q0}, x_{q1}$ and $x_{q2})$ are the states at the memory section, then $(x_{q0}, x_{q1}$ and $x_{q2}) = (000), (001), (010), (011) (100), (101), (110)$ and (111) are the eight combinations possible for the eight different states of the memory section. [If there are two external inputs, there will be 32 cells in columns 2 to 5.]

Table 9.2 shows transition table of the Table 9.1, which was the excitation-cum-transition table for an exemplary asynchronous sequential circuit operating in fundamental mode.

TABLE 9.2 Transition table for an exemplary asynchronous sequential circuit operating in fundamental mode

Present state	Next state after transition $(x'_{q_0}, x'_{q_1})^+$		Present output Y_0	
	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$
$(x_{q_0}, x_{q_1})^*$				
0, 0	0, 0	1, 0	0	0
0, 1	0, 1	1, 0	1	0
1, 0	0, 1	1, 1	1	1
1, 1	0, 1	0, 0	1	1

*Stable state is marked by box over the state variables. x_{q_1} and \bar{x}_{q_2} in Figure 9.3(b) (equation 9.2)

Example 9.2 will show construction of transition table using state variables (x_{q_1}, \bar{x}_{q_2}) and excitation table for circuit-5 (Figure 9.3(b)).

9.5.3 State Table

All columns of state table are same as transition table except that instead of state variables, the states are given.

Each set of $(x_{q_0}, x_{q_1}$ and $x_{q_2})$ is assigned a state by a circuit designer. For example, S_0 , S_1 , S_2 , ..., and S_7 in a three-state variable case. A state table is a tabular representation of the present state as per the assignments of state variables to a state.

Present states are at the column 1. Next states after the transition are in a set of succeeding 2^m columns where m is the number of possible sets of the external inputs. The outputs are shown in next set of 2^m columns.

Number of rows in each column equals 2^n for a memory section of n state-variables.

For example, if (x_{q_0}, x_{q_1}) are the state variables at the memory section, then let us assign the followings:

$$S(x_{q_0}, x_{q_1}) = S(0, 0) = S_0 \text{ for } (0, 0) \text{ values of } (x_{q_0}, x_{q_1}).$$

$$S(x_{q_0}, x_{q_1}) = S(0, 1) = S_1 \text{ for } (0, 1) \text{ values of } (x_{q_0}, x_{q_1})$$

$$S(x_{q_0}, x_{q_1}) = S(1, 0) = S_2 \text{ for } (1, 0) \text{ values of } (x_{q_0}, x_{q_1}), \text{ and}$$

$$S(x_{q_0}, x_{q_1}) = S(1, 1) = S_3 \text{ for } (1, 1) \text{ values of } (x_{q_0}, x_{q_1}).$$

A state in a row on first column is now either S_0 , S_1 , S_2 or S_3 for the row 1, row 2, row 3 or row 4, respectively. [These are as per four possible values of (x_{q_0}, x_{q_1}) in case of two state variables used at the memory section.]

The number of columns for the next state S equals the number of possible combinations of external inputs in the set \underline{X} . It equals 2^m . For each set of inputs, there is a set of memory section states resulting after the transition at the memory section, for example, corresponding to each set of external inputs, there will be four sets of next states. A state in a row is either S_0 or S_1 , S_2 or S_3 as per the post transition values of next state (x'_{q_0}, x'_{q_1}) .

Table 9.3 shows state table corresponding to the Table 9.2 transition table.

TABLE 9.3 State table for an exemplary asynchronous sequential circuit operating in fundamental mode

Present state (x_{q_0}, x_{q_1})	Next state after transition $(x'_{q_0}, x'_{q_1}) S^*$		Present output Y_0	
	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$
0, 0 S_0	S_0	S_2	0	0
0, 1 S_1	S_1	S_2	1	0
1, 0 S_2	S_1	S_3	1	1
1, 1 S_3	S_1	S_0	1	1

* Stable state is marked by circle or box over the state name.

Example 9.3 will show the construction of state table from the state table for circuit-5.

9.5.4 State Diagram

A state diagram is diagrammatic representation of the state table. Each element of a set of present $(x_{q_0}, x_{q_1}, \dots, x_{q_{n-1}})$ is denoted by a state. There are $z (= 2^n)$ maximum possible states $S_0, S_1, S_2, \dots, S_{z-1}$ in asynchronous circuit with z -memory or delay section circuits. (The 2^n states can be reduced by state minimization method of finding the equivalent states. In that case state table will have less number of rows and state diagram less number of nodes).

1. States S_0, S_1, S_2 and S_3 are labeled at the centers of each circle representing a node.
2. Each arc or circular arc is labeled with present input and the present output at the transition.
3. Each arc or circular arc can have more than one set of (pre-transition input/post transition output) labeled on it if there are more than one sets of (pre-transition input/post transition output) that are having the same transition from a node to another.
4. The number of nodes = number of rows in the state table.
5. For two flip flops, there are four states S_0, S_1, S_2 and S_3 . So four circles are drawn for the four nodes of a graph.
6. A directed arc from the present state node to the next state node shows a transition.
7. A small diameter circular directed arc marks a transition in which the state remains unchanged.
8. The number of directed arcs equals the number of transitions in which the state changes.
9. The number of directed circular arcs equals the number of transitions in which the state does not change.

Figure 9.4(a) shows the state diagram corresponding to an exemplary state table in Table 9.3. Figure 9.4(b) shows its flow diagram and 9.4(c) shows the races due to feedbacks. [Sections 9.5.8 and 9.6.]

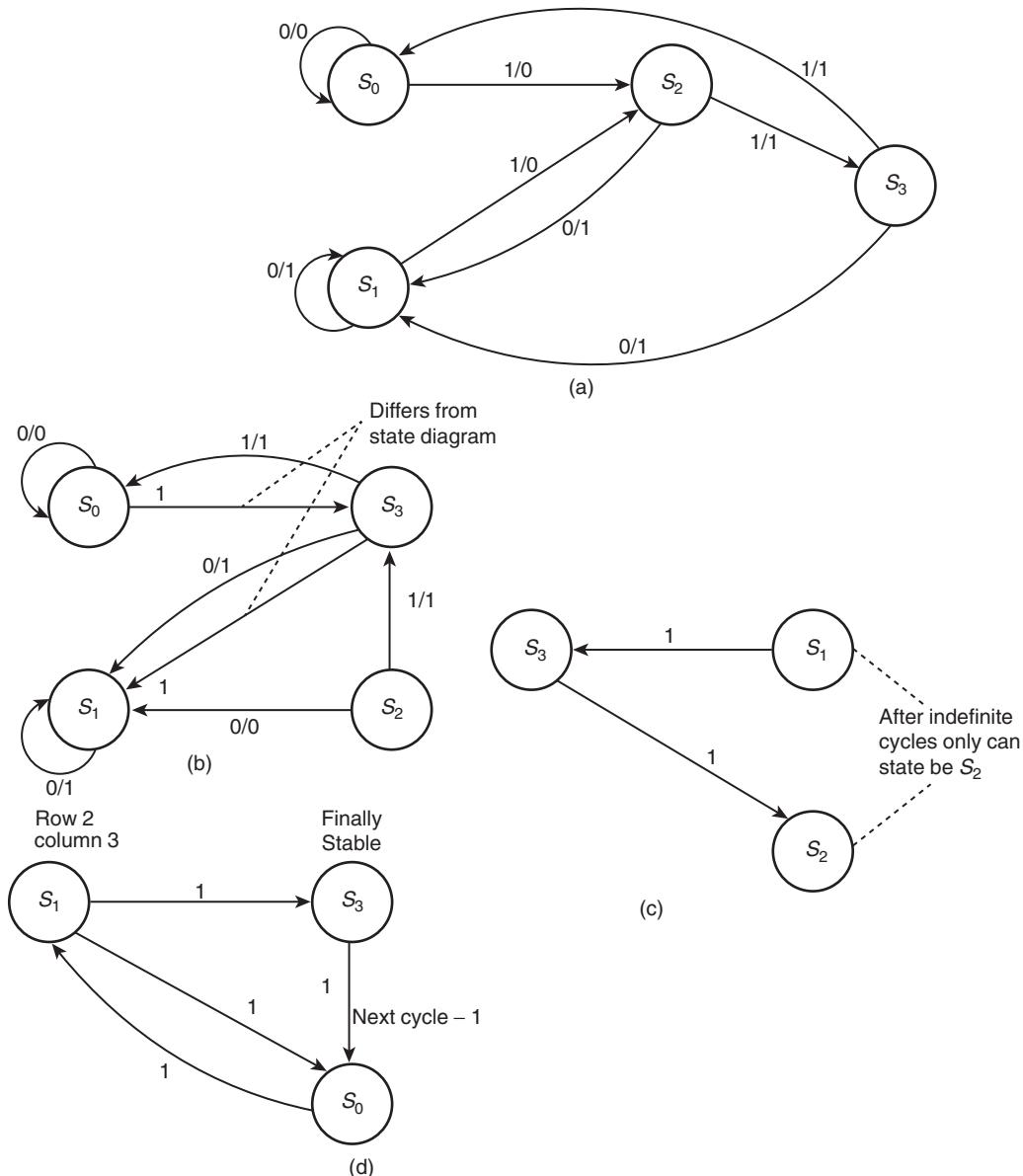


FIGURE 9.4 (a) State diagram for asynchronous circuit having state table for as per Table 9.3. (b) Flow diagram using flow table 9.4 constructed for circuit-5 of excitation-cum-transition table in Table 9.1. (c) Races arising due to the feedback next-state variables, which don't occur simultaneously when two bits change at the delay section during change when a state changes. (d) An alternate flow diagram.

9.5.5 Flow Table

A state table gives the following:

1. A state table gives the stable next state and output before this state. This state is that which finally exists after one or more feedback-cycles of next states from the memory section after change of an input X_k . That is fine when the finally what state(s) is achieved that matters.
2. However, a state table gives the intermediate state(s) also, which are unstable (refer vertical directed arcs in column 3 of Table 9.1) and leads to other unstable state(s) to which the asynchronous circuit is passing through in case of more than one feedback-cycles at the memory section.
3. A state table does give the output at the intermediate unstable state(s) also (refer vertical directed arcs in column 5 of Table 9.1).
4. A state table gives all the next state(s) whether a certain set of inputs is specific as nonexistent after a certain sequence of set on inputs or existent. For example, when two simultaneous input bit changes not permitted the 01 will not follow 11 for X_0 and X_1 . However, state table will mention that state also.

A state table does not give the following:

1. A state table does not give the information about state that will not exist due to certain specified input constraints as it gives all cases entries.
State table therefore gives too much. It does not provide the information about the flow (next sequence) of the stable states and outputs for the final stable existence after one or more feedback-cycles of next states from the memory section after change of an input X_k . For a condensed view of the asynchronous circuit behavior, the flow part is required.

A flow table gives the following:

A flow table gives the present state, for the different input combinations next states (stable) and present outputs, which corresponds to ones leading to stable state. These states exist after one or more feedback-cycles to next stable states at the memory section after change of an input X_k .

However the unstable states, which lead to other unstable states, are not shown in flow table and at a table next state entry only the eventually occurring states are shown. Flow table gives the condensed view of the circuit.

Number of columns and rows of the flow table are the same as state table except that a flow table row(s) does no longer exist if at least one of the next states given at the row is the finally stable one (encircled or boxed one) for any set of inputs at the state able. In other words, each row will show at least one encircled or boxed state.

Entries in columns and rows of the flow table are the same as corresponding rows (now these can be less) at the state table except the following:

1. A flow table row(s) does not show an entry of the unstable state (recognized by no encircling over it in state table) that is going to lead to another unstable state and a stable state entry name (without encircling) replaces it and that is the state that will eventually occur in the next feedback-cycle(s).

In other words, each row will give in the columns the next stable states, that are going to occur and will change each state-table unstable state entry if that leads to another state which is stable (Table 9.4).

2. A flow table row(s) show a next state entry by dash if there is another stable state in the row and the transition to this entry from that will need more number of simultaneous input changes than permitted to occur. In other words, it does not show two identical entries for the stable state if both have the inputs differing at two or more bit places when only one input bit change is the only change permitted in a circuit as per constraint specified. For example, for there are encircled C and C' at two input conditions $(0, 0)$ and $(0, 1)$. Then none of them is replaced by the dash and encircled C and C' entries are at two input conditions $(0, 0)$ and $(1, 1)$, then one of them is replaced by the dash. (Refer explanation given later in a foot note of Table 9.6.)
3. A flow table row(s) output entry is shown only for the outputs, which lead to stable next states in the state table, else an output is shown by the dash.
4. Number of rows in each column equals 2^n for a memory section of n state-variables *minus* the number of rows having no input condition with an encircled state *minus* the state that are equivalent if the reduction is performed for minimization.

Table 9.4 gives the flow table constructed from excitation-cum-transition table in Table 9.1. Comparison of the table shows that the output that preceded the unstable state is shown by the dashes. Also only the finally achieved state is shown and intermediate states are not shown. [Note that row for S_1 has been removed as Y for $X = 1$ is $-$.]

9.5.6 Example of an Excitation-cum-Transition Table

Consider another excitation-cum-transition table, Table 9.5. Assume that it is for a fundamental mode operating circuit-6. Figure 9.5(a) shows the state diagram.

9.5.7 Flow Table from Excitation-Transition Table

Table 9.6 gives the flow table constructed from the Table 9.5 transition table, that which showed the directed arcs to give the transitions. Flow table therefore represents the stable behavior of a fundamental mode circuit in contrast to state table.

TABLE 9.4 Flow table for an exemplary asynchronous sequential circuit operating in fundamental mode

Present state (x_{q0}, x_{q1})	Next state after transition (x'_{q0}, x'_{q1})		Present output Y_0		
	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$	
0, 0	S_0	S_0	S_3	0	—
1, 0	S_2	S_1	S_3	0	1
1, 1	S_3	S_1	S_0	1	1

9.14 Digital Systems: Principles and Design

TABLE 9.5 Excitation-cum-transition table for an exemplary circuit-6 asynchronous sequential circuit operating in fundamental mode

Present state (x_{q0}, x_{q1})	Next state after transition (x'_{q0}, x'_{q1})*				Present output Y_0			
	Input $X_1X_0 = 00$	Input $X_1X_0 = 01$	Input $X_1X_0 = 10$	Input $X_1X_0 = 11$	Input $X_1X_0 = 00$	Input $X_1X_0 = 01$	Input $X_1X_0 = 10$	Input $X_1X_0 = 11$
0, 0	0, 1	0, 0	1, 0	1, 0	0	0	1	1
0, 1	0, 1	0, 0	1, 0	1, 0	1	0	1	0
1, 0	1, 1	1, 1	1, 0	1, 0	0	1	0	0
1, 1	1, 1	0, 1	1, 1	1, 0	1	1	0	1

* Stable state is marked by circle or box over the state variables. Horizontal directed arc(s) with arrow shows the action at the combinational circuit. Vertical directed arc(s) with arrow shows the action at the memory section by latches. Number of vertical arcs equal the number of feedback-cycles after which stability is achieved.

TABLE 9.6 Flow table for an exemplary circuit-6, asynchronous-sequential circuit operating in fundamental mode

Present state (x_{q0}, x_{q1})	Next state after transition (x'_{q0}, x'_{q1})				Present output Y_0			
	Input $X_1X_0 = 00$	Input $X_1X_0 = 01$	Input $X_1X_0 = 10$	Input $X_1X_0 = 11$	Input $X_1X_0 = 00$	Input $X_1X_0 = 01$	Input $X_1X_0 = 10$	Input $X_1X_0 = 11$
S_0	S_1	S_0	–	S_2	0	0	–	1
S_1	S_1	S_0	S_2	S_1^{\wedge}	1	–	1	0
S_2	S_3	S_1^{\wedge}	S_2	S_1^{\wedge}	0	–	0	1
S_3	S_3	S_1	S_3	S_2	1	1	0	1

Note the following:

- * Stable state is marked by a circle or box over the state.
- S_1^{\wedge} replaces the S_2 in Table 9.5 at third row column 5, because S_1 is the final stable state after two arcs to S_1 . Similarly S_1 replaced by S_3 in row 3 column 3 because S_1 is the final stable state for the given state of inputs and outputs after one arc to S_1 .
- Dash signs in columns between columns 6 and 9 rows 2 and 4 shows that that output with the given X_1X_0 lead to unstable intermediate states.
- A dash sign in row 1 column 4 shows that S_2 will be constrained to non-existence because two input changes are not allowed simultaneously. In row 1 S_0 is the present state. When input changes to 01, then also the state is S_0 . Now at the Table 9.6, the states are S_2 for both column 4 and column 5 row 1. Both are S_2 . Only 01 to 11 input changes are permitted due to input constraints. Hence a dash sign replaces S_2 in column-4 row-1.
- Also the column 8 row output also shows a dash because the corresponding state is shown as non-existent.

Flow table in Table 9.6 represents the condensed view of the flow from one set of states to another set of stable states in different input X conditions.

9.5.8 Flow Diagram

Figure 9.4(b) shows a flow diagram constructed from circuit-5 of excitation-cum-transition table in Table 9.1. A flow diagram is diagrammatic representation of the

flow table. States S_0 , S_1 , S_2 and S_3 are then labeled at the centers of each circle representing a node.

1. Each arc or circular arc is labeled with present input and the present output at the transition. If there is dash, then slash sign and outputs for that are not shown.
2. Each arc or circular arc can have more than one set of (pre-transition input/post transition output) labeled on it if there are more than one sets of (pre-transition input/post transition output) that are having the same transition from a node to another.
3. The number of nodes = number of rows in the flow table.
4. A directed arc from the present state node to the next state node shows a transition to the stable state after one or more feedback-cycles through the memory or delay section.
5. A small diameter circular directed arc marks a transition in which the flow does not appear to occur because the state remains unchanged after one or more feedback-cycles.
6. The number of directed arcs equal the number of transitions in which the state changes and the flow occurs to next stable state.
7. The number of directed circular smaller diameter arcs equals the number of transitions in which the flow does not occur.

Figure 9.5(b) shows an example of flow diagram constructed from circuit-6 flow table in Table 9.6 using the above guidelines (Figure 9.5(a) shows state diagram).

Note that a state diagram in Figure 9.5(a) for circuit-6 shows two directed arc for S_0 to S_2 , one is for 10/1 and other is for 11/1, present inputs/output. Flow diagram drawn from it removes one arc 10/1 because two simultaneously bit changes are assumed to be non-existent (Figure 9.5(b)).

A comparison of Figures 9.5(b) and 9.5(a) (flow diagram and state diagrams) shows the followings:

1. The directed arc for the case of two simultaneous input changes at figure left most side at the middle from S_0 to S_2 has been removed.
2. A directed arc from S_2 to S_1 for the cases of the inputs 01 and 11 is shown in place of unstable state transition through S_3 for 01/1 in the state diagram.

9.6 RACES

Example Circuit 5

Closely examine the Table 9.1 and look for the entries of (x'_{q0}, x'_{q1}) which differ for both the bits with respect top present state (x_{q0}, x_{q1}) places. Examination reveals the followings:

1. (x_{q0}, x_{q1}) and (x'_{q0}, x'_{q1}) [the present and past states] are (0, 1) and (1, 0), respectively. (Refer column 3 row 2).
2. (x_{q0}, x_{q1}) and (x'_{q0}, x'_{q1}) [the present and past states] are (1, 1) and (0, 0), respectively (Refer column 3 row 4).

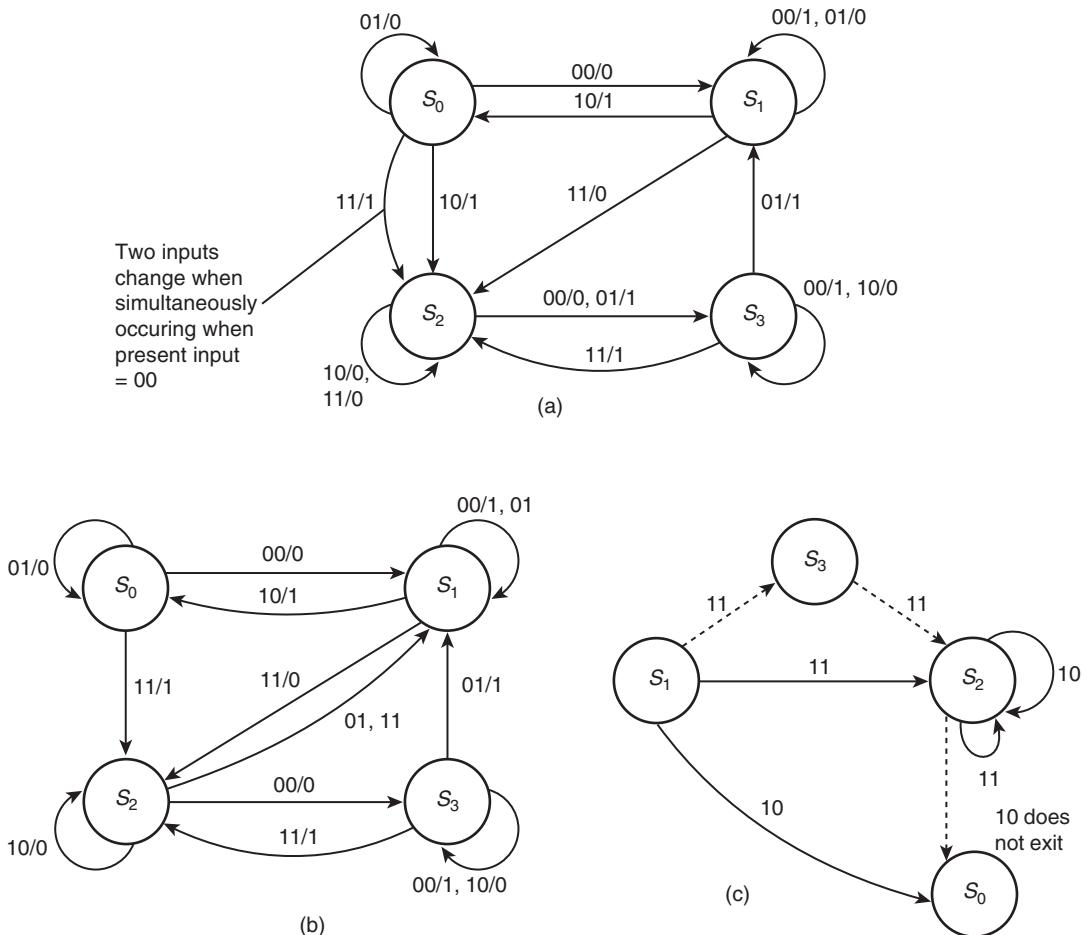


FIGURE 9.5 (a) State diagram of circuit-6 having state table as per Table 9.5. (b) Flow diagram of circuit-6 having state table as per circuit-6 Table 9.5. (c) The race arising due to the feedback of next state variables, which don't occur simultaneously when two bits change at the delay section during change when a state changes.

There are two races at two entries of next state outputs for the present states in the transitions given by Table 9.1. Figure 9.4(c) showed two races due to the feedback next state variables, which don't change simultaneously when two bits change at the delay section during change when a state changes. The following explains this:

There are two-bit changes at the memory or delay sections. There is no guarantee, which will change x_{q0} first or x_{q1} first to x'_{q0} or x'_{q1} , respectively as two circuits propagation-delays are never exactly identical. Therefore, (0, 1) present state can change to (1, 1) for some period (S_1 can temporarily be S_3 in place of S_2). Alternatively, (0, 1) can temporarily change to (0, 0). Therefore, (0, 1) present state can change to (0, 0) for some period (S_1 can temporarily be S_0 in place of S_2). This is called race condition.

Also, (1, 1) present state can change to (1, 0) for some period (S_3 can temporarily be S_2 in place of S_0). Alternatively, (1, 1) can temporarily change to (0, 1). Therefore, (1, 1) present state can change to (0, 1) for some period (S_3 change temporarily be S_1 in place of S_0). This is called race condition.

Example Circuit 6

Closely examine the circuit-6 Table 9.5 and look for the entries of (x'_{q0}, x'_{q1}) in columns 2 to 4 which differ at both the places of the bits with respect the present state (x_{q0}, x_{q1}) in column 1. Examination reveals the followings:

1. (x_{q0}, x_{q1}) and (x'_{q0}, x'_{q1}) the present and next states are (0, 1) and (1, 0), respectively (Refer columns 1 and 5 row 2).

There are two-bit changes at the memory or delay sections. There is no guarantee, which will change x_{q0} first or x_{q1} first to x'_{q0} , or x'_{q1} , respectively as two circuits propagation-delays are never exactly identical. Therefore, (0, 1) present state can change to (1,1) for some period (S_1 can temporarily be S_3 in place of S_2). Alternatively, (0, 1) can temporarily change to (0, 0). Therefore, (0, 1) present state remains unchanged (0, 1) for some period.

This is race condition in first case only. There is only one race for the present state transitions given by Table 9.5. S_1 raced to S_3 in place of S_2 due to $(0, 1) \rightarrow (1, 1)$. Figure 9.5(c) shows the race due to the feedback of two next state variables, which don't occur simultaneously when two bits change at the delay section during change when a state changes.

Point to Remember

A race is said to occur if between next set of states \underline{x}_q and present states \underline{x}'_q , there are two or more bits change at a memory or delay section.

9.6.1 Cycles of the Races

There are two possibilities when a temporary change to an intermediate state occurs due to differing propagation delays for two bits in a feedback cycle.

Non-Critical Race (Deterministic Number of Cycles of race)

The state momentarily changed can lead to a stable state with one or more additional feedback-cycles and the state eventually turns out to be the same as would have been expected had there been no race. Race condition is not permanent and the race returns to normal after calculatedly number of races. Asynchronous circuit is deterministic when a non-critical race condition exists.

For example, examine Table 9.1 and note the followings:

1. The present state (0, 1) if momentarily changes to (0, 0); the momentarily state will be S_0 . The S_0 for $X=1$ gives the output (1, 0) again and gives the state S_2 in next feedback cycle.
2. The present state (0, 1) if momentarily changes to (1, 1), the momentarily state will be S_3 . The S_3 for $X=1$ gives the output (0, 0) and state S_0 again in next feedback-cycle. S_0 gives the state S_2 in next feedback cycle.

Therefore after two state transitions, the same stable state is obtained in case (2) and after one more state transitions in case (1) as would have been had there been no race to case (1) or (2) above. The race is therefore not critical in its effect on the behavior, except additional delay. Flow table shall remain unaltered.

Critical Race (Indeterminate Number of Cycles of Races)

The state momentarily changed can lead to a different stable state with any known number one or more additional feedback-cycles and the state will most often turn out never to be the same as would have been expected had there been no race. Such a race is called critical race. Race condition is indefinite and the race returns to normal after indefinite number of races because when two propagation delays exactly match is never known. Asynchronous circuit is probabilistic, not deterministic when a critical race condition exists.

For example, examine Table 9.1 row-4 column-3 and note the followings:

3. The present state (1, 1) if momentarily changes to (0, 1); the monetarily state will be S_1 . The S_1 for $X = 1$ gives the stable state S_3 back after two feedback cycles.
4. The present state (1, 1) if momentarily changes to (1, 0), the monetarily state will be S_2 . The S_2 for $X = 1$ gives the state (1, 1) S_3 again in next feedback cycle.

Therefore, after two or one state transition, the starting stable state S_3 is obtained in cases (1) and case (2), respectively. Therefore, even after any number of transitions S_0 [(0, 0) state] will never occur unless both delay circuits are exactly identical, which may need an infinite wait. Flow table state (S_0) will rarely be obtained from S_3 when input changes from 0 to 1. Such a race is called *critical race*. Consider Table 9.5 circuit 6 case Figure 9.5(c) shows that $(1, 0) \rightarrow (1, 1)$ is critical race as there is no path to return to S_0 for inputs 10.

9.7 RACE-FREE ASSIGNMENTS

Memory (or delay) section next-state variables when have the bits differing by more than one during a transition, the critical (indeterminate time for stable state) and non-critical (deterministic) races occur. Critical race must not occur in an asynchronous sequential circuit.

Race free assignments is an assignment of state variables such that next-state variables when don't have the bits differing by one during a transition. Recall the bit changes in adjacent cells in a Karnaugh map. Only a single bit changes between two adjacent cells.

Step 1: Finding Non-Adjacencies in Transitions for Different Input Conditions

Let us build a Karnaugh map type table for each set of X in a flow table in Table 9.4. Tables 9.7 part (a) and (b) are adjacency map for the assignments of present and next state variables for $X = 0$ and $X = 1$, respectively. It shows in a cell the present state by 0 and next state by 1. Dotted lines in table show the non adjacency.

TABLE 9.7 Adjacency map for the assignments of present and next state variables

	$\leftarrow X = 0 \rightarrow$				$\leftarrow X = 1 \rightarrow$			
(x'_{q0}, x'_{q1}) (x_{q0}, x_{q1})	00 S_0	01 S_1	11 S_3	10 S_2	00 S_0	01 S_1	11 S_3	10 S_2
00 S_0	0	-			0		1	
01 S_1	0					-	-	
11 S_3		1	0		1	-	-0	
10 S_2		1	-	-0			1	0

$\leftarrow (a) \rightarrow$ $\leftarrow (b) \rightarrow$

1. It is observed that for the present state S_3 , the transition to S_0 is not to the adjacent cell. S_0 and S_3 have to be the neighbors. [Table 9.7(b)]
2. It is observed that for the present state S_2 , the transition to S_1 is not to the adjacent cell. S_2 and S_1 have to be the neighbors. [Table 9.7(b)]

Using the map number of races (non-critical plus critical) is thus found to be two only for asynchronous sequential circuit when the flow table is as per Table 9.4.

Let us build a Karnaugh map type table for each set of $X_0 X_1$ in flow table in Table 9.6. Tables 9.8 parts (a) to (d) show the adjacencies (including wrapping adjacencies) and non-adjacencies (by dotted lines) for $X=00, 01, 11$ and 10 , respectively. It shows in a cell the present state by 0 and next state by 1. The adjacencies of 0 and 1 are looked for. (Cells at left and right borders are adjacent; these have wrapping adjacency).

1. It is observed that for the present state S_2 , the transition to S_1 is not to the adjacent cell. S_2 and S_1 have to be the neighbors. [Row 4 Table 9.8(b)]
2. It is observed that for the present state S_1 , the transition to S_2 is not to the adjacent cell. S_1 and S_2 have to be the neighbors. [Row 2 Table 9.8(c)]
3. It is observed that for the present state S_2 , the transition to S_1 is not to the adjacent cell. S_2 and S_1 have to be the neighbors. [Row 4 Table 9.7(c)]

All three observations above point out that the S_1 and S_2 state have to be neighbors.

Number of races (non-critical plus critical) is thus found to be three for asynchronous sequential circuit with flow table as per Table 9.6.

Step 2: Finding Race free State Assignments for Different Input Conditions

Method 1: Replacement of an unstable non-adjacent state by another adjacent state in flow table so that in the next cycle the stable state is obtained.

Consider circuit example (Table 9.4). The S_3 in row 1 of Table 9.4 can be replaced by S_2 , the S_2 will lead to S_3 in next cycle. [Modification of flow table is permitted if it does not change the result finally achieved.]

S_0 present state is unresponsive to any input condition. [$S_0 \rightarrow S_0$ for $X=0$ and for $X=1$, the present Y is $-$] S_0 S_0 can be assigned three state variables, $(x_{q0}, x_{q1}, x_{q2}) = 101$ then the race free condition can be achieved

Consider circuit-6 example in Table 9.6.

TABLE 9.8 Adjacency map for the assignments of present and next state variables

		$X_0X_1 = 00$				$X_0X_1 = 01$			
(x'_{q0}, x'_{q1})	(x_{q0}, x_{q1})	00 S_0	01 S_1	11 S_3	10 S_2	00 S_0	01 S_1	11 S_3	10 S_2
00 S_0		0	1			01			
01 S_1			01				01		
11 S_3				01			1	0	
10 S_2				1	0		1—	—	0

		(a)				(b)			
		$X_0X_1 = 11$				$X_0X_1 = 10$			
(x'_{q0}, x'_{q1})	(x_{q0}, x_{q1})	00 S_0	01 S_1	11 S_3	10 S_2	00 S_0	01 S_1	11 S_3	10 S_2
00 S_0		0 ⁺	—		1 ⁺	0			—
01 S_1			0—	—	—1	1	0		
11 S_3				01				0	1
10 S_2			1—	—	—0				1

		(c)				(d)			
		$X_0X_1 = 11$				$X_0X_1 = 10$			
(x'_{q0}, x'_{q1})	(x_{q0}, x_{q1})	00 S_0	01 S_1	11 S_3	10 S_2	00 S_0	01 S_1	11 S_3	10 S_2
00 S_0		0 ⁺	—		1 ⁺	0			—
01 S_1			0—	—	—1	1	0		
11 S_3				01				0	1
10 S_2			1—	—	—0				1

⁺ Wrapping adjacency

S_1 and S_2 can be made neighbors by using three variable assignments. S_1 can be assigned three state variables, $(x_{q0}, x_{q1}, x_{q2}) = 101$. S_0, S_2 and S_3 are assigned 000, 100 and 110, respectively. [Now $S_1 \rightarrow S_2$ cause 101 \rightarrow 100, which means only one variable change.]

Method 2: There is a systematic method, called one-hot method of race free state assignments.

1. *Action 1:* Let number of rows in flow table = m . Use m state variables. Assign each row $(x_{q0}, x_{q1}, \dots, x_{qn-1})$ in sequence as 00...01, 00...10, ..., 10...00. In k -th row, the k -th state variable is 1. For example, if there are four rows in the flow table, S_0, S_1, S_2 and S_3 , assign 1000, 0100, 0010 and 0001, respectively to $(x_{q0}, x_{q1}, \dots, x_{qn-1})$ with $m = 4$.
2. *Action 2:* Present-state column 1 contains the states as per the state variable assignments given above.
3. *Action 3:* Assign the stable state variables to the state variable as per corresponding state variables used in present-state column and leave presently unstable state as such. For example for the Table 9.6, assign present state and state variables as shown in Table 9.9.
4. *Action 4:* For unstable state, now write the assignment after ORing with the assignment for its next cycle transition. For example, (i) now consider row 2 column 4 assignment for input 10. Stable case S_0 assignment would have been 1000. The state vertical next cycle transition would be to 0010 row-3 column-4. ORing these two gives 1010. Enter this value in the for row 2 column 4.

TABLE 9.9 Assignments for the present state and stable states after actions 1, 2 and 3 on the flow table in Table 9.6

Present state (x_{q0}, x_{q1})	Next state after transition (x'_{q0}, x'_{q1})				Present output Y_0			
	Input $X_1X_0 = 00$	Input $X_1X_0 = 01$	Input $X_1X_0 = 10$	Input $X_1X_0 = 11$	Input $X_1X_0 = 00$	Input $X_1X_0 = 01$	Input $X_1X_0 = 10$	Input $X_1X_0 = 11$
$S_0 1000$	S_1	1000	-	S_2	0	0	-	1
$S_1 0100$	0100	S_1	S_0	S_2	1	-	1	0
$S_2 0010$	S_3	S_1	0010	0100	0	-	0	1
$S_3 0001$	0001	S_1	0001	S_2	1	1	0	1

(ii) Consider row 1 column 2 unstable S_1 next cycle transition is S_1 for input 01. For S_1 stable state assignment is 0100 and S_1 stable state it would have been 0100. Assignment after ORing the two will again be 0100. Therefore, enter 0100 assignment in column 2 of row1. (iii) Now consider row 3 column 2 assignment for input 00. Stable case S_3 assignment would have been 0001. The state vertical next cycle transition would be 0001 row-4 column-2. Its assignment is also 0001. ORing these two gives 0001 again. Enter this value in the for row 3 column 2. Complete the assignments for all unstable states by this ORing procedure. Now Table 9.9 will get two type of next-state assignments: assignments with only one variable = 1 and assignment with two variables = 1.

5. Action 5: Now add an extra row for each of those next states, which have two variables as 1s. The assignment for the same vertical column of the state with two variable 1s can now be done so that next state transition occurs to the same when the next state occurs, which was ORed with the unstable state assignment before. Rewriting Table 9.9 as Table 9.10 explains this.

TABLE 9.10 Assignments in one-hot method (method-2) of race free assignment after actions 1 to 5 (Table 9.9 rewritten after incorporating actions 4 and 5 of the method)

Present state (x_{q0}, x_{q1})	Next state after transition (x'_{q0}, x'_{q1})				Present output Y_0			
	Input $X_1X_0 = 00$	Input $X_1X_0 = 01$	Input $X_1X_0 = 10$	Input $X_1X_0 = 11$	Input $X_1X_0 = 00$	Input $X_1X_0 = 01$	Input $X_1X_0 = 10$	Input $X_1X_0 = 11$
$S_0 1000$	0100	1000	-	0110 ^b	0	0	-	1
$S_1 0100$	0100	0100	1010 ^a	0110 ^c	1	-	1	0
$S_2 0010$	0001	0100	0010	0100	0	-	0	1
$S_3 0001$	0001	0100	0001	0110 ^b	1	1	0	1
$S_4 1010^b$	-	-	0010	-				
$S_5 0110$	-	-	-	0100				

^a New assignment to unstable state after ORing with its next cycle state assignment in column 1 for present states.

^b New row addition for an unstable state assignment in row-2 column-4. Arrow shows the next cycle state 0010. It is same as expected from Table 9.6. ^c New assignment to the unstable state after ORing with its next cycle state assignment in column 5 for present states.

6. *Action 6:* Change the output column entry for the state assigned two variables as 1s. Put the next cycle stable state. This is to prevent two times changes in an output when adding another extra row in the flow table. Rewriting the Table 9.10, Table 9.11 gives final race free assignment table after the actions 1 to 6 of one-hot method. From an unstable state, there is a path now to hot (stable) state always feasible.

It can be noted from Table 9.11 that indeterminate race condition will not be nonexistent.

TABLE 9.11 Assignments in one-hot method (method-2) of race free assignment after actions 1 to 6 (Table 9.10 rewritten after incorporating action 6 for output entries change for the unstable states)

Present state (x_{q0}, x_{q1})	Next state after transition (x'_{q0}, x'_{q1})					Present output Y_0			
	Input $X_1X_0 = 00$	Input $X_1X_0 = 01$	Input $X_1X_0 = 10$	Input $X_1X_0 = 11$		Input $X_1X_0 = 00$	Input $X_1X_0 = 01$	Input $X_1X_0 = 10$	Input $X_1X_0 = 11$
$S_0 \ 1000$	0100	1000	-	0110		0	0	-	1
$S_1 \ 0100$	0100	0100	1010 ^a	0110		1	-	0 ^b	1 ^b
$S_2 \ 0010$	0001	0100	0010	0101	0010	0	-	0	1
$S_3 \ 0001$	0001	0100	0001	0110	0001	1	1	0	1
$S_4 \ 1010$	-	-	0010	-	0010				
$S_5 \ 0110$	-	-	-	-	0100				

^a Now there is the next cycle after which there becomes stable state output entry [0010] row 3, column 4. ^b New output entry (entered from the next row of same column).

■ EXAMPLES

Example 9.1

Make the excitation table for the circuit-5 of Figure 9.3.

Solution

There is one external input and two next state inputs after a feedback cycle. Therefore, we take two variables (x_{q1}, x_{q2}) and four rows $(x_{q1}, x_{q2}) = 00, 01, 10$ and 11 for the excitation table. Since there is one external input, there will be two columns in the entries for the excitation variables and two columns in the output.

There are three excitation variables in Equation (9.1) for the circuit of Figure 9.3(a). Using Equation (9.1), the entries of the table are now filled up. Table 9.12 gives the excitation table. Let starting values of Q_1 and Q_2 are same as X and \bar{X} , respectively.

TABLE 9.12 Excitation table for asynchronous sequential circuit-5 operating in fundamental mode circuit

Present state (\bar{Q}_1, \bar{Q}_2)	Excitation variables before the transition (y_{q0}, y_{q1}, y_{q2})		Present output Y_0	
	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$
(x_{q1}, x_{q2})				
0, 0	0, 0, 0	1, 0, 1	0	0
0, 1	0, 0, 0	1, 0, 1	1	0
1, 0	0, 1, 0	0, 1, 1	1	1
1, 1	0, 1, 0	0, 1, 1	1	1

Example 9.2 Make the transition table from the circuit of Figure 9.3(b) using the excitation table made in Example 9.1.

Solution

Table 9.13 gives the transition table obtained after using Equation (9.2). The equations give the next states (x'_{q1}, x'_{q2}) after a single feedback cycle. Excitation table first variable will be complemented as there is a D latch between y_{q0} and \bar{x}'_{q1} . Excitation table y_{q1} and Y_0 will give the next state as expected from a JK latch \bar{Q} . When these are 10, $x_{q1} = 1$; when these are 01, $x_{q1} = 0$; When these are 00, the result is same as present state. Assume that for 11 also, there is complementation like a JK . Mark by circle or ellipse the transitions, which will be changing, in the next cycle. We find that none of the state is stable. This is because use of JK latch, which toggles.

TABLE 9.13 Transition table for asynchronous sequential circuit-5 operating in fundamental mode

Present state	State variables before the transition (x'_{q1}, x'_{q2})		Present output Y_0	
	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$
(x_{q1}, x_{q2})				
0, 0	1, 0	0, 0	0	0
0, 1	1, 0	0, 0	1	0
1, 0	1, 1	0, 1	1	1
1, 1	1, 1	0, 0	1	1

Note: All states are unstable. This is expected when using a JK latch.

Example 9.3 Make the state table for the circuit-5 (Figure 9.3(b)) using the transition table made in Example 9.2.

Solution

Make Table 9.14 after assigning the states S_0, S_1, S_2 and S_3 to $(x_{q0}, x_{q1}) = 00, 01, 10$ and 11, respectively, in Table 9.13.

Example 9.4 Why is the flow table is not possible for a circuit with state table as Table 9.14?

Solution

This is because all entries are unstable due to $JK FF$. In flow table, at least one stable state must be there in a row. State table shows that Y is changing in both the cases.

TABLE 9.14 State Table for Figure 9.3(b) asynchronous sequential circuit operating in fundamental mode

Present state (x_{q1}, x_{q2})	Excitation variables before the transition (y_{q0}, y_{q1}, y_{q2})		Present output Y_0	
	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$
0, 0 S_0	S_2	S_0	0	0
0, 1 S_1	S_2	S_0	1	0
1, 0 S_2	S_3	S_1	1	1
1, 1 S_3	S_3	S_0	1	1

When X input is 1, $S_2 \rightarrow S_1$, $S_1 \rightarrow S_0$, S_2 is unstable. Similarly, when $X = 0$, $S_0 \rightarrow S_2$, S_3 . The S_0 is unstable when $X = 0$.

Example 9.5 Draw state diagram from Table 9.14.

Solution

Figure 9.6 shows a state diagram from Table 9.14. State diagram shows that when X undergoes transition from 0 to 1, the S_0 may show intermediate state S_2 and S_3 also.

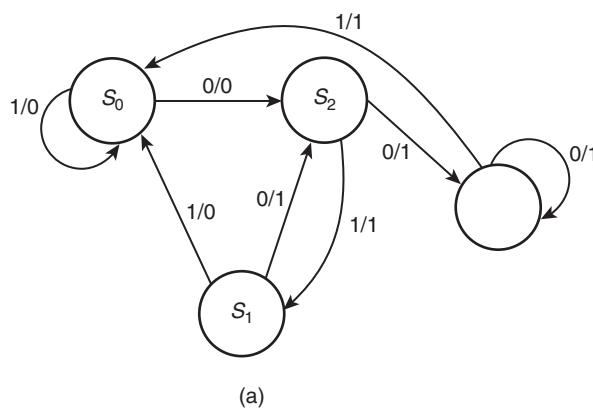


FIGURE 9.6 State diagram for the circuit-5 using Table 9.14.

■ EXERCISES

1. Make the excitation table for the circuit-7 of Figure 9.7.
2. Make the transition table from the circuit-7 of Figure 9.7 using the excitation table made in Exercise 1.

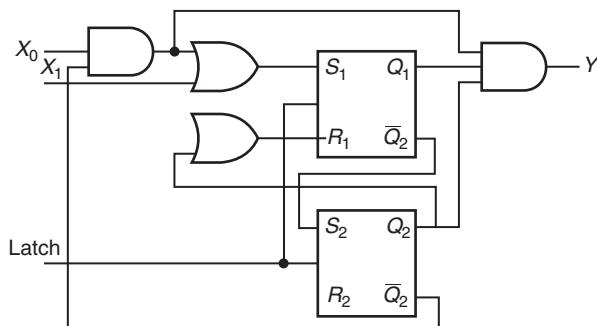


FIGURE 9.7 An asynchronous circuit-7 for Exercise 10.1.

3. Make the state table from the circuit-7 of Figure 9.7 using the transition table made in Exercise 2.
4. Make the flow table from the circuit-7 of Figure 9.7 using the transition table made in Exercise 3.
5. Make a flow diagram from flow table in Exercise 4.
6. Make the transition table from the excitation table made in Table 9.15 of a circuit-8. Delay section has just the delay elements.

TABLE 9.15 Excitation table for an exemplary asynchronous sequential circuit-7 operating in fundamental mode

Present state (x_{q0}, x_{q1})	Excitation variables before the transition (y_{q0}, y_{q1})		Present output Y_0	
	Input $X = 0$	Input $X = 1$	Input $X = 0$	Input $X = 1$
0, 0	1, 0	0, 0	0	0
0, 1	1, 0	0, 0	1	0
1, 0	1, 1	0, 1	0	1
1, 1	1, 1	0, 0	1	1

7. Make the transition table from the circuit-8 using the excitation table made in Exercise 6.
8. Make the state table from the circuit-8 using the transition table made in Exercise 7.
9. Make the flow table and flow diagram from the circuit-8 using the state table made in Exercise 8.
10. Find the races in a circuit whose flow table is as per Table 9.16. Examine and tell which of the race is critical.
11. Perform race free assignments for a flow table in Table 9.16 using the systematic method

TABLE 9.16 Flow table of an exemplary circuit-8; asynchronous sequential circuit operating in fundamental mode

Present state (x_{q0}, x_{q1})	Next state after transition (x'_{q0}, x'_{q1})					Present output Y_0		
	Input $X_1X_0 = 00$	Input $X_1X_0 = 01$	Input $X_1X_0 = 10$	Input $X_1X_0 = 11$	Input $X_1X_0 = 00$	Input $X_1X_0 = 01$	Input $X_1X_0 = 10$	Input $X_1X_0 = 11$
S_0	S_1	S_0	S_3	S_2	0	0	-	1
S_1	S_1	S_1	S_0	S_2	1	-	1	0
S_2	S_1	S_3	S_2	S_3	0	-	0	1
S_3	S_0	S_1	S_3	S_3	1	1	0	1

■ QUESTIONS

- When will you call a circuit a asynchronous sequential circuit?
- Describe what do you mean by operating in fundamental mode in a asynchronous sequential circuit.
- Why is the assumption that one input bit changes at an instance is a practical one most of the times, explain?
- Why do you get unstable states in a asynchronous circuit?
- When do you call a state as stable?
- List the steps used for analyzing a given asynchronous sequential circuit.
- Describe procedure to get state table from excitation table in an asynchronous sequential circuit. How does it differ from synchronous sequential circuit? (Hint: By encircling the stable states and by identifying the unstable states).
- How do you get output specifications from a flow table in asynchronous sequential circuit operating in fundamental mode?
- When do you get the non-critical races?
- When do you get the critical races?
- How do you perform race free assignments by adding extra rows in the flow table?
- Describe a procedure to identify the races in asynchronous sequential circuit.

CHAPTER 10

Hazards and Pulse Mode Sequential Circuits

OBJECTIVE

We will learn in this chapter the hazards that are present in a logic circuit when two inputs change due to a single logic variable. We will also learn about pulse mode sequential circuits. Hazard means a source of trouble or risk.

We learnt in Chapter 9 the following important points:

1. A general asynchronous sequential circuit is a circuit in which there are no control(s) for the instance(s) at which the output(s) change like a control by a clocked instance in synchronous sequential circuit.
2. The circuit has a section consisting of the (i) combinational circuits at the input and output stages and (ii) latches (without clock inputs) or an equivalent delay devices, which have the circuit outputs as the excitation inputs and after the transitions and delays give the new input states.
3. A mode called fundamental mode is a mode of operation in which external input changes only when the circuit is in stable state. Stable state means further transitions don't create any new state.
4. Asynchronous sequential circuit is analyzed in fundamental mode by first making the tables for the excitations, transitions, states and then finally the flow tables and flow diagrams.
5. Asynchronous sequential circuit becomes unstable when the next state(s) changes repeatedly without settling to a stable state in a finite time.

6. There can be a race condition. It occurs when two or more states, which are the inputs after the delay, changes the states in a cycle. For example, x_q and x'_q inputs, which are the outputs of the latches or delay elements after a transition cycle, change to 10 from 01 or change from 10 to 11 and then to 01 due to the different amount of delays for obtaining x_q and x'_q in any two transitions. Note: It is natural that two circuits can rarely be having exactly identical propagation delays. Even temperature in vicinity of identically made gate can cause the difference.
7. Race condition can sometimes be critical. That means the transition cycle leads to an unstable state from which the finally settable state(s) is not recovered in the succeeding cycles. Finally settable state(s) is one that would have been had the two delays are identical when there are two transitions like of the change to 01 from 10 or change is from 10 or to 01 or 11 from 00 or 00 from 11.
8. Race free assignments are possible and that results in the asynchronous circuit free from critical race condition.

10.1 HAZARDS

Consider a NAND gate (of propagation average delay = Δt) in which one of the inputs gets input from a logic variable X . Other input of NAND gets input from X due to a NOT (of propagation average delay = $\Delta t'$) between X and that input. The NAND characteristics suggest that NAND with X and X inputs will always give output = 1. In actual practice, the output of NAND will show a transient from 1 to 0, then again to 1 during the period = $\Delta t'$. This is because in the Boolean operation $Y = X \cdot \bar{X}$, when X changes, the \bar{X} changes only after a period = $\Delta t'$. Figures 10.1(a) and (b) show the circuit with two inputs dependent on one variable and the timing diagrams for X , \bar{X} , and Y in this operation. The NOT is a delaying element in this circuit. Figures 10.1(c) and (d) show the use of a delay element, which is some Boolean expression implementing circuit to give the output as a source of a hazard and show the timing diagram, respectively.

Suppose that NOT implementing NAND in circuit of Figure 10.1(a) has much shorter delay compared to output stage NAND. Now, there is no hazard when the response time of the sequential or combinational next stage happens to be longer than that.

From the timing diagrams in Figure 10.1(b) and (c) it appears that the output effects only momentarily for $\Delta t'$ or $\Delta t'''$ only. Suppose this output Y is input to a counter reset input. Then counter will lose all previous counts. In that case, the hazard creates a malfunction, which is not momentarily. Suppose Y is input to a clocked sequential circuit. Then the hazard creates no malfunction because input at the clocking instance only is important.

Point to Remember

A hazard arises and becomes a potential source of momentary or permanent malfunctioning in a combinational circuit and asynchronous sequential circuit

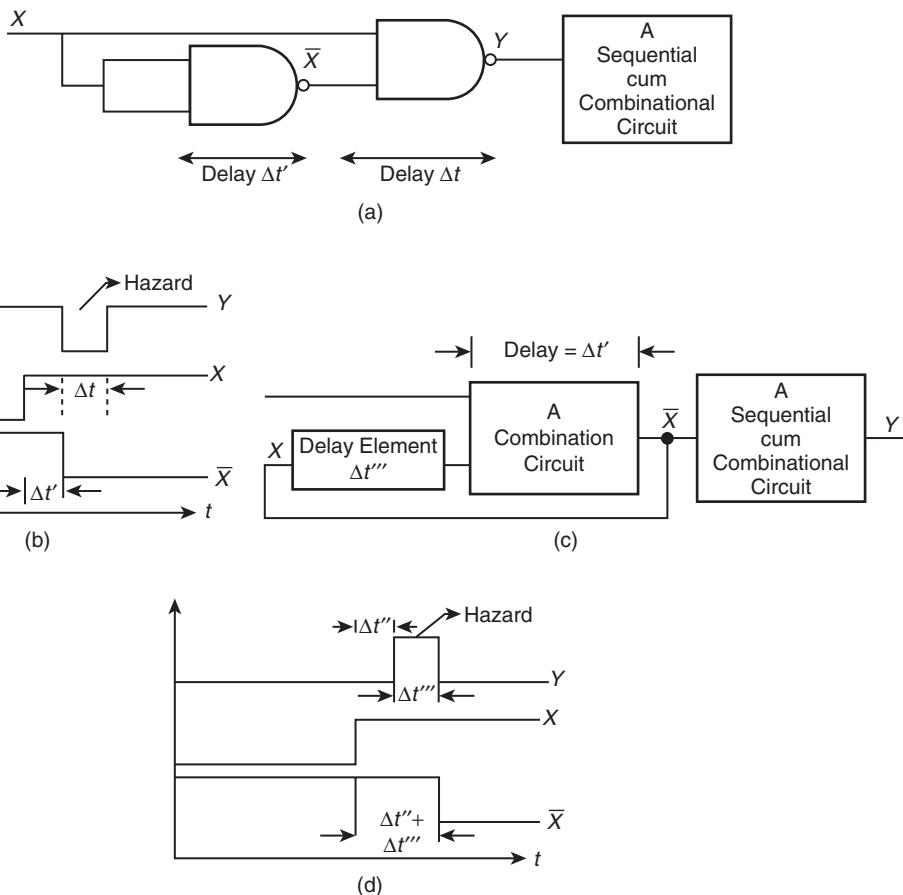


FIGURE 10.1 (a) Two NANDs circuit with two inputs of NAND dependent on one variable (b) Timing diagram for X, \bar{X} , and Y in this operation. (c) Output, which creates a static-1 hazard.

when a variable(s) and its complement(s) are used with complement(s) having a delay element(s) before it, so that for a certain period the complement is not available, leading to a different output either momentarily for the delay period or permanently.

10.1.1 Static-0 Hazard

Figure 10.2(a) shows a general circuit network consisting of logic circuit-1 and circuit-2. Static-0 hazard is a hazard in which a Boolean algebraic output or next state output expected is 0, but momentarily, it undergoes transition to 1. For example, the circuit of Figure 10.1(c) expected $Y = 0$, but it becomes 1 for a period of propagation delay $\Delta t'''$ of the complementing circuit. Figure 10.2(b) shows the output, which creates a static-0 hazard.

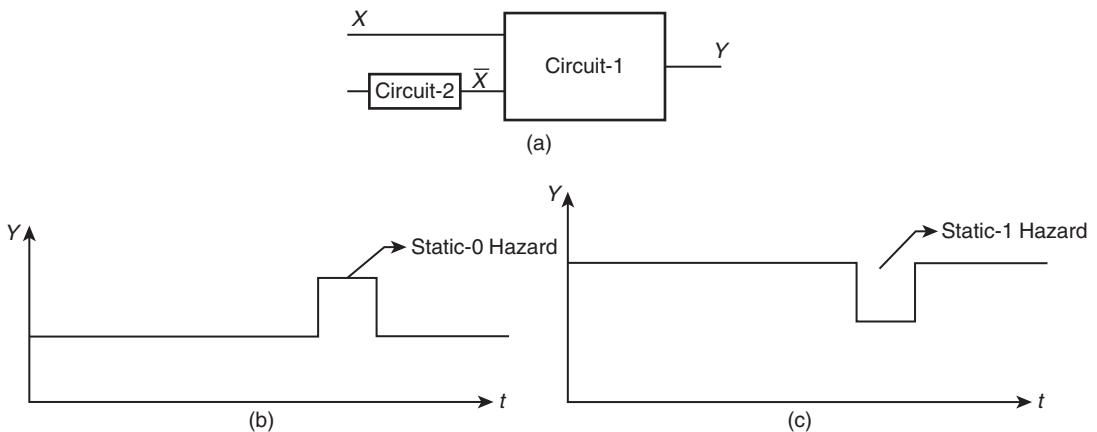


FIGURE 10.2 (a) General circuit network consisting of logic circuit-1 and circuit-2 (b) Timing diagram for Y showing a static-0 hazard operation (c) Timing diagram for Y showing a static-1 hazard operation.

10.1.2 Static-1 Hazard

Static-1 hazard is a hazard in which a Boolean algebraic output or next state output expected is 1, but momentarily it undergoes transition to 0. For example, the circuit of Figure 10.1(a) expected $Y = 1$, but it becomes 0 for a period of propagation delay $\Delta t'$ of the complementing circuit. Figure 10.2(c) shows the output, which creates a static-1 hazard.

Point to Remember

Static-0 hazard arises when two logic circuit elements one of which is using X and other X and are expected to give an output 0 but due to the differing delays, the momentarily the output undergoes transition to 1.

Static-1 hazard arises when two logic circuit elements one of which is using X and other \bar{X} and are expected to give an output 1 but due to the differing delays, momentarily the output undergoes transition to 0.

Consider a circuit of Figure 10.3. Assume the followings for the circuit:

1. It gives two external inputs X_1 and X_0 to an asynchronous circuit element. The X_1 has combination circuit such that it is at 1 and has static-1 hazard due to the use of two paths one for X_1' and other for \bar{X}_1' .
2. The asynchronous circuit element is such that it has the transition table as per Table 10.1.

Consider row 1. When X_1 and X_0 are 10 and 11, present state (x_{q0}, x_{q1}) is $(0, 0)$, the present outputs are 1 in both cases of the inputs. Assume that X_1 has a static-1 hazard due to presence of an X_1 output implementing circuit, which has the delaying element. X_1 undergoes a static hazard, such that momentarily it undergoes a transition to 0 from 1. During the period when $X_1 X_0$ are 00 in place of 10, the next state becomes $(0, 1)$ and output becomes 0. It will lead to either (i) another next state unstable $(0, 0)$ with output = 0 in case the $X_1 X_0$ again settle to 10 or (ii) to a stable $(0, 1)$ with output = 1.

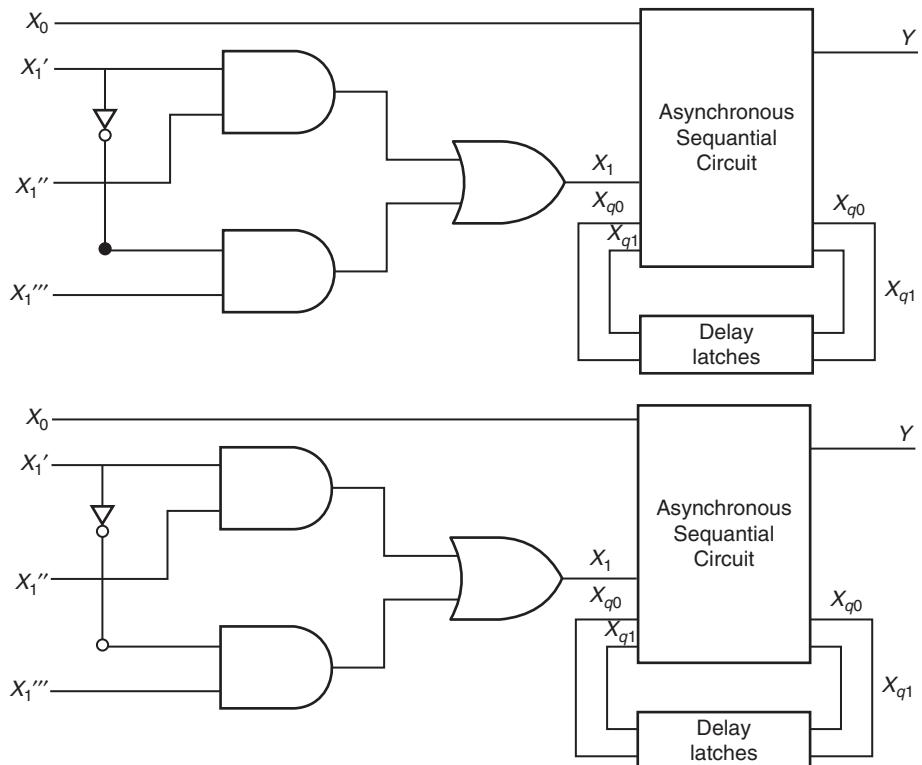


FIGURE 10.3 A circuit with two outputs, which are two external inputs X_1 and X_0 to an asynchronous circuit element with X_1 having a preceding combination circuit with static-1 hazard due to the use of two paths one for X_1' and other for X_1'' .

TABLE 10.1 Excitation-cum-Transition table* for an exemplary circuit consisting of two circuit elements one with a very small propagation and other with a significant delay

Present state (x_{q0}, x_{q1})	Next state after transition (x'_{q0}, x'_{q1})				Present output Y_0			
	Input $X_1X_0 = 00$	Input $X_1X_0 = 01$	Input $X_1X_0 = 10$	Input $X_1X_0 = 11$	Input $X_1X_0 = 00$	Input $X_1X_0 = 01$	Input $X_1X_0 = 10$	Input $X_1X_0 = 11$
0, 0	0, 1	0, 0	1, 0	1, 0	0	0	1	1
0, 1	0, 1	0, 1	0, 0	1, 0	1	0	0	0
1, 0	1, 1	1, 1	1, 0	1, 0	0	1	0	0
1, 1	1, 1	0, 1	1, 1	1, 0	1	1	0	1

10.2 IDENTIFYING STATIC HAZARDS

The question is how can one find that what are the hazards. First, follow the following steps:

Step 1: Write Boolean expression for the logic circuit and assume variable X and its complement as two separate variables.

Step 2a: Find POS form of the expression to identify static-0 hazard without using $X_0 + \bar{X}_0 = 1$ and $X_0 \cdot \bar{X}_0 = 0$ or expressions dependent on these, where X_0 is a Boolean variable that is used both as X_0 and \bar{X}_0 . For example, use $X_1 + X_2 = X_1 + X_1 \cdot X_2$ or $X_1 \cdot X_2 + X_1 \cdot X_3 = X_1 \cdot X_2 + X_2 \cdot X_3 + X_1 \cdot X_3$ or $X_1 \cdot (X_1 + X_2) = X_1 \cdot X_2$ is not permitted when finding static hazards. Prefer use of DeMorgan theorem or distributive laws.

Step 2b: Find SOP form of the expression to identify static-1 hazard without using $X_0 + \bar{X}_0 = 1$ and $X_0 \cdot \bar{X}_0 = 0$ based expressions, where X_0 is a Boolean variable that is used both as X_0 and \bar{X}_0 .

10.2.1 Identification from the Boolean Expressions

For example, consider the circuit of Figure 10.1(a).

$$\begin{aligned}
 Y &= \overline{X \cdot \bar{X}} && (X \cdot \bar{X} = 0, \text{ not allowed to be used for analyzing transients).} \\
 &= \bar{X} + X; && (\text{Using DeMorgan theorem, get an OR expression.}) \\
 &&& (\text{Modify further as OR expression not allowed in this form for analysis).}) \\
 &= \bar{X} + X \cdot X; && (\text{Use rule } X_1 \cdot X_1 = X_1). \\
 &= X_1 + X_2 \cdot X_2; && (\text{Assume a variable and its complement as a separate set of two variables).})
 \end{aligned}$$

When $X_1, X_2 = 0, 0$ the output = 0.

When $X_1, X_2 = 1, 0$ the output = 1.

When $X_1, X_2 = 1, 1$ the output = 1

When $X_1, X_2 = 0, 1$ the output = 1

We find that there is one condition of X_1 and X_2 in which output will show transient to 0. Hence circuit has static-1 hazard.

10.2.2 Identification from the Karnaugh Map (Only One-variable Input Case)

Use steps 1, 2a and 2b described in Section 10.2.

Step 3: Draw two variable Karnaugh map by assuming the \bar{X}_0 and \bar{X}_0 as separate variables, X_1 and X_2 .

Step 4: Find two pair of diads at right angle (90°) to each other. [Diad means pair of adjacent cells with the 1s or 0s in SOP or POS, respectively.] Show the pairs by dotted lines, as circuit for implementing does not exist. It is needed to implement that path when there is a transition of the variable. When we change from adjacent column to another, there is a single variable change. Diads at the right angle means, when the complement variable changes by following another path to final output, there is a reversal of the state.

Static 1 Hazard

Steps 1 and 2b have already been given in Section 10.2.1. Karnaugh map form SOP form of the Figure 10.1(a) circuit is shown in Table 10.2. It shows two diads at the right angles with dotted envelopes. Hence the circuit has a static-1 hazard for transition 10 to 11.

TABLE 10.2 Two Variable SOP Karnaugh map for the circuit of Figure 10.1(a)

X_1	X_2	0	1
0	0	0	1
1	1	1	1

Dotted envelope means circuit does not exist to implement it. There is only one variable present. Arrows show the transitions.

Static 0 Hazard

Now let us find static-0 hazard, if any, in circuit of the Figure 10.1(c) assuming NOT gate as the delaying element for the period $\Delta t'''$. Combination circuit of the figure be assumed to be AND gate of shorter delay $\Delta t''$.

Following are the steps to find static 0 hazard:

Step 1: Boolean expression for the circuit is $Y = \bar{X}_1 \cdot X_1$. (Not permitted to be used for transient analysis).

Step 2: Find POS form as follows.

$$\begin{aligned}
 \bar{Y} &= \overline{\bar{X}_1 \cdot X_1} \\
 &= \overline{X_1 + \bar{X}_1} \quad (\text{Use DeMorgan Theorem}). \\
 &= \overline{X_1 + X_2} \quad (\text{Assume } X_1 \text{ and } \bar{X}_1 \text{ as a set of two separate variables}).
 \end{aligned}$$

When $\bar{X}_1, \bar{X}_2 = 0, 0$ the $\bar{Y} = 0$

When $\bar{X}_1, \bar{X}_2 = 1, 0$ the $\bar{Y} = 0$

When $\bar{X}_1, \bar{X}_2 = 1, 1$ the $\bar{Y} = 1$

When $\bar{X}_1, \bar{X}_2 = 0, 1$ the $\bar{Y} = 0$

Draw the Karnaugh map as per Table 10.3 for POS form (representation of \bar{Y} and on the map) of the expression.

TABLE 10.3 Two Variable Karnaugh map POS form forthe circuit of Figure 10.1(c)

\bar{X}_1	\bar{X}_2	1	0
1	1	0	0
0	0	0	0

Arrows show the transition

Map shows two diads at right angle to each other. Hence the output has a static-0 hazard.

[Example 10.1 will show the static 1 and static 0 hazard detection for the circuit for X_i in Figure 10.3.]

10.2.3 Identification from the Karnaugh Map (Three-Variable Input)

First use steps 1, 2a and 2b as described in Section 10.2.

Step 3: Draw Karnaugh map for the three-variable case.

Step 4: Find a pairs of diads with no overlapping between the two diads.

Consider when we change from adjacent column to another, there is a single variable change. If in the given logic circuit, that path does not exist, then a static hazard arises.

Static 0 Hazard

Find from the three-variable in POS-form map, a no-overlap diad (s) between the pair of diads. No overlap means there is no logic circuit existing to implement that path on a transition of the variable. A pair of complementary variables X_1 and X_2 means X_1 change between 0 and 1 in first diad and X_2 in second diad. A pair of 0s in these adjacent cell diads separate by the two complementary pairs. In case of overlapping (available circuit to implement), the 0s in each differ only by one of the variable change not by two variable changes. Static 0 hazard exists when there exists a pair of 0s in these adjacent cell diads separate by the two complementary pairs. Let the circuit POS form is as follows.

$$\bar{Y} = (X_1 + X_2) \cdot (\bar{X}_3 + \bar{X}_2); \text{ (Without using above mention OR and AND equations).}$$

...(10.1)

Table 10.4 shows a Karnaugh map with two full lined pair diads *a* and *b* with non-overlapping 0s between them and thereby having a static 0 hazard during transition from 111–100. Consider the shaded envelope for a diad pair *c* over a non-overlapping diad pair. Pair forms *b* and *c* by two adjacent cells in the table by wrapping adjacency. It means the logic circuit $(\bar{X}_1 + \bar{X}_3)$ and therefore the term *C* is not present to implement transition along that path. Full-line envelope over a diad pair means the expression terms and logic circuit are available to implement it.

TABLE 10.4 Three variable Karnaugh map for $\bar{Y} = (X_1 + X_2) \cdot (X_3 + X_2)$

$\bar{X}_1 \bar{X}_2$	\bar{X}_3	1	0
11	$\bar{X}_1 + \bar{X}_2$	a 0	d 0 ←
01	$X_1 + \bar{X}_2$		
00	$X_1 + X_2$		b 0
10	$\bar{X}_1 + X_2$		c 0 ←

When the logic circuit does not have the term $(\bar{X}_1 + \bar{X}_3)$, 111 – 100 the transition has a Static-0 Hazard

Note: Full line box over a diad means the terms exist in equation (10.1) and the shaded box over a diad means term (circuit) does not exist in Boolean expression and that creates a hazard 0 in the present case.

Static 1 Hazard

Three-variable SOP-form map with the no overlap between the pair of diads means, when there is a pair of complementary variables, which change between a 0 in first

diad of 1s and a 1 in second diad of 0s. A pair of 1s in these adjacent cell diads separate by the two complementary pairs. In case of overlapping diads, the 1s in each differ only by one of the variable change not by two variable changes. Static 1 hazard exists when there exists a pair of 1s in these adjacent cell diads separate by the two complementary pairs. Static 1 hazard occurs when there is a transition from one diad cell to another diad adjacent cell.

Assume that the Boolean expression for the logic circuit in SOP form is as follows:

$$Y = \bar{X}_2 \cdot X_1 + \bar{X}_3 \cdot X_2; \text{ (Without using OR and AND equations in Boolean laws).} \quad \dots(10.2)$$

Table 10.5 shows the Karnaugh map with two pair diads (a, b) with non-overlapping 1s between them and thereby having a static 1 hazard during transition 110–101. Refer the dotted envelope over a non-overlapping diad pair. Pair forms by the two adjacent cells in the table by adjacency or wrapping adjacency. It means the logic circuit and therefore the term c not present to implement transition along that path. Full-line envelope over a diad pair means the expression terms and the logic circuit is available to implement it.

TABLE 10.5 Three variable SOP form Karnaugh map for circuit $Y = \bar{X}_2 \cdot X_1 + \bar{X}_3 \cdot X_1$

X_3	0	1
$X_1 X_2$		
00		
10	a [1] 1	
11	c [1]	
01	b [1]	

When the logic circuit does not have the term $X_1 \cdot \bar{X}_3$ there is a static-1 Hazard

Note: The full line box is because of diad pair $\bar{X}_2 \cdot X_1 \cdot (\bar{X}_3 + X_3)$ and shaded box is for $X_2 \cdot (X_1 + \bar{X}_1) \cdot \bar{X}_3$ pair. Equation does not have a term $X_1 \cdot \bar{X}_3$ and therefore the shaded diad and therefore a hazard is present.

10.2.4 Detecting Absence of Static 1 Hazard from the POS Form of Boolean Expression

If a Boolean expression has a term present in POS or SOP with either a variable or its complement only, then a static 0 or 1 hazard is ruled out, respectively. This however does not mean that if a variable as well as complement occur in the POS or SOP expression, then static 1 or 0 hazard may or may not be present, respectively. Presence can then be detected by Karnaugh map for appropriate form.

For example, the circuit implementation of the following first two expressions do not have static 1 hazard:

$$\bar{Y} = (X_1 + \bar{X}_2) \cdot (\bar{X}_2 + X_3);$$

(X_1, X_3 and \bar{X}_2 complements are not existing. So static 1 hazard ruled out).

$$\bar{Y} = (X_1 + \bar{X}_3).(X_1 + \bar{X}_2).(\bar{X}_3 + \bar{X}_2);$$

(X_1, \bar{X}_2 and \bar{X}_3 complements don't exist. So static 1 hazard ruled out. Third term is always 1 and hence not taken into account).

$$\bar{Y} = (X_1 + X_3).(\bar{X}_1 + X_2 + X_3).(\bar{X}_3 + X_2);$$

(X_1 and X_3 complements also exist. So static 1 hazard is not ruled out, may be present or may not be present).

10.2.5 Detecting Absence of Static 0 Hazard from the SOP Form of Boolean Expression

If a Boolean expression has a term present in SOP with either a variable or its complement only, then static 0 hazard is ruled out. This is however does not mean that if a variable as well as complement occur in the expression, then static 1 hazard may or may not be present. Presence can then be detected by Karnaugh map for SOP form.

For example, the circuit implementation of the following expressions do not have static 0 hazard:

$$Y = \bar{X}_1 + X_2;$$

(\bar{X}_1 and X_2 complements are not existing. So static 0 hazard ruled out).

$$Y = X_1.X_3 + X_1.X_2.X_3 + \bar{X}_3.X_3;$$

(X_1, X_2 and X_3 complements don't exist. So static-0 hazard ruled out. Third term is always 0 and hence not taken into account).

$$Y = X_1.X_3 + \bar{X}_1.X_2.X_3 + \bar{X}_3.X_3;$$

(X_1 complement also exists. So static-0 hazard is not ruled out, may be present or may not be present).

Points to Remember

1. Step 1: Boolean expression
2. Step 2a and 2b: POS and SOP forms to detect static 0 and static 1 hazards
3. Step 3: Karnaugh maps for POS and SOP forms
4. Step 4: Analysis by finding the non existant adjacencies and wrapping adjacencies, which are in the map but not in the given Boolean expression.

10.3 ELIMINATING STATIC HAZARDS

A static hazard occurs because a variable and its complement passed through different number of logic operations. For example, in circuit of Figure 10.1(a), X input passed through one NAND and X passed through two NANDs. If an additional

holding circuit adds into the logic circuit, the static hazard eliminates. For example adding an AND gate with common inputs after X and before the output stage NAND, will eliminate the static hazard.

What is the systematic way of eliminating a static hazard? We follow steps 1, 2a, 2b, 3 and 4 described in Section 10.2 and obtain a Karnaugh map as given in Table 10.4 or Table 10.5. Term not available between a pair of diads is shown by a shaded pair. Addition of that term eliminates the hazard (Section 10.2.3).

Eliminating Static 0 Hazard

For example, consider the circuit of Figure 10.4(a), which implements the Boolean expression (10.1) and corresponds to the POS form map shown in Table 10.4.

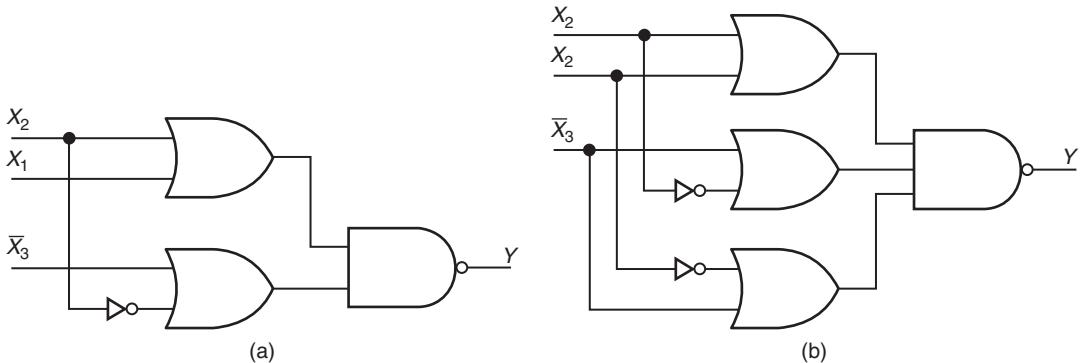


FIGURE 10.4 (a) A circuit, which implements the Boolean expression (10.1) and corresponds to the POS form map (shown in Table 10.4) (b) The circuit after removing static-0 hazard by using expression (10.3).

The expression (10.1) if added with the term $(\bar{X}_1 + \bar{X}_3)$ as follows, then the static-0 hazard eliminates. This term corresponds to dashed-pair implementation by the circuit.

$$Y = (X_1 + X_2) \cdot (\bar{X}_3 + \bar{X}_2) \cdot (\bar{X}_1 + \bar{X}_3) \quad \dots(10.3)$$

Figure 10.4(b) shows the new circuit for static-0 hazard free implementation.

Eliminating Static 1 Hazard

For example, consider the circuit of Figure 10.5(a), which implements the Boolean expression (10.2) and corresponds to the SOP form map shown in Table 10.5.

The expression (10.2) if added with the term $(X_1 \cdot \bar{X}_3)$ as follows, then the static-1 hazard eliminates. This term corresponds to dotted-pair implementation by the circuit.

$$Y = (\bar{X}_2 \cdot X_1) + (\bar{X}_3 \cdot X_2) + (X_1 \cdot \bar{X}_3) \quad \dots(10.4)$$

Figure 10.5(b) shows the new circuit for static 1 hazard free implementation using the expression (10.4).

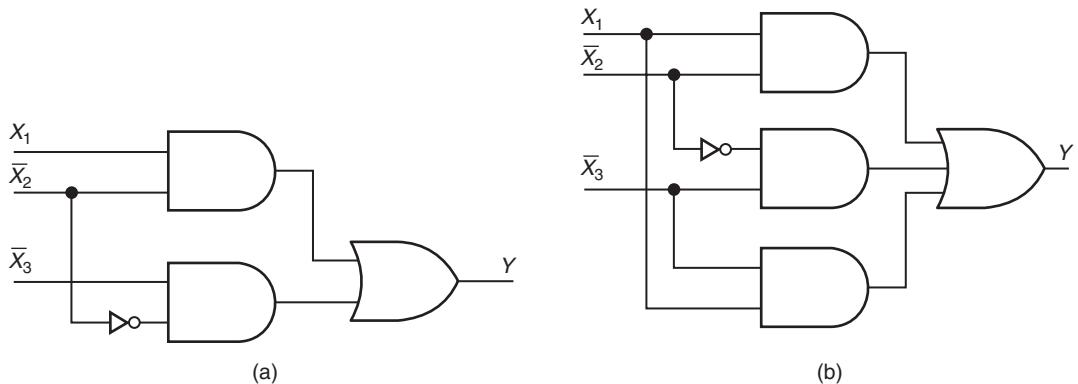


FIGURE 10.5 (a) A circuit, which implements the Boolean expression (10.3) and corresponds to the SOP form map shown in Table 10.5 (b) The circuit after removing static 1 hazard by using expression (10.4).

Point to Remember

Use of additional logic circuit terms in POS and SOP terms eliminates static 0 and static 1 hazards, respectively.

10.4 DYNAMIC HAZARDS

When an output, though suppose to undergo transition only once, if changes total $(2n + 1)$ times, where $n = 1$ or 2 or 3 and so on, then there is a dynamic hazard. The n is the number of times the output toggle back to the initial value. Dynamic hazards arise due to later stage gates undergoing the transitions slower than the previous stage gates or vice versa. Figures 10.6(a) and 10.6(b) show the output transitions due to presence of the dynamic hazards. Dynamic hazards can also cause logic circuit malfunction.

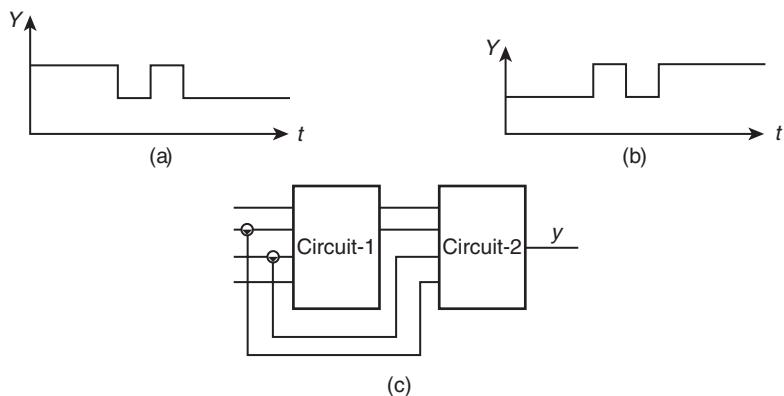


FIGURE 10.6 (a) Output 1 to 0 transition in presence of a dynamic hazard (b) Output 0 to 1 transition in presence of a dynamic hazard (c) Circuit, which is potential source of the dynamic hazards.

Consider a circuit combination of Figure 10.6(c). It consists of the circuit-1 and circuit 2. Let circuit-2 gets part of the inputs from circuit-1 and part of the outputs directly. Therefore, the different inputs of circuit-2 are appearing at the different instances. This is a potential source of dynamic hazards.

10.5 HAZARDS FREE CIRCUITS

Elimination static 0 and static 1 hazards have been described above. If each variable, which is used as the input at any stage (after adding the terms in POS and SOP that eliminates the static hazards), is used in only one form—either as such or complement form, then the circuit is hazard free.

Hazard free circuit is a circuit free from static as well as dynamic hazards.

10.6 ESSENTIAL HAZARDS

Consider asynchronous sequential circuit. Let us assume that (i) Static hazards have been eliminated (ii) Dynamic hazard has been eliminated (iii) Races that are critical have been eliminated (iv) The circuit operates in fundamental mode so that the input is changed only after a stable state is reached.

Asynchronous sequential circuit has following structure:

1. There is a set \underline{X}_i of m present input variables X_0, X_1, \dots, X_{m-1} . These are applied along with present \underline{x}_q to a combinational circuit. The output of \underline{Y}_j along with \underline{y}_q is a set consisting of present state outputs \underline{Y} of which \underline{y}_q is a subset. (\underline{Y} consists of two subsets; \underline{Y}_j with outputs Y_0, Y_1, \dots, Y_{j-1} with no feedback and \underline{y}_q with feedback to the input via the memory section).
2. The circuit memory section consists of the m latches without clock edges or pulse inputs to control the instance or period of their operations. These have a set \underline{x}_q variables from the m present state outputs $\underline{x}_q = x_{q0}, x_{q1}, \dots, x_{qn-1}$ from the y_q inputs after a delay.
3. The present state of \underline{Y} and therefore its subset \underline{y}_q changes by the change in \underline{x}_q after a delay at the memory section.

Another hazard will still be present. It is essential hazard. It is due to different instances at which change in the excitation variable \underline{y}_q occurs when there is a change in an input variable X_i . Essential hazard arises out that an input variable affects the different feedback cycle variables at different instances. Before the expected set of all \underline{y}_q excitation input changes finish, the state variable(s) \underline{x}_q can change, which may lead to circuit not functioning as expected.

Consider a circuit shown in Figure 10.7(a), for which Table 10.6 gives the excitations-cum-transitions.

Suppose inputs $X_1 X_0$ are at 11. Therefore, the stable state as per column 5 row 3 is (0, 1) with output $Y=1$. Let X_0 change from 1 to 0. Expected stable state is (1, 0) with output $Y=1$ as per column 4 row 1. There will be one additional feedback cycle, which is shown by a vertical transition in the table and then only the stable state reaches.

10.14 Digital Systems: Principles and Design

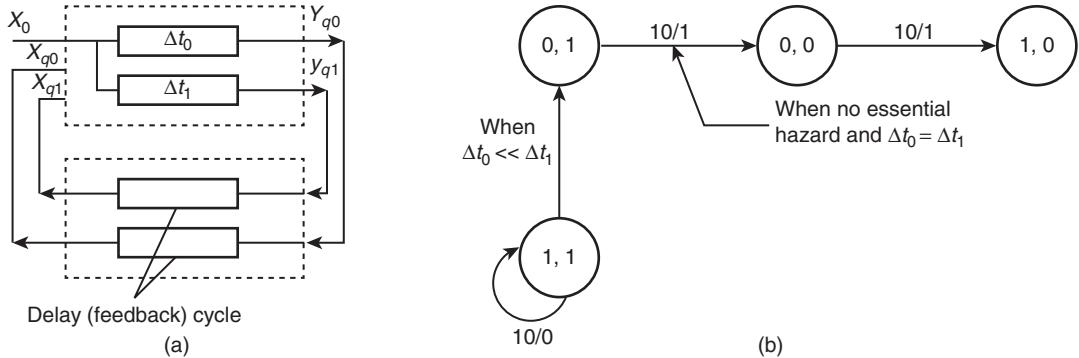


FIGURE 10.7 (a) Structure of an asynchronous sequential circuit, which has excitations and transitions as per Table 10.6 (b) Resulting effects due to essential hazard from $\Delta t_0 \ll \Delta t_1$, assuming that feed cycle delays are same for both transitions to x_{q0} and x_{q1} .

TABLE 10.6 Excitation-cum-Transition table for an asynchronous sequential circuit exemplary circuit-6, which has (Table 9.5)

Present state (x_{q0}, x_{q1})	Next state after transition (x'_{q0}, x'_{q1})				Present output Y_0			
	$X_1 X_0 = 00$	$X_1 X_0 = 01$	$X_1 X_0 = 10$	$X_1 X_0 = 11$	$X_1 X_0 = 00$	$X_1 X_0 = 01$	$X_1 X_0 = 10$	$X_1 X_0 = 11$
$0, 0$	0, 1	0, 0	1, 0	1, 0	0	0	1	1
$0, 1$	0, 1	0, 1	1, 1	1, 0	1	0	1	0
$1, 0$	0, 1	1, 1	0, 0	0, 1	1	1	1	1
$1, 1$	0, 1	0, 0	1, 1	1, 0	0	1	0	1

Assume that the 1-to-0 transition of X_0 affects y_{q0} and therefore x_{q0} earlier in time Δt_0 than y_{q1} , x_{q1} affects in time Δt_1 , which is later. ($\Delta t_0 \ll \Delta t_1$. and assume that feed cycle delays are same for both transitions to x_{q0} and x_{q1}).

Therefore, temporarily, the state becomes $(1, 1)$ with output $Y = 0$. The stable state achieved is $(1, 1)$ with $Y = 0$.

Due to essential hazard arising out of quicker excitation of one feedback than another when there is an input change, the different stable state has resulted. Figure 10.7 (b) show the resulting transitions.

Point to Remember

Essential hazard in an asynchronous sequential circuit operation arises out when an input variable affects the different feedback cycle excitation variables at the different instances. Introduction of the additional delays greater than the delays in the combination circuits providing the excitations in the feedback cycles eliminates essential hazard.

10.7 PULSE MODE SEQUENTIAL CIRCUIT

Synchronous sequential circuits undergo the excitations on clock edges. Synchronous circuit operated on the periodic pulses of the clock is called pulse mode sequential circuit.

When the period ΔT of the clock pulse, which excites a synchronous sequential circuit is greater than the time periods during which an input variable(s) affects the different feedback cycle excitation variables, the circuit will be free from essential hazards.

Figure 10.8 shows a pulse mode sequential circuit.

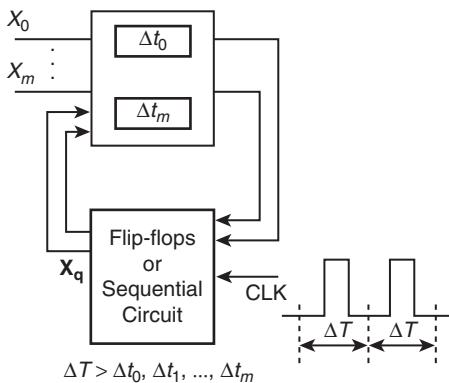


FIGURE 10.8 Pulse mode sequential circuit.

■ EXAMPLES

Example 10.1

Detect the static 1 and static 0 hazards for x_1 output in the circuit of Figure 10.3 and find the missing circuit to provide transition paths.

Solution

Step 1: First we write the Boolean expression for the logic circuit of Figure 10.3.

$$X_1 = (X_1' \cdot X_1'') + (\bar{X}_1' \cdot X_1''')$$

Step 2a: POS form for static-0 hazards detection

Use DeMorgan theorem as follows:

$$\begin{aligned} \bar{X}_1 &= \overline{(X_1' \cdot X_1') + (\bar{X}_1' \cdot X_1''')} \\ &= \overline{(X_1' \cdot X_1'')} \cdot \overline{(X_1' \cdot X_1''')} \\ &= (\bar{X}_1' + \bar{X}_1'').(X_1' + \bar{X}_1''') = (\bar{X}_1' + \bar{X}_1'' + \bar{X}_1''').(\bar{X}_1' + \bar{X}_1'' + X_1''').(X_1' + \\ &\quad \bar{X}_1'' + \bar{X}_1''').(X_1' + X_1'' + \bar{X}_1''') \end{aligned}$$

[On expanding the minimized 2-variable to 3-variable expression] Two terms become four

Step 3: Three Variable Karnaugh map for $(\bar{X}_1' + \bar{X}_1'').(\bar{X}_1' + \bar{X}_1''')$

10.16 Digital Systems: Principles and Design

TABLE 10.7 Two terms expression expanding to four terms for $(X_1' \cdot X_1''') + (\bar{X}_1 \cdot X_1'')$ POS

\bar{X}_1'''	1 X_3	0 Z_1
$\bar{X}_1' \bar{X}_1''$		
11 $\bar{X}' + \bar{X}''$	0	0 a
01 $X' + \bar{X}''$	0	
00 $X' + X''$	b	0
10 $\bar{X}' + X''$		

When the logic circuit does not have the term $(X_1' + \bar{X}_1''')$, 110 - 000 transition is a static-0 hazard

There is a static 0 hazard in 110-to-000 transition of $X_1', X_1'', \bar{X}_1'''$.

Step 2b: SOP form for static-1 hazards detection

$$Y_1 = (X_1' \cdot X_1'') + (\bar{X}_1' \cdot X_1''') = X_1' \cdot X_1'' \cdot \bar{X}_1''' + X_1' \cdot X_1'' \cdot X_1'''$$

+ $\bar{X}_1' \cdot \bar{X}_1'' \cdot X_1''' + \bar{X}_1' \cdot X_1'' \cdot X'''$ [On expanding the minimized 2 variable to 3 variable expression.]

Step 3: Three variable Karnaugh map for $(X_1' \cdot X_1'') + (X_1' \cdot X_1''')$

Static-1 hazard is present in 011-to-111 transition.

TABLE 10.8 Two terms expression expanded to $(X_1' \cdot X_1''') + (\bar{X}_1 \cdot X_1'')$ SOP map

X_1'''	0	1
$X_1' \bar{X}_1''$		
00		b 1
10		
11	1	1
01	c	1

When the logic circuit does not exist for $X_1' \bar{X}_1''$, 111 to 001 $\bar{X}' \bar{X}_1'' \cdot X_1'''$ is a static-1 hazard

Example 10.2 From the given Karnaugh three variables maps (Table 10.9) find the static 1 and static 0 hazards and show that logic path exists such that no hazards exists in first map.

Solution

TABLE 10.9 Three Variable Karnaugh map for $Y = (\bar{X}_1 + \bar{X}_2) \cdot (\bar{X}_3 + \bar{X}_1) \cdot (X_2 + \bar{X}_3)$ map (when all variables are 1s in the expression)

\bar{X}_3	1	0
$\bar{X}_1 \bar{X}_2$		
11 $\bar{X}_1 + \bar{X}_2$	0	0 ←
01 $X_1 + \bar{X}_2$		
00 $X_1 + X_2$		0 ←
10 $\bar{X}_1 + X_2$		0 ←

Due to $(\bar{X}_3 + \bar{X}_1)$ presence, no static 0 hazard present, as all variables are in the form of 0 only. No static 0 hazard present

Also, no static 0 hazard present as all the three paths (diads) involving single variable change are available in the circuit of the Boolean expression.

Static-1 hazard exists for 001 to 010 transition a path does not exist in which there is single variable change to implement. [There is X_2 as well as \bar{X}_2 in the Boolean expression.]

Example 10.3

From the given Karnaugh three variable maps (Tables 10.10 and 10.11) find the static 1 and static 0 hazards and show that no hazard exists. Show the logic existence using Karnaugh map.

TABLE 10.10 Three Variable SOP form Karnaugh map for circuit $Y = \bar{X}_2 \cdot \bar{X}_1 + X_3 \cdot X_2$

X_3	0	1
$X_1 \cdot X_2$		
00 $\bar{X}_1 \cdot \bar{X}_2$	1	1
10 $X_1 \cdot \bar{X}_2$		
11 $X_1 \cdot X_2$		1
01 $\bar{X}_1 \cdot X_2$		1

When the logic circuit does not have the term $\bar{X}_1 \cdot X_3$ static-1 hazard

TABLE 10.11 Three variable SOP form Karnaugh map for circuit $Y = \bar{X}_2 \cdot \bar{X}_1 + \bar{X}_3 \cdot \bar{X}_2$

X_3	0	1
$X_1 \cdot X_2$		
00	1	1
10	1	
11		
01		

Due to an overlapping diad through 000 ($\bar{X}_2 \cdot X_1$) term, no static-1 hazard exists in 000 to 001 or 100 transition.

Solution

Table 10.10 map shows static-1 hazard. In SOP form Table 10.11, the variables in the Boolean expression are only the complement forms of X_2 , X_1 and X_3 and there is no dual use of variable and its complement. Hence static 0 hazard is also ruled out.

Example 10.4

From the given Karnaugh two variables map (Table 10.12) for $\bar{X}_2 \cdot X_1 + \bar{X}_1 \cdot X_2$, find the static 1 and static 0 hazards and show that a static 1 hazard exists. Find the missing terms, which would have made hazards non existant.

10.18 Digital Systems: Principles and Design

TABLE 10.12 Two variable SOP form Karnaugh map for circuit $Y = \bar{X}_2 \cdot X_1 + \bar{X}_1 \cdot X_2$

X_2	0	1
X_1		
0	.	1
1	1	

Static-1 hazard exists in 10 to 01 as it involves two single variable path changes as the term $\bar{X}_1 \bar{X}_2$ does not exist

Solution

In SOP form, the variables used are un-complemented as well as un-complemented complements of X_2, X_1 . Hence static-0 hazard can't be ruled out.

Let us find POS form (Table 10.13) of the logic circuit to find the hazard source.

$$\bar{Y} = \overline{\bar{X}_2 \cdot X_1 + X_2 \cdot \bar{X}_1} = \overline{\bar{X}_2 \cdot X_1} \cdot \overline{X_2 \cdot \bar{X}_1} = (X_2 + \bar{X}_1) \cdot (\bar{X}_2 + X_1)$$

TABLE 10.13 Three variable POS form Karnaugh map for circuit $\bar{Y} = X_2 \cdot \bar{X}_1 + X_1 \cdot \bar{X}_2$

\bar{X}_2	1	0
\bar{X}_1	.	
1	.	0
0	0	

Static-0 hazard exists in 10 to 01 as it involves two single variable path changes. The missing term $\bar{X}_1 \bar{X}_2$ is the hazard source.

Example 10.5

Find whether static 0 hazard does not exist in implementing the Boolean expression: $Y = X_1 \cdot X_3 + X_1 \cdot \bar{X}_2 \cdot X_3 + X_3 \cdot \bar{X}_2$.

Solution

Since complements of X_1, \bar{X}_2 and X_3 do not occur in any of the terms, the static 0 hazard does not exist.

Example 10.6

Find whether static-0 hazard may or may not exist in implementing the Boolean expression: $Y = X_1 \cdot X_3 + \bar{X}_1 \cdot \bar{X}_2 + \bar{X}_2 \cdot X_3$. Can you rule out the existence of static-1 hazard when implementing the following POS form circuit?

Solution

$$\bar{Y} = (\bar{X}_1 + \bar{X}_3) \cdot (X_1 + X_2) \cdot (\bar{X}_3 + X_2);$$

Since X_1 and its complement exist in the terms, the existence of static 1 can't be ruled out, may be present or may not be present.

Example 10.7

$Y = (X_1 + X_3) \cdot (X_1 + X_2 + X_3) \cdot (X_3 + X_2)$ has no static-1 hazard. Find out.

Solution

Since no variable is present in both forms (variable and its complement) in this POS expression, hence the static 1 hazard is ruled out.

Example 10.8 From the given transition table (Table 10.14) find whether essential hazard exists.

TABLE 10.14 Excitation-cum-Transition table for an asynchronous sequential circuit

Present state	Next state after transition (x'_{q0}, x'_{q1})		Present output Y_0	
	Input $X_0 = 0$	Input $X_0 = 1$	Input $X_0 = 0$	Input $X_0 = 1$
(x_{q0}, x_{q1})				
0, 0	0, 0	1, 0	0	1
0, 1	0, 1	0, 1	0	1
1, 0	0, 0	1, 0	1	0
1, 1	0, 1	0, 0	0	1

No essential hazard exists when the state as (0,1) as the X transition does not affect the final stable condition. No essential hazard exists in state (0, 0). Because when X changes from 0 to 1, state will eventually settle to (0, 1) though after a delay in case x_{q0} changes early and x_{q1} later or vice-versa after a delay.

■ EXERCISES

1. Detect the static 1 and static 0 hazards for X_1 output in the circuit of Figure 10.9.

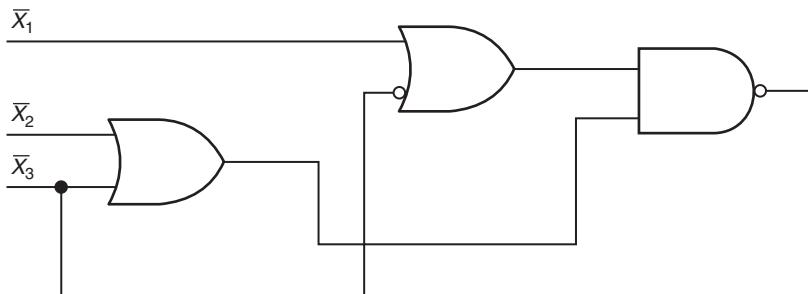


FIGURE 10.9 Circuit for Exercise 1.

2. From the given Karnaugh three variable map, (Table 10.15) write the Boolean expression, draw the circuit and find the static 1 and static 0 hazards

TABLE 10.15 Three variable Karnaugh map

$\bar{X}_1 \bar{X}_2$	\bar{X}_3	1	0
11		0	0
01		0	0
00			
10			

10.20 Digital Systems: Principles and Design

3. From the given Karnaugh three variable map (Table 10.16) find the Boolean expression, logic circuit, and static-1 and hazards.

TABLE 10.16 Three variable SOP form Karnaugh map for circuit

$X_3 \backslash X_1 X_2$	0	1
00	1	1
10	1	1
11		1
01		1

4. From the given Karnaugh two variables map (Table 10.17), find the static 1 and static 0 hazards and find that a static 1 hazard exists or not.

Table 10.17 A Karnaugh map for two variables

$X_2 \backslash X_1$	0	1
0	1	1
1	1	1

5. Find whether static 0 hazard does not exist in implementing the Boolean expression: $Y = X_1 \cdot X_3 + X_1 \cdot X_2 \cdot X_3 + \bar{X}_2 \cdot X_3$. If exists, then find the static hazards present.
6. Find whether static 0 hazard may or may not exist in implementing the Boolean expression: $Y = \bar{X}_1 \cdot X_3 + \bar{X}_1 \cdot X_2 \cdot X_3 + X_2 \cdot X_3$. If exists, then find the static hazards present.
7. Can you rule out the existence of static 1 hazard when implementing the following POS-form circuit?

$$\bar{Y} = (\bar{X}_1 + X_3) \cdot (\bar{X}_1 + \bar{X}_2) \cdot (\bar{X}_3 + X_2)$$

8. $\bar{Y} = (X_1 + X_3) \cdot (\bar{X}_1 + X_2 + \bar{X}_3) \cdot (X_3 + X_2)$ has static hazards. Trace them.
9. From the given transition table (Table 10.18) find whether essential hazard exists. If existing, list these.
10. From the given circuit find in Figure 10.10 whether dynamic hazard exist.
11. What are the additional circuits needed to remove static a hazard in exercise 2 and in 3.
12. Detect all the hazards possible in circuit of Figure 10.11 and find ways to eliminate these.

TABLE 10.18 Transition table for an asynchronous sequential circuit

Present state (x_{q0}, x_{q1})	Next state after transition (x'_{q0}, x'_{q1})		Present output Y_0	
	Input $X_0 = 0$	Input $X_0 = 1$	Input $X_0 = 0$	Input $X_0 = 1$
0, 0	1, 0	0, 1	0	1
0, 1	1, 1	0, 1	0	1
1, 0	0, 0	1, 0	1	0
1, 1	0, 1	0, 1	0	1

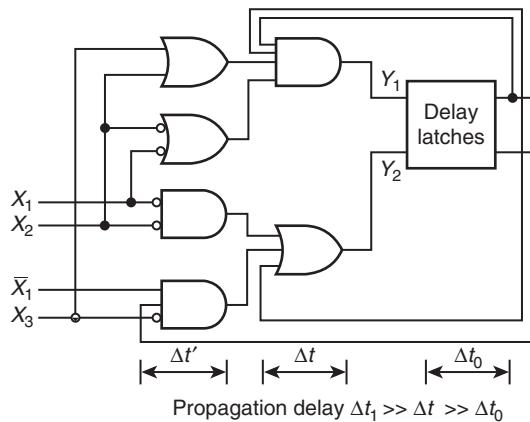


FIGURE 10.10 Circuit for Exercise 10.

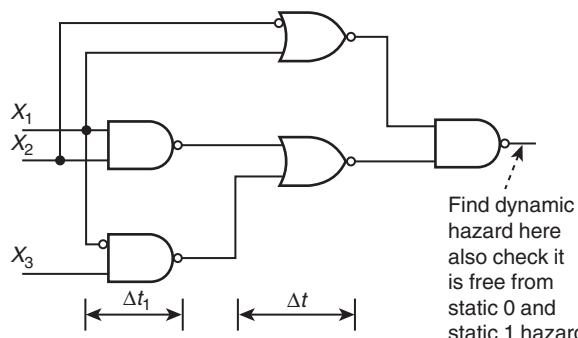


FIGURE 10.11 Circuit for Exercise 12 to find the static 0, static 1, dynamic and essential hazards.

■ QUESTIONS

1. How do the races differ from the static hazards?
2. What is a static 0 hazard? What are the conditions for arising of a static 0 hazard?

10.22 Digital Systems: Principles and Design

3. How do the races differ from the dynamic hazards?
4. What is a static 1 hazard? What are the conditions for arising of a static 1 hazard?
5. What is a dynamic hazard? What are the conditions for arising of a dynamic hazard?
6. Using a Karnaugh map, how do we detect static 0 hazard in three variable cases and in four variable cases and also find the missing circuit? (Hint in four variable case, use quads in place of diads).
7. What are the steps followed in detecting static hazards?
8. What is the method of detecting essential hazards?
9. How do you eliminate static and dynamic hazards?
10. How do you eliminate essential hazards? (Hint: Additional delays in feed-back elements or use of pulse mode sequential circuit).

CHAPTER 11

Implementation of Combinational Logic by Standard ICs and Programmable ROM Memories

OBJECTIVE

We learnt in earlier chapters that a combinational circuit has following characteristics:

1. These have n inputs and m outputs. For examples, (i) A NOT (inverter) gate, $n = 1$ and $m = 1$. (ii) An XOR gate, $n = 2$ and $m = 2$. (iii) An 8-bit adder, which accepts in its inputs a carry bit, 8 bits of X and 8 bits of Y and results at outputs the 8 bit sum Z and final carry CY , $n = 17$ and $m = 9$.
2. Logic state, at any of the outputs in the combinational circuits, depends only on the inputs at any given instant (not considering always present propagation delay period) and is not correlated at all with any of its previous outputs or outputs.
3. A truth table of a combinational circuit gives the values of all the m outputs for each possible combination of the n inputs and has 2^n rows and $(n + m)$ columns.
4. A combination circuit can be designed for obtaining the m outputs using the m Boolean expressions.
5. Each expression can be represented by SOP or POS format. A Karnaugh map-based technique or computer-based minimization technique is used to get least cost (minimum number of gates) or least delay (minimum number of levels between inputs and the corresponding output).

We learnt that the minimized circuit is then implemented easily using AND-OR arrays or NANDs or OR-AND arrays or NORs.

Alternatively, a logic design implements by decoders, encoders, multiplexers or demultiplexers, or binary arithmetic adders, adder/subtractors, code converters, comparator, bit-wise 8-bit AND, OR, XOR circuits or parity generators. Standard ICs are commercially available. We will learn the standard ICs and PROMS that are used for these in this chapter.

A combinational circuit may be complex enough to assemble using standard ICs. We wish to design a circuit for a character in a line printer, which gives a 64-bit output for each pixel in the character when the ASCII code of that is given as the input (refer section 5.1.3 for ASCII codes). These 64-bits are input to the printer head pins driving circuit. It means that we need a combination circuit for each ASCII character which has $n = 8$ (7-bit ASCII code + one parity bit) and $m = 64$. Another example is of a LCD line display or multi line display circuit, that has further complexity. Another example is of an advertisement displayed by an array or matrix of LEDs.

The implementation of the complex circuits with standard ICs can be extremely bulky in many situations. How do we then design such a complex circuit? One method nowadays for a complex combinational circuit is using a programmed or programmable logic memories, ROM or EPROM or EEPROM. Another is by using programmable logic devices PALs, PLAs, GALs and FPGAs.

We will learn the ROM, EPROM and EEPROM memories and implementation of complex logic circuits using these in this chapter. Other method of using PALs and PLAs will be described in Chapter 12 and FPGAs in Chapter 14.

11.1 STANDARD ICS FOR DESIGN IMPLEMENTATION

We have learnt in chapters 4 and 5 the binary arithmetic circuits, decoders, encoders, multiplexers, code converters, digital comparator for magnitude and equality, parity generators and checkers and bit wise AND, OR, NOT logic processing circuits. Standard ICs are available for these circuits. The combinational circuits are made using these ICs.

11.1.1 Adder/Subtractor IC and Magnitude Comparator

CMOS based 74HC 83 and 7483 family ICs are the four bits full adders. Figure 11.1 shows the IC 7483 connections when using it as an adder/subtractor.

MSI IC 7485 (or its HCMOS version 74HC85) is a 4-bit magnitude comparator. It also as well incorporates all the requirements shown in Figure 5.5. Its block diagram was described in Figure 5.6. Figure 11.2 shows the IC 7485.

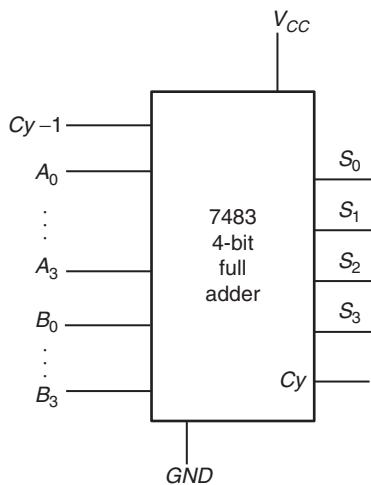


FIGURE 11.1 IC 7483 connections and its use as an adder/subtractor.

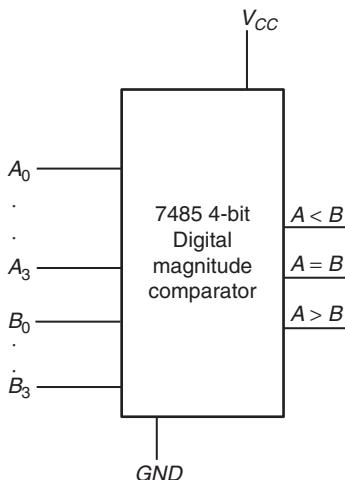


FIGURE 11.2 Digital comparator IC 7485.

11.1.2 Decoder IC

Figure 4.5(a) showed 3-line to 8-line (1 of 8) decoder with three control pins \overline{G}_1 , \overline{G}_2 , and G_3 . The \overline{G}_1 , and \overline{G}_2 control pins are activated by 0 and the G_3 pin is activated by 1. A MSI chip 74138 has the pin corresponding to the decoder shown in the figure. Figure 11.3(a) shows an exemplary connection in 74138.

11.1.3 Encoder IC

MSI IC74148 is an 8 to 3 encoder with an active 0 gate-enable for the inputs and an active 0 gate-enable pin for the outputs. Figure 11.3(b) shows connections using 74148.

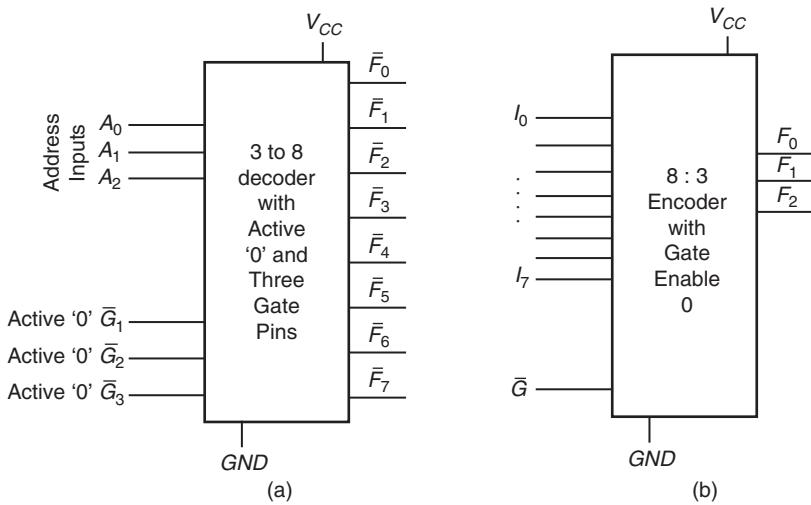


FIGURE 11.3 (a) Connections in using 74138 (b) Connections in using 74148.

11.1.4 Multiplexer IC

Figure 4.10 showed a multiplexer IC. The MSI IC74156 is a four-channel multiplexer. It selects the inputs I_0 , I_1 , I_2 and I_3 as per the channel selector lines A_0 and A_1 . Figure 11.4 shows the connections using 74156. The stroke pin is MUX enable pin.

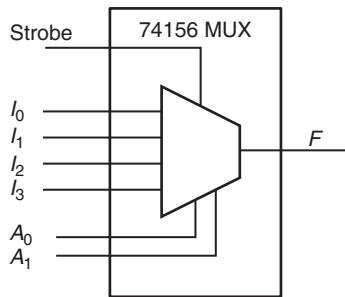


FIGURE 11.4 Connections using a 74156 multiplexer IC.

11.2 PROGRAMMING AND PROGRAMMABLE LOGIC MEMORIES

11.2.1 ROM (Pre-Programmed Read Only Memory) and PROM (Programmable Read Only Memory)

A read only memory, abbreviated as ROM or read only memory, is also a ready to use set of combinational circuits. A ROM is a previously programmed ‘Decoder-OR’ array based logic device using appropriate masks at the manufacturing stage.

Points to Remember

- (i) A decoder with one-line output decoder is also a minterm or maxterm generator, and
- (ii) a decoder can be used to implement a Boolean expression each by using an OR gate with those decoder outputs, which happens to be the minterms of the expression. If given Boolean variables at the address input points, then a programmed ROM implements a combinational circuit, each with an OR gate at the output.

ROM special versions programmable logic devices (PLDs) called PROMs (Programmable Read Only Memories) [An EPROM or an E²PROM or flash or OTP ROM] is programmable at the laboratory scale].

The ROMs do not change or lose its programmed data upon an interrupt of power to it. (This feature is also called non-volatility of a ROM). Non-volatility is a most important feature in ROM and PROM that is useful in a computer system, where the ROMs or EPROMs or E²PROMs store the frequently needed programs and data sets for the system users or the truth tables for implementing the combinational circuits. Let us consider applications, such as a preprogrammed toy circuit, a preprogrammed robot circuit, a standard look up table, or an arithmetic function table generator, a user defined code generator, a character generator, a printable or displayable fonts table, a set of the data or a set of instructions or an arithmetic function table generator. The instructions, table, and /or the data) must remain stored in the memory in these applications, even after a power interruption or power switched OFF. A ROM or PROM is used for it.

Figure 11.5 shows block diagram of a ROM with n address inputs, one select input and m data bit outputs and one read input [control gate (enabling) input]. As diodes between 1 and 2, and 5 and 6 are masked, when the address inputs are all 0s then Y_0 activates and $D_0 = 0, D_1 = 1, D_2 = 0, D_3 = 1$. The output is 1010. The masks are as programmed before manufacturing the chip.

Points to Remember

Each ROM or PROM has n inputs, called address bits (for one of the 2^n memory locations). This ROM on activating an input, called read input (actually a control gate input), generates for each set of address inputs, a distinct set of m outputs, called data bits (most often $m = 8$) from the addressed memory location. The data bits are as per programmed bits during manufacturing of ROM by silicon masking or any other processing. When we use a ROM for the implementation of the combinational circuits, the Boolean variables are the inputs at the address input pins.

11.2.1.1 An Exemplary 3 × 2 ROM

A ROM has an address decoder (n to 2^n decoder) within it. Figure 11.6(a) shows a 3 × 2 ROM block diagram.

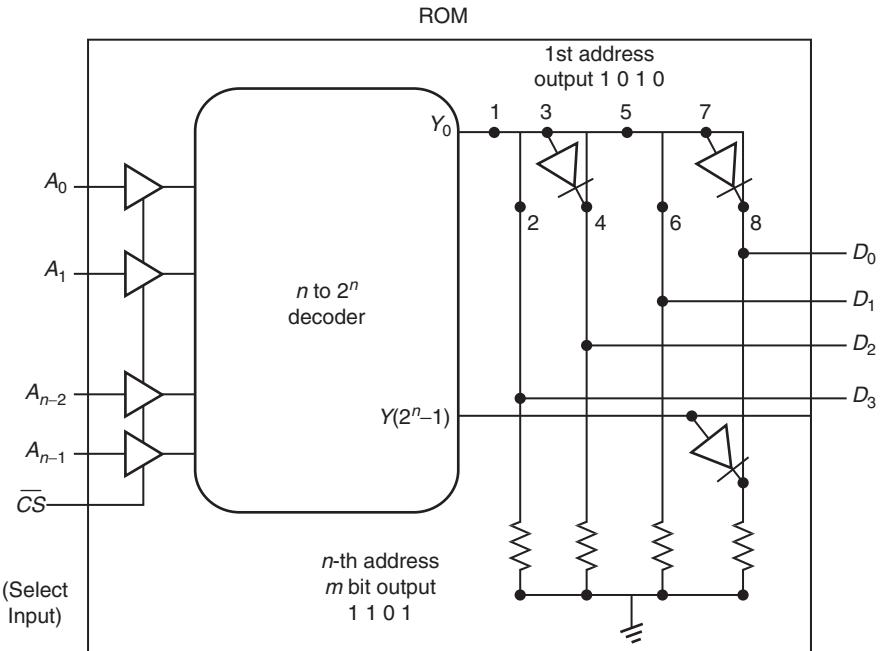


FIGURE 11.5 Block diagram of a ROM with n address inputs, one select input and m outputs.

1. The decoder thus consists of $2^3 = 8$ memory locations (outputs) and thus implements 8 combination circuits. A decoder output is active 1 when a minterm happens to be present in the Boolean expression for the given input address bits, A_2, A_1 and A_0 .
2. The 8 outputs of the decoder are given to two number eight-input OR gates to obtain two outputs ($m = 2$), D_0 and D_1 .
3. Total 16 fusible links exist with 8 fuses per OR gate. Fusing the links generates the outputs as per Boolean expression and its truth table.

Figure 11.6(b) shows truth table, which the 3×2 programmed ROM implements.

Figure 11.6(c) shows a 3×2 ROM programmed ROM, which has the fused (snapped) links to implement the truth table of Figure 11.6(b) by appropriate masking at the manufacturing stage.

Characteristic of a ROM circuit of Figure 11.6(c) is that just apply the address bits A_2, A_1 and A_0 and activate \overline{RD} input and we get the output data bits D_0 and D_1 as per the truth table logic from the row as per the address bits. When $\overline{RD} = 1$, the D_0 and D_1 are in tristates.

Figure 11.6 shows a simpler representation of fuse links at the multi-input OR gate(s). This representation has become conventional for the fuse links at the multi input AND or OR in case of the AND-OR or OR-AND arrays. In place of many input lines to an AND or OR, a single line is shown with each dot representing a separate line, which is limited.

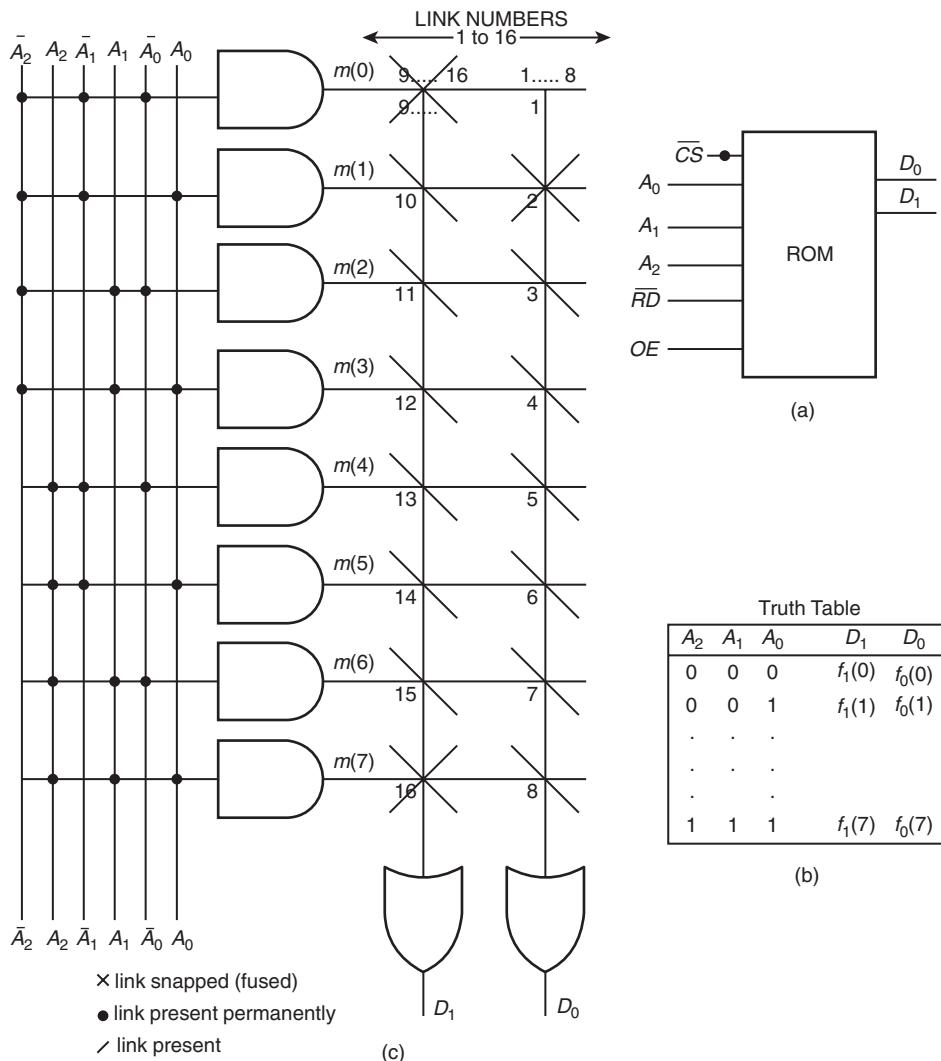


FIGURE 11.6 (a) Block diagram of a 3×2 ROM (b) Truth table to be implemented by the 3×2 programmed ROM (c) 3×2 ROM programmed ROM, which has the fused (snapped) junctions to implement the truth table. A simpler representation of multi-input OR gate with fuse links is used here.

11.2.1.2 An Exemplary $4k \times 8$ ROM (4 kB ROM)

A $4k \times 8$ ROM has $m = 8$, eight data bits at the output. It is also called 4 kB ROM (B is for a byte). Data bits are $D_0, D_1, D_2, \dots, D_7$.

4k means $(4 \cdot 1024)$ addresses. [With reference a memory unit, 1 k is conventionally taken as 1024 ($=2^{10}$)]. $2^n = 4 \cdot 1024$. There $n = 12$ and address bits will be $A_0, A_1, A_2, \dots, A_{11}$. $4k \times 8$ ROM implements 4096 combinational circuits, each with 8 outputs and 12 Boolean variables as the inputs. The outputs are as per the programming designed according to a truth table used during manufacturing.

ROM Programming means selecting the appropriate fusible links, which are snapped in the programmed ROM.

11.2.1.3 ROM Special Versions for Laboratory Scale Programming [EPROM, E²PROM, flash and OTP ROM]

A PROM [EPROM, an E²PROM and an OTP ROM] is the user programmable logic device (called EPLD, electrically programmable logic device).

An EPROM is a special ROM. It can be erased as well as programmed in a laboratory. A user can program and thereby store the data at the various referencable address locations. A special unit called an EPROM programmer does the programming. Programming at a selected address is done with the help of a 12.5 V or 25 V supply at one of its input, and then at another input, a programming pulse of duration say 50 ms or less. The EPROM then stores the data bits at the selected address. These bits can be read when this EPROM is used in another circuit. *Read* means obtaining the outputs corresponding to a selected address. These are the inputs to other circuits like a processor, which reads these.

The programmer can program all the 2^n addresses. The programmed data bits then have erase immunity from any power interruption or power OFF. The EPROM can now only be erased through a UV light of, say, a 12 mW/cm^2 intense UV source at 2.5 cm above the quartz window on the EPROM IC chip. The IC is exposed for a period of about 20 m.

The laboratory erase and programming facilities are not available in case of ROM, unlike an EPROM because a ROM has to be programmed at its manufacturing stage itself.

Figure 11.7 shows an EPROM model diagram for a $2k \times 8$ IC chip 2716.

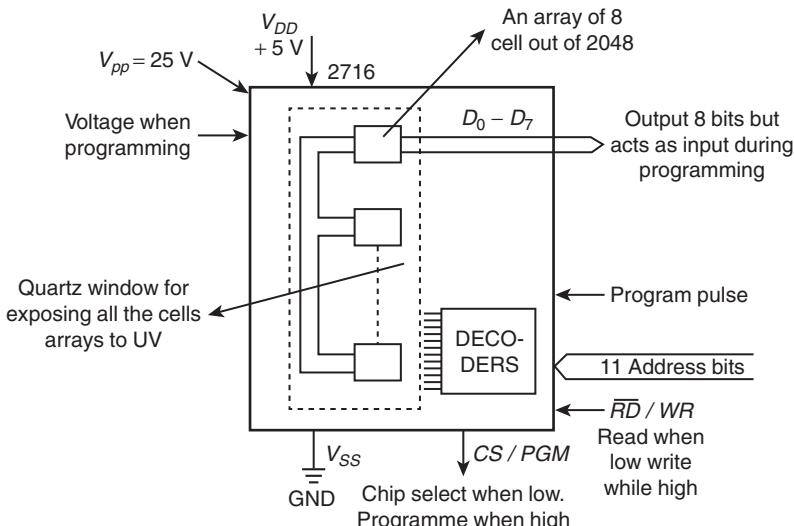


FIGURE 11.7 An EPROM model diagram for an IC chip $2k \times 8$ 2716 showing the address inputs, one select input and m outputs and other pins.

- (1) There is an additional pin in the EPROM that should be applied a specified voltage, V_{pp} (typical values are +25 V, +21 V or +12.5 V) during programming (writing) of data bits (byte) at a selected address location inside it.
- (2) A pin, \overline{CS}/PGM , when at 1, enables programming of this EPROM through a pulse for a period of 50 ms to another pin.
- (3) The \overline{CS}/PGM pin, when at 0, enables the chip selection and the inputs and outputs, D_0 to D_{m-1} , and disables the programming in order to permit only a reading of the D bits from a select address.
- (4) The \overline{RD} pin, when at 1, makes the outputs, D_0 to D_{m-1} in tristates and disables the programming in order to permit only a reading of the D bits from a select address.

An EPROM laboratory programmer unit performs the following steps in a sequence:

1. Applies gives the n bits (address bits) to the decoder of the array of cells and applies as inputs the D bits, which are meant for the outputs later on during the read operation corresponding to the selected address.
2. Applies a high voltage to make programming feasible and applies a very short duration (as per EPLD specification) to cause fusing (snapping) of the desired links in the array due to the high voltage.
3. Repeat from the step 1 by applying the next higher address than the previous one.
4. Repeat till all addressed are programmed.
5. Verify the programmed bits.

During read operation, the V_{DD} and V_{SS} are the supply inputs in EPROM chip and are at 5 V and GND, respectively.

An IC E²PROM can be erased as well as programmed like (2k × 8) 2816 without an exposure to UV light. We call such EEPROM; electrically erasable and programmable read only memory i.e. E²PROM. These E²PROMs, in case possess a short access as well as programming time and has full erasibility in one cycle then are called flash memories, for example, an E²PROM IC 28F256 is a flash memory. (F in an IC number of a ROM denotes a flash memory).

E²PROM is erasable and programmable above 10000 (in latest E²PROMs) times by the electrical means. Erase of a byte is by writing by 1111 1111 at D_0 to D_7 inputs from some external circuit. Before programming, the 0s must be erased by replacing them by 1s. Programming of a byte is by writing appropriate bits (for example 1000 1111 for writing 1000 1111) by the successive D_0 to D_7 inputs and the corresponding address inputs one by one from some external circuit (processor or laboratory programmer). Erase is byte by byte in E²PROM and full sector wise in flash.

Flash is also erasable and programmable above 10000 times (in latest flash memories) by the electrical means. Erase of a whole sector of bytes (in a flash) is by writing by 1111 1111 at D_0 to D_7 inputs from some external circuit. Before programming, the 0s must be erased by replacing them by 1s. Programming of a byte is by writing by appropriate bits (for example 1000 1111 for writing 1000 1111) by successive D_0 to D_7 inputs and the corresponding address inputs one by one from some external circuit (processor or laboratory programmer). Flash can have a sector reserved to work as the OTP.

11.10 Digital Systems: Principles and Design

A certain E²PROM chip, in case is one time programmable then it is called an OTP. An OTP does not have a quartz window. It stores data, which is unmodifiable by a user except once by an application of a high voltage pulse. (An OTP is like an electronic paper onto which writing is done by permanent ink).

■ EXAMPLES

Example 11.1 Show how will we design an eight-bit adder using a four-bit adder IC 7483.

Solution

We will cascade two adders by an adder IC for lower nibble addition given input carry = 0 and next upper nibble adder given the next stage carry of lower nibble as the input carry for its 0th stage. Figure 11.8 shows the circuit for implementing an eight-bit adder using two numbers four-bit adder IC 7483s.

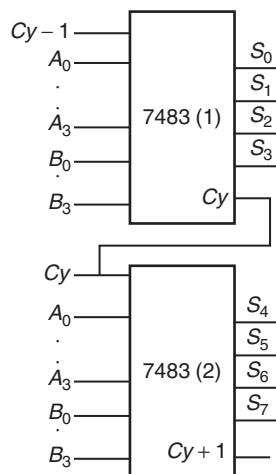


FIGURE 11.8 Circuit for implementing an eight-bit adder using two numbers four-bit adder IC 7483s.

Example 11.2 Show how will we verify using a digital comparator whether outputs from the adder circuit in Example 11.1 gives all bits 0s.

Solution

We will give the eight bits of Figure 11.8 adder to a comparator IC 7485. The inputs are as shown in the Figure 11.9. Comparison will be done with other eight inputs for B as shown there. The output $ZF_1 = 1$ shows all 7 bits of A as 0s, else output will be 0.

Example 11.3 Show how will we design a decoder tree for 6 six encoded inputs $A_0 \dots A_5$ using 74138.

Solution

Figure 11.10 shows a decoder tree for six encoded inputs $A_0 \dots A_5$ using 74138.

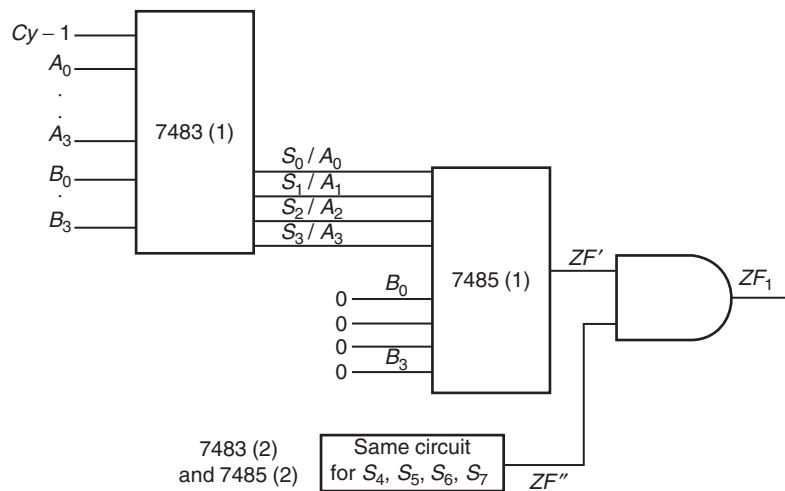


FIGURE 11.9 Two digital comparators to find whether outputs from the adder circuit have all 7 bits as 0s.

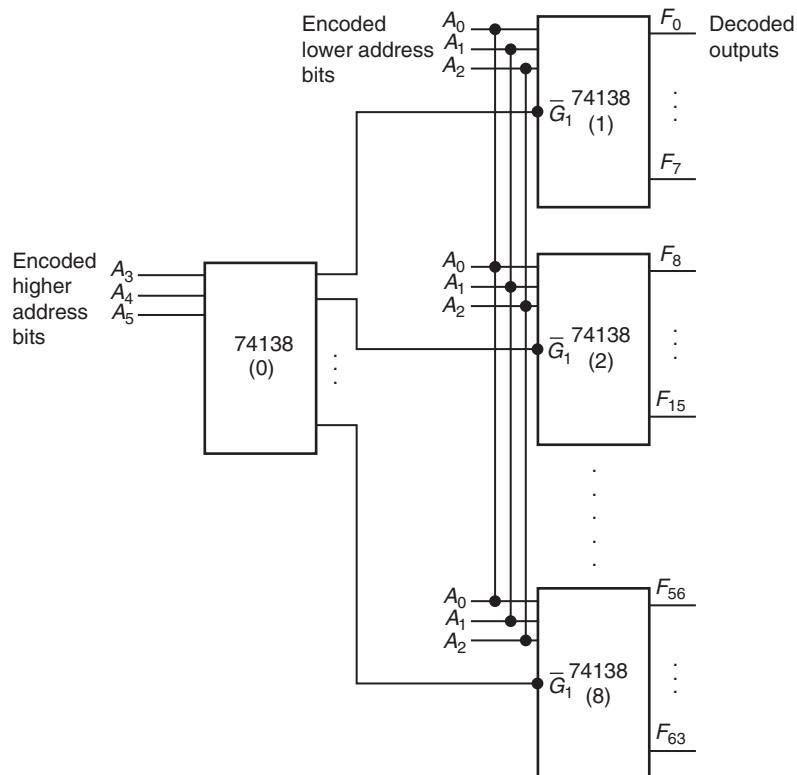


FIGURE 11.10 Decoder tree using 9 ICs 74138 for six encoded inputs using 74138. G_2 and G_3 are 0 and 1 in each 74138.

11.12 Digital Systems: Principles and Design

Example 11.4

Show how will we design a multiplexer tree for 16 inputs using 74156.

Solution

Figure 11.11 shows a multiplexer tree for 16 inputs using 74156.

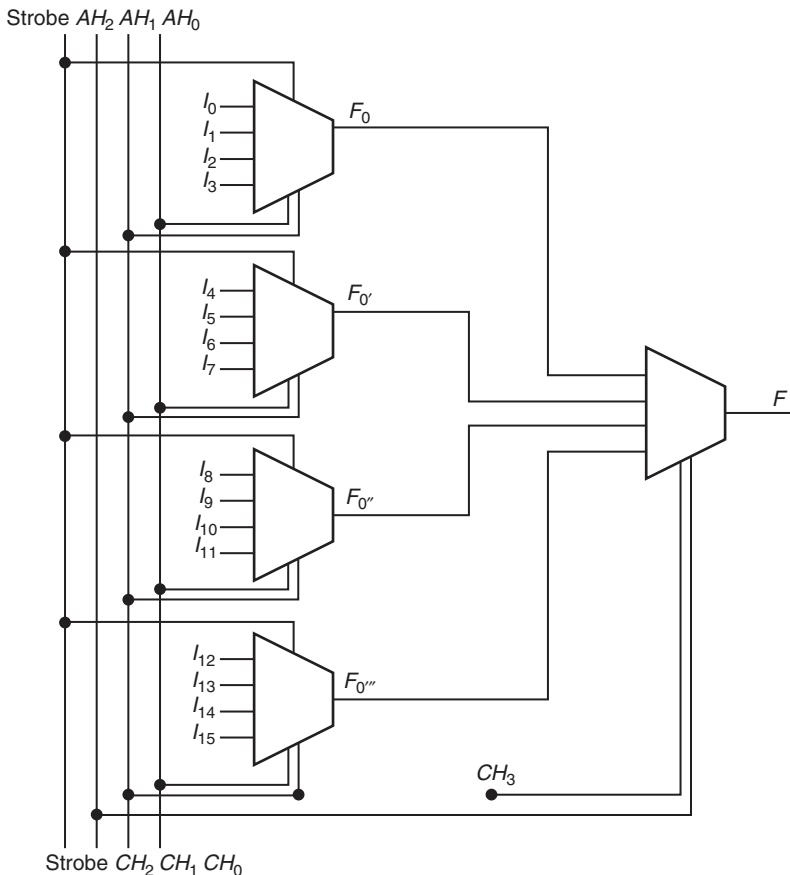


FIGURE 11.11 Multiplexer tree for 16 inputs using 74156D.

Example 11.5

Show how will we design a BCD to Gray code converter using 74156.

Solution

Figure 11.12 shows a BCD to Gray code converter using the 74156.

Example 11.6

Show how to program the fusible links to get a 4-bit Gray code from the binary inputs using a PROM.

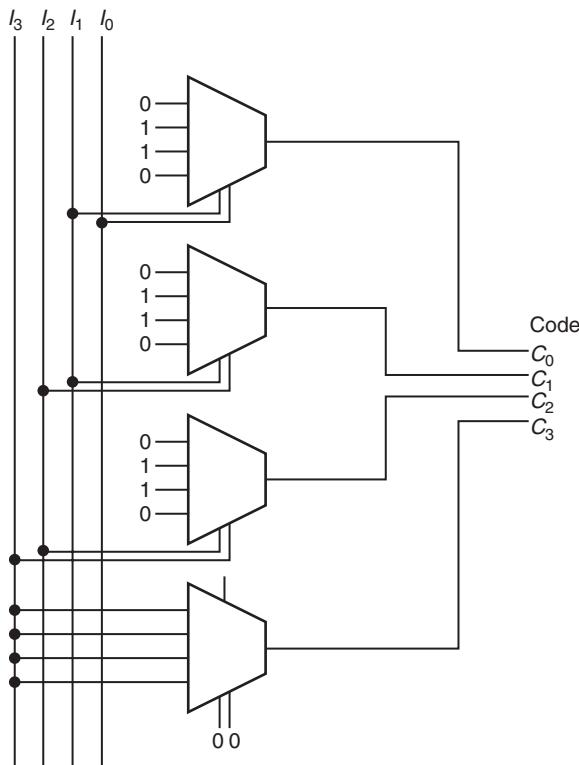


FIGURE 11.12 BCD to Gray code converter using the 74156.

Solution

Step 1: Finding the PROM address bits and data bits and number of fused links needed:

Four OR gates are needed to get a four bit Gray code. There are four binary inputs. There will be $2^4 = 16$ AND gates in the decoder. We need 16×4 PROM. There will be 64 fusible links, 1 to 64. (Figure 11.13(a)). Let $OR_0, OR_1; OR_2$ and OR_3 give the outputs D_0, D_1, D_2 and D_3 , respectively.

Step 2: Finding the fused links:

Table 11.1 gives the address inputs and Gray code for each set of inputs. Now, let us number these starting from $OR_0 m(0)$ taken as 1 to $OR_0 m(15)$ taken as 16, $OR_1 m(0)$ taken as 17 to $OR_1 m(15)$ taken as 32, $OR_2 m(0)$ taken as 33 to $OR_2 m(15)$ taken as 48 and $OR_3 m(0)$ taken as 49 to $OR_0 m(15)$ taken as 64. Link snaps at place where there are 0s in bits in column 5 (Gray codes). Figure 11.13(b) shows the solution for OR_0 and OR_1 .

Example 11.7

Show how to program the fusible links to get the four Boolean expressions correspond to the given sets of four combinational circuits implemented using PROM.

Solution

$$F_0 = \Sigma m(0, 1, 4, 5); F_1 = \Sigma m(1, 3, 5, 7); F_2 = \Sigma m(0, 2, 4, 6); F_3 = \Sigma m(0)$$

11.14 Digital Systems: Principles and Design

4 INPUT VARIABLES

$\bar{A}_3 A_3 \bar{A}_2 A_2 \bar{A}_1 A_1 \bar{A}_0 A_0$

LINK Numbers

1 to 64

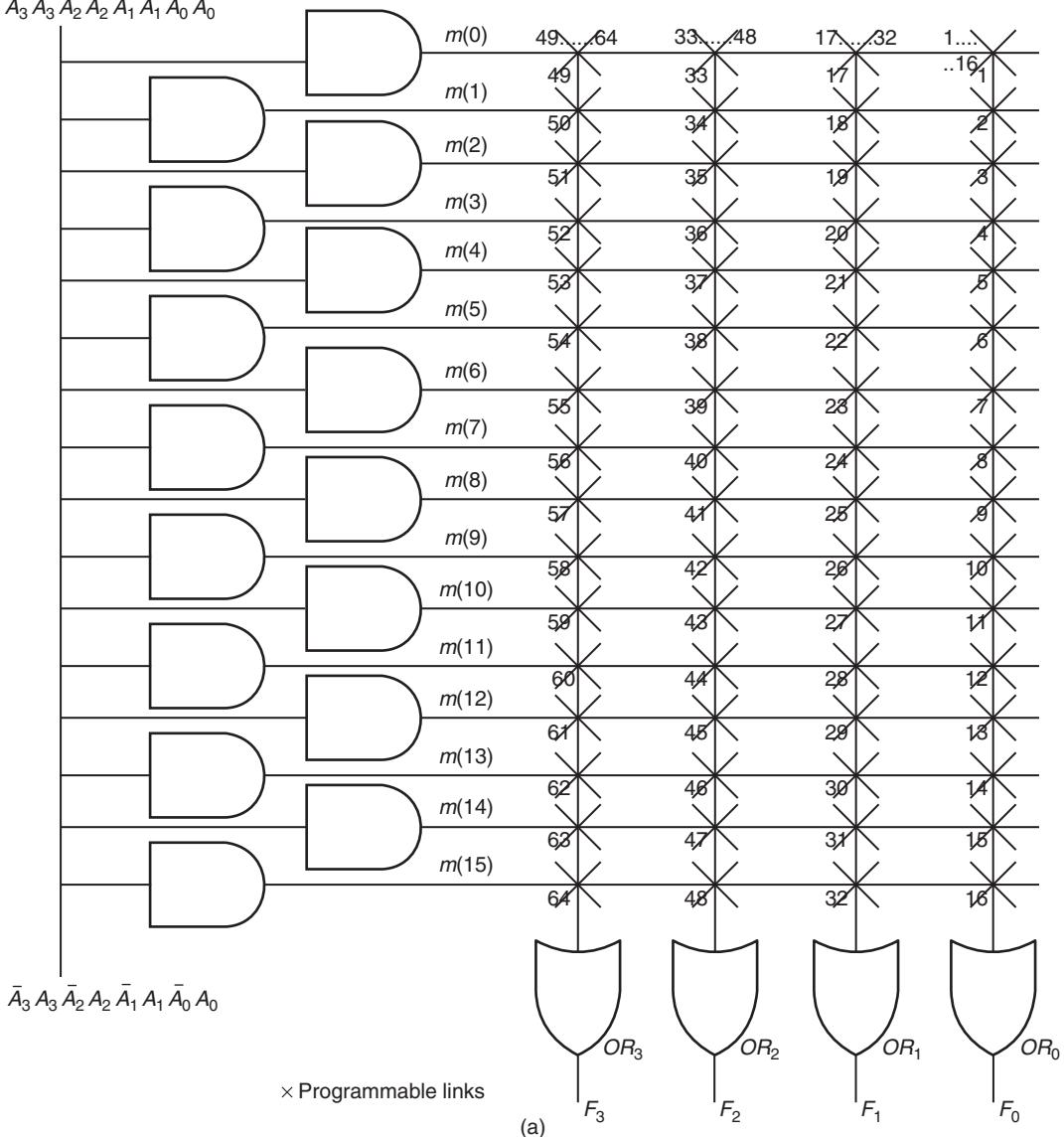


FIGURE 11.13 (a) 64 fusible links, 1 to 64 in a 16 × 4 PROM (b) Snapped links for OR_0 and OR_1 outputs F_0 and F_1 as per D_0 and D_1 in column 5 Table 11.1.

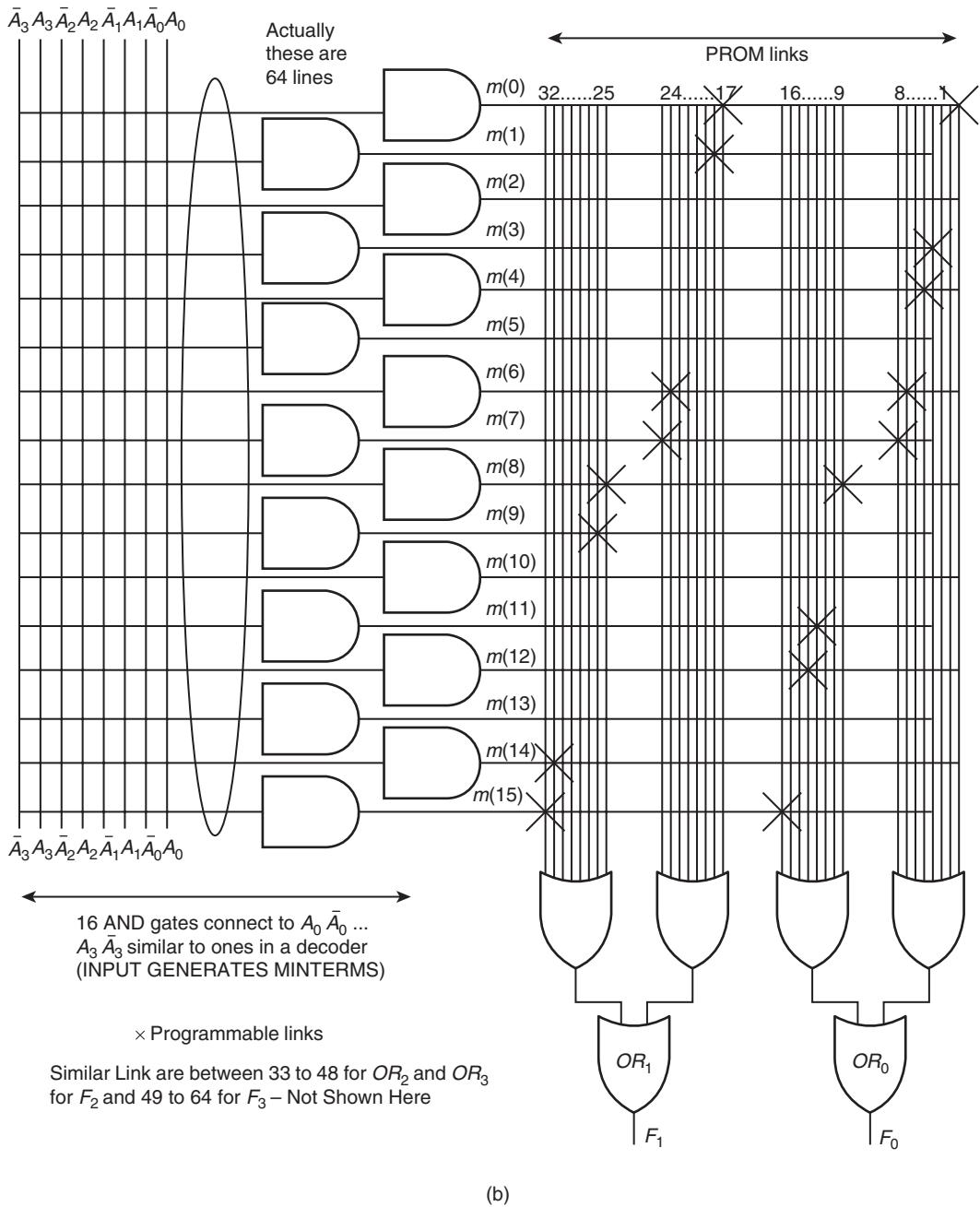


FIGURE 11.13 Contd.

11.16 Digital Systems: Principles and Design

TABLE 11.1 Snapped link numbers corresponding to four gray code bits from solution in Example 5.4 for the Karnaugh map

Address inputs				Gray code	Snapped link numbers			
A_3	A_2	A_1	A_0	$D_3D_2D_1D_0$	OR_3	OR_2	OR_1	OR_0
0	0	0	0	0000 _{gray}	49	33	17	1
0	0	0	1	0001 _{gray}	50	34	18	
0	0	1	0	0011 _{gray}	51	35		
0	0	1	1	0010 _{gray}	52	36		4
0	1	0	0	0110 _{gray}	53			5
0	1	0	1	0111 _{gray}	54			
0	1	1	0	0101 _{gray}	55		23	
0	1	1	1	0100 _{gray}	56		24	8
1	0	0	0	1100 _{gray}			25	9
1	0	0	1	1101 _{gray}			26	
1	0	1	0	1111 _{gray}				
1	0	1	1	1110 _{gray}				12
1	1	0	0	1010 _{gray}		45		13
1	1	0	1	1011 _{gray}		46		
1	1	1	0	1001 _{gray}		47	31	
1	1	1	1	1000 _{gray}		48	32	16

Step 1: Finding the PROM address bits and data bits and number of fused links needed:

Four OR gates are needed to obtain F_0 , F_1 , F_2 and F_3 outputs. There are three inputs, because maximum minterm given is $m(7)$. There will be eight AND gates in the decoder. We need 8×4 PROM. There will be 32 fusible links, 1 to 32 in it. (Figure 11.14).

Step 2: Finding the fused links needed:

Let OR_0 , OR_1 , OR_2 and OR_3 give the outputs F_0 , F_1 , F_2 and F_3 , respectively (Table 11.2). Now, let us number the links starting from $OR_0 m(0)$ as 1 to $OR_0 m(7)$ as 8, $OR_1 m(0)$ taken as 9 to $OR_1 m(7)$ taken as 16, $OR_2 m(0)$ taken as 17 to $OR_2 m(7)$ taken as 24, and $OR_3 m(0)$ taken as 25 to $OR_3 m(7)$ taken as 32 in Figure 11.14. Link snaps at places corresponding to the 0s (shown by a \times sign) in column for the outputs at the Table 11.2. Link is present at places shown by the dot.

Example 11.8

Take an $8 \text{ mm} \times 8 \text{ mm}$ area on a centimeter graph paper and put points with a pencil so that these points correspond to a character A . Show how to design and program the fusible links to get the 8 successive set of outputs to print ‘A’ using a PROM.

Solution

Step 1: Finding the PROM address bits and data bits and number of fused links needed:

(i) There are three inputs, because maximum minterm is $m(7)$. There will be eight AND gates in the decoder. (eight rows in the truth table). (ii) Eight OR gates are needed for 8 pixels at a row. We need 8×8 PROM. There will be 64 fusible

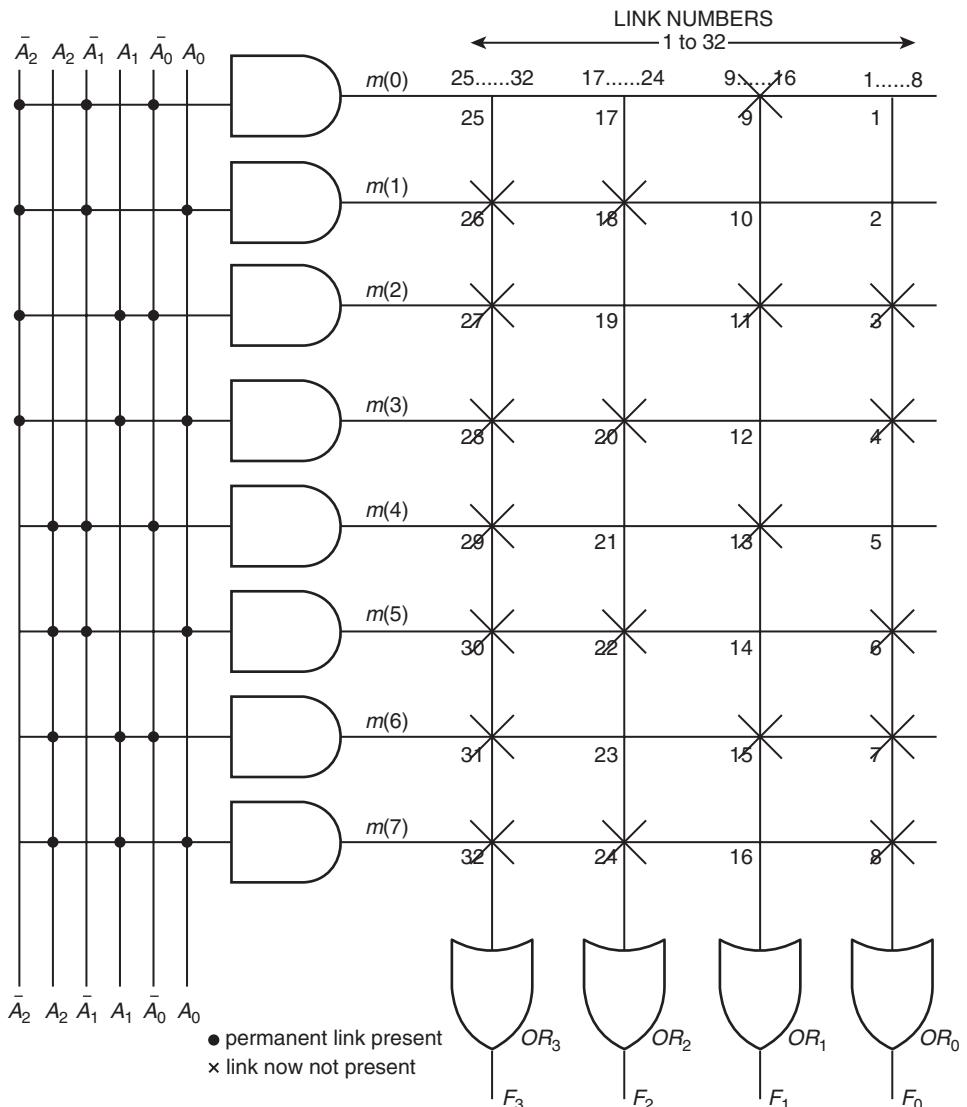


FIGURE 11.14 Implementation using 8×4 PROM four Boolean expressions $F_0 = \sum m(0,1,4,5)$; $F_1 = \sum m(1,3,5,7)$; $F_2 = \sum m(0,2,4,6)$; $F_3 = \sum m(0)$ for four combinational circuits.

links, numbered 1 to 64 in it. Let OR_0, OR_1, OR_2, \dots and OR_7 give the outputs F_0, F_1, F_2, \dots and F_7 , respectively, to drive the dots in the matrix at the print head.

Step 2: Finding the Fused Links:

Table 11.3 gives relative address (address with respect to a base address) at which 8 rows pixel data stores as per column 2 to 9. Now, let us number these starting from $OR_0 m(0)$ taken as 1 to $OR_0 m(7)$ taken as 8, $OR_1 m(0)$ taken as 9 to

TABLE 11.2

Address inputs			Outputs	Present link numbers			
A_2	A_1	A_0	$F_3 F_2 F_1 F_0$	OR_3	OR_2	OR_1	OR_0
0	0	0	1 1 0 1		25	17	1
0	0	1	0 0 1 1			10	2
0	1	0	0 1 1 0		19		
0	1	1	0 0 1 0			12	
1	0	0	0 1 0 1		21		5
1	0	1	0 0 1 1			14	6
1	1	0	0 1 0 0		23		
1	1	1	0 0 1 0			16	

Note: F_3 is 1 for $m(0)$ only. So fuse-link number 25 remains as such. F_2 is 1 for $m(0)$, $m(2)$, $m(4)$ and $m(6)$. Therefore, 17, 19, 21 and 23 fuse-links are not snapped. Similar is the method for determining the links for F_2 and F_1 .

TABLE 11.3

Relative address	Eight × Eight pixels for 'A'			Eight bit (8 ORs) links needed							
0	*			33							
1	* * *			42							
2	* * *			51							
3	* * * * * * *			60							
4	* * * * * * *			61	53	45	37	29	21	13	
5	* * * * *			62							14
6	* * * *			63							15
7	* * * *			64							16

$OR_1 m(7)$ taken as 16, $OR_2 m(0)$ taken as 17 to $OR_3 m(7)$ taken as 24 and so on up to 64 for $OR_7 m(7)$. A Link is present at place where there is a 0 in column 2 to 9 in Table 11.3.

Figure showing the fused (snapped) links can be easily drawn using Figure 11.13(a) in a identical way shown in Figures 11.13(a) and 11.14. Drawing of the figure is left as exercise to the reader.

Example 11.9

Consider the seven bits of ASCII codes for the alphanumeric characters and assume 8th parity bit = 0. Show how to design and program the fusible links to get the eight successive sets of outputs to display 'Welcome' using a PROM using sixteen segment character displays.

Solution

Step 1: Finding the PROM address bits and data bits requirements:

- (i) Given are the number of inputs = 8 (seven bit code + one bit parity).
- (ii) Since 'Welcome' has characters W and m, which can't be displayed by seven-segment unit, a sixteen-segment unit is needed for each character. [Figure 11.15(a) shows a sixteen segment unit with segments labeled from a to p as well as 0 to 15]. For sixteen-segment unit, sixteen outputs are needed. Therefore, an 8×16 PROM

is required. [Note: Figure 11.15 (b) also shows sixteen-segment units in an array of 8 units to display ‘Welcome’. Figure 11.15(c) shows an array of 16 units to display a full line message. Figure 11.15(d) shows a (16×4) 64-unit matrix to display four lines, 16 characters, each using the PROM outputs.

Step 2: Finding the fused links:

Let base address b be the starting address in PROM for the 256 addresses for placing the ASCII table. Table 11.4 gives the relative address r (address with respect to the base address at which the segments for a character in the ‘Welcome’ selects among the sixteen segments. If links are not fused (snapped) at the places shown then the segments shown in column 10 display. Column 10 gives segment numbers, which equal the [fuse Link numbers with respect to (relative address + base address)*16] for the eight rows for the characters of ‘Welcome’. If segment number = 2 then corresponding fuse link is $(b + r)$, if 3 then $(b + r + 1)$, if 3 then $(b + r + 2)$ and so on up to $(b + r + 15)$. From the segment numbers we find the fuse link numbers for the fuses not to be snapped.

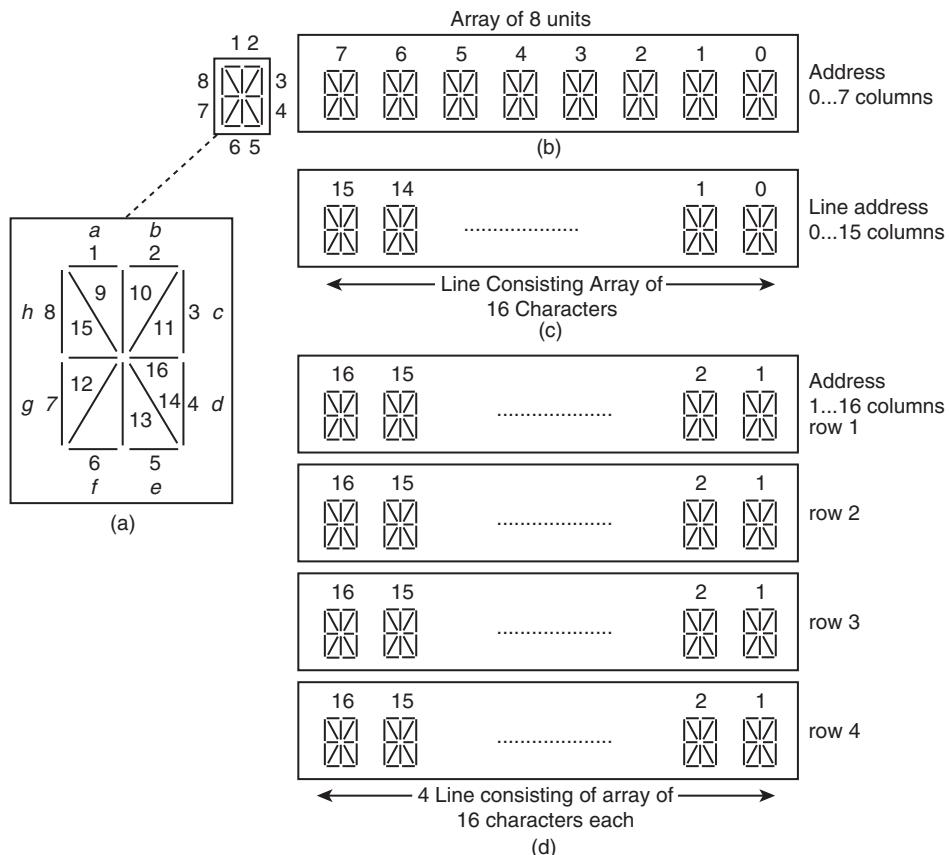


FIGURE 11.15 (a) Sixteen-segment unit with segment numbers labeled from a to p as well as 0 to 15 (b) An array of 8 units to display ‘Welcome’ (c) An array of 16 units to display a line (d) A matrix of 64 units to display four lines, 16 characters, each.

TABLE 11.4

Relative address in ASCII table	Character	ASCII coded eight inputs for eight characters welcome								Segment numbers = {fuse link numbers with respect to the (Relative address + Base address) $\times 16$ } for 8 rows for eight characters welcome
87	W	0	1	0	1	0	1	1	1	8, 7, 12, 14, 4, 3
101	e	0	1	1	0	0	1	0	1	15, 10, 1, 8, 7, 6
108	/	0	1	1	0	1	1	0	0	10, 13
99	c	0	1	1	0	0	0	1	1	1, 8, 7, 6
111	o	0	1	1	0	1	1	1	1	15, 7, 6, 13
109	m	0	1	1	0	1	1	0	1	7, 15, 13, 16, 4
101	e	0	1	1	0	0	1	0	1	15, 10, 1, 8, 7, 6
32	Space	0	0	1	0	0	0	0	0	None

Example 11.10 How will you design a circuit for continuously incremental display from 000 to 999 at regular intervals?

Solution

Figure 11.16 shows a circuit using PROM to display continuously 000 to 999 sequentially using three numeric display units, nd_0 , nd_1 and nd_2 , each of 7 segments. A seven segment display unit (Figure 10.16(a)) displays decimal numbers 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 depending on the 7 inputs bits, O_0 , O_1 , O_2 , O_3 , O_4 , O_5 , O_6 , and O_7 as (i) as 1, 1, 1, 1, 1, 0 (ii) 0, 1, 1, 0, 0, 0 (iii) 1, 1, 0, 1, 1, 0, 1 (iv) 1, 1, 1, 1, 0, 0, 1 (v) 0, 1, 1, 0, 0, 1, 1 (vi) 1, 0, 1, 1, 0, 1, 1 (vii) 1, 0, 1, 1, 1, 1, 1 (viii) 1, 1, 1, 0, 0, 0, 0 (ix) 1, 1, 1, 1, 1, 1 and (x) 1, 1, 1, 1, 0, 1, 1, respectively. The unit has a common cathode interconnected to a bit, called \overline{EN} bit. This \overline{EN} bit, if at logic state 0 or 1 a display unit for a digit glows or not, respectively. At three consecutive addresses in the PROM, the details of the three bits corresponding to nd_0 , nd_1 , and nd_2 are stored. Three sequences are as follows. The first digit glows when $A_0 = 0$ and $D_7 = 1$. Second digit glows when $A_0 = 1$ and $D_7 = 1$. Third digit glows when $A_1 = 1$, $A_0 = 0$ and $D_7 = 1$.

At each address, the $D_0 \dots D_7$ bits corresponds to 7 segments O_0 , O_1 , O_2 , O_3 , O_4 , O_5 , O_6 , and O_7 , respectively, where $i = 1, 2, 3$. The total number of addresses, which are programmed in the sequential pattern, will be 3000 as there are 3 display units, and there are one thousand sequences to be displayed.

■ EXERCISES

- Recall the concept of lookahead carry for the fast processing of the carry. IC74283 has a faster processing of the C_y at the higher stages. Show the 74283 IC connections.
- Show how will you design an eight-bit adder cum subtractor using a four-bit adder IC 7483s and XOR ICs.

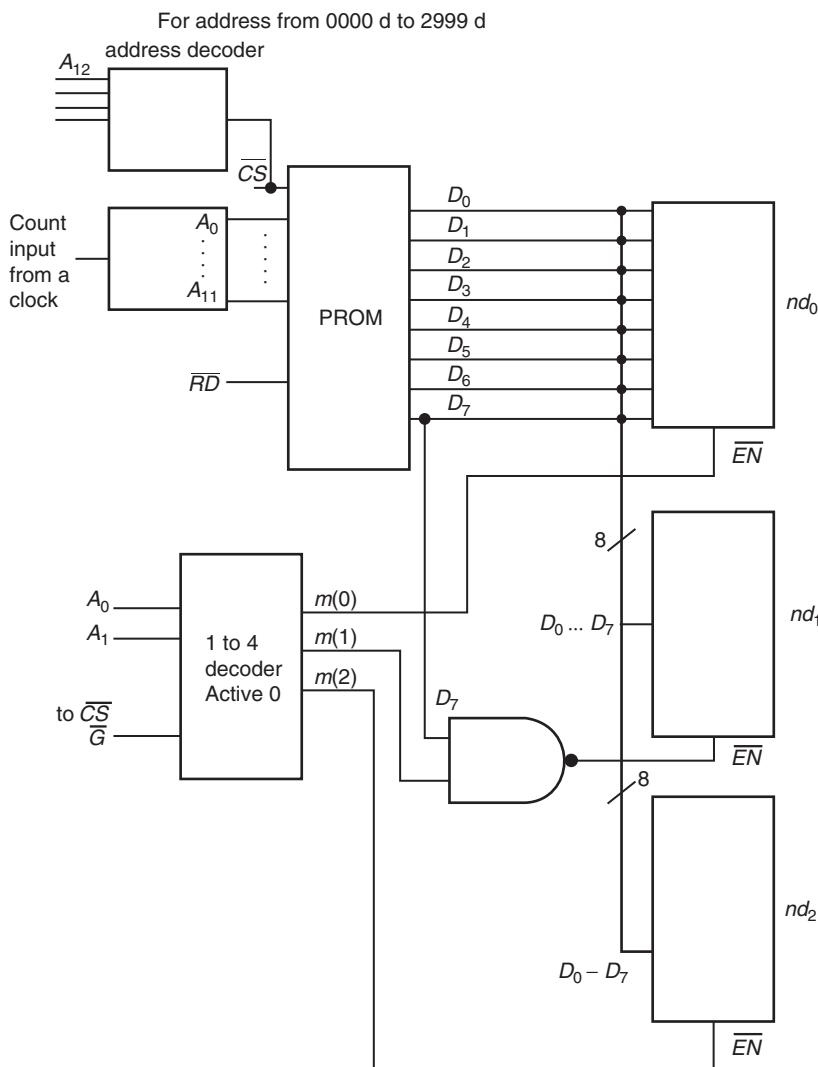


FIGURE 11.16 Circuit using $4k \times 8$ PROM with 3000 programmed addresses to display continuously 000 to 999 sequentially using three numeric display units, nd_0 , nd_1 , and nd_2 , each of seven segments.

3. Show how will we verify using a digital comparator whether the outputs from a subtractor circuit will generate carry.
4. Show how will we design a decoder to decode eight address lines using 74138.
5. Design using ICs an encoder for encoding inputs from the 16 keys.
6. Show how to program the fusible links to get a 4-bit Excess-3 Gray code from the binary inputs using a PROM.
7. Show how to program the fusible links to get the Boolean expression correspond to the given Karnaugh map implemented using PROM.

Three variable Karnaugh Map

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

8. Show how to program the fusible links to get the Boolean expression correspond to the given sets of four combinational circuits implemented using PROM.

$$F_0 = \Sigma m(1, 4); F_1 = \Sigma m(3, 5); F_2 = \Sigma m(7); F_3 = \Sigma m(2)$$

9. Show how to program the fusible links to get the four Boolean expressions correspond to the given sets of four combinational circuits implemented using PROM.

$$F_0 = \Sigma m(3, 4, 5); F_1 = \Sigma m(1, 3, 5); F_2 = \Sigma m(4, 6); F_3 = \Sigma m(7)$$

10. Take an $8\text{ mm} \times 8\text{ mm}$ area on a centimeter graph paper and put points with a pencil so that these points correspond to first character of your university's name. Show how to design and program the fusible links to get the eight successive set of outputs to print the name using a PROM.
11. Consider the seven bits of ASCII codes for the alphanumeric characters and assume 8th parity bit = 0. Show how to design and program the fusible links to get the eight successive sets of outputs to display 'Good Bye!' using a PROM using sixteen segment character displays.
12. How will you design a circuit for continuously decremented display from 00 to 99 at regular intervals?

■ QUESTIONS

1. List the ICs you will use for the arithmetic and logic operations. Give two exemplary circuits of each.
2. List the ICs you will use for the decoders, encoders, demultiplexers and multiplexers. Give two exemplary circuits of each.
3. What is a PROM? Describe four applications of PROMs as (a) implementer of Boolean expressions and combinational circuit, (b) memory unit for data and instructions of a program, (c) code converter and (d) display units driver.
4. Is a ROM or PROM not accessible randomly?
5. When do we use an EPROM, when a E²PROM, when a flash, when an OTP and when factory marked ROM?

6. What do we mean by the *erase* in EPROM, in flash and in E²PROM? (Hint: All bytes erase by UV, all bytes of a sector by writing all 1s, and single byte erase by writing 1s).
7. How many distinct addresses are programmed in a PROM, when we wish to drive the six numeric display units, each unit with 16 segments?
8. Why should it be that an active 0 two inputs 1 out of 4 address decoder is used to connect a common cathode of the numeric display units?

This page is intentionally left blank.

CHAPTER 12

Implementation of Combinational Logic by Programmable Logic Devices

OBJECTIVE

We learnt in Chapter 11 that instead of implementing a minimized circuit using AND-OR arrays or NANDs or OR-AND arrays or NORs or ICs for the decoders, encoders, multiplexers or demultiplexers, or binary arithmetic adders, adder/subtractors, code converters, comparator, bit-wise 8-bit AND, OR, XOR circuits or parity generators, a complex combinational circuit can be easily assembled by programmable logic memories (ROM, EPROM, EEPROM or Flash or OTP). In this chapter, we shall learn about two other forms of PLDs called PAL and PLA. We shall also study the implementation of the circuits using PAL and PLA.

12.1 BASICS POINTS TO REMEMBER WHEN USING THE PLDS (PROMS, PALs, PLAs)

For any logic function implemented by a combination circuit, there is a truth table. A row in a truth table can be specified as a minterm in a Boolean expression form, called sum of the products (SOPs) expression. There is one SOP expression each for each output column of the table. For example, consider a sum of the products expression for an output, Y .

$$Y = \bar{A} \cdot B \cdot \bar{C} + A \cdot B \cdot C \quad \dots(12.1)$$

12.2 Digital Systems: Principles and Design

This means that we have a truth table at which in an output column for the output Y the logic state=1 in those two rows where the inputs, (A, B, C), are $(0, 1, 0)$ and $(1, 1, 1)$, respectively. For the remaining 6 rows of the truth table, we have $Y=0$. A complemented variable at above expression, like \bar{A} , means its state is logic 0, and a variable without complement sign like B means state 1. Maximum number of rows in case of the three input variables is 8, and therefore equation (12.1) can have maximum eight terms on the right hand side. There can be maximum eight sums of product (SOP) terms with each product term having 3 variables, either as such or in its complemented form.

Similarly, consider another SOP expression for another truth table.

$$Y_0 = \bar{A} \cdot B \cdot \bar{C} \cdot \bar{D} + \bar{A} \cdot B \cdot C \cdot D + A \cdot \bar{B} \cdot \bar{C} \cdot \bar{D} \quad \dots(12.2a)$$

$$Y_1 = \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot D + \bar{A} \cdot B \cdot \bar{C} \cdot D \quad \dots(12.2b)$$

It means that $Y_0 = 1$ for the three rows of out of the 16 rows of the corresponding truth table for four inputs A, B, C and D . $Y = 0$ in all rows except the ones in which A, B, C and D are $0, 1, 0$ 01 and $0; 1, 1, 1$ and $1, 0, 0$ 0, respectively. Maximum number of rows in four input variables are there are 16 and therefore, two expressions (12.2a and b) can have maximum 16 terms each on the right hand side, since there can be maximum 16 sums of product terms with each product term having four variables either as such or in a complemented form. Each output, Y , in its sum of products form can be implemented by a circuit having the AND-OR arrays. This is explained as follows:

Let us consider a logic circuit in Figure 12.1. Figure 12.1 is for n inputs (2^n truth table rows) a matrix of the $2n$ -input AND gates and 2^n input m OR gates. ($2n$) input AND is used because of n variables and their n complements). Here, $n = 3$ and $m = 1$. Let us consider the array of six ANDs in the figure. Each AND has a distinct set of three plus three inputs each among A, B and C plus complements \bar{A}, \bar{B} and \bar{C} . The 6 inputs of the AND are interconnected as per 8 possible logic states of the 3 input variables in the Boolean expression.

A horizontal line known as product line represent a set of inputs for a truth table row or a minterm. Let a connection mean permanent link and intersection mean temporary fusible link. There can thus be 48 points of intersections or connections in the AND array inputs for six ANDs. An AND array output connects to an eight input-line OR (shown by single vertical line) through another eight connections or intersections.

In general for the n inputs, there are maximum 2^n numbers $2n$ -input AND gates (called AND plane) and m OR gates (called OR plane). A matrix of these AND-OR arrays have maximum 2^n rows and m columns. (The $n = 3$ and $m = 1$ in Figure 12.1).

At each intersection of the horizontal line (AND plane inputs) and vertical line (AND plane outputs), there can be a fusible link, which may be present or missing. The maximum number of intersections is equal to the number of elements in the matrix of 2^n rows and are in m columns. Maximum number of fusible links can be $(2n \cdot 2^n \cdot m)$. There can thus be $(2n \cdot 2^n \cdot m)$ links, which are to be programmed in general for a link present or link (missing, fused and snapped). (Figure 12.2 shows case of $2n \cdot l \cdot m$ links).

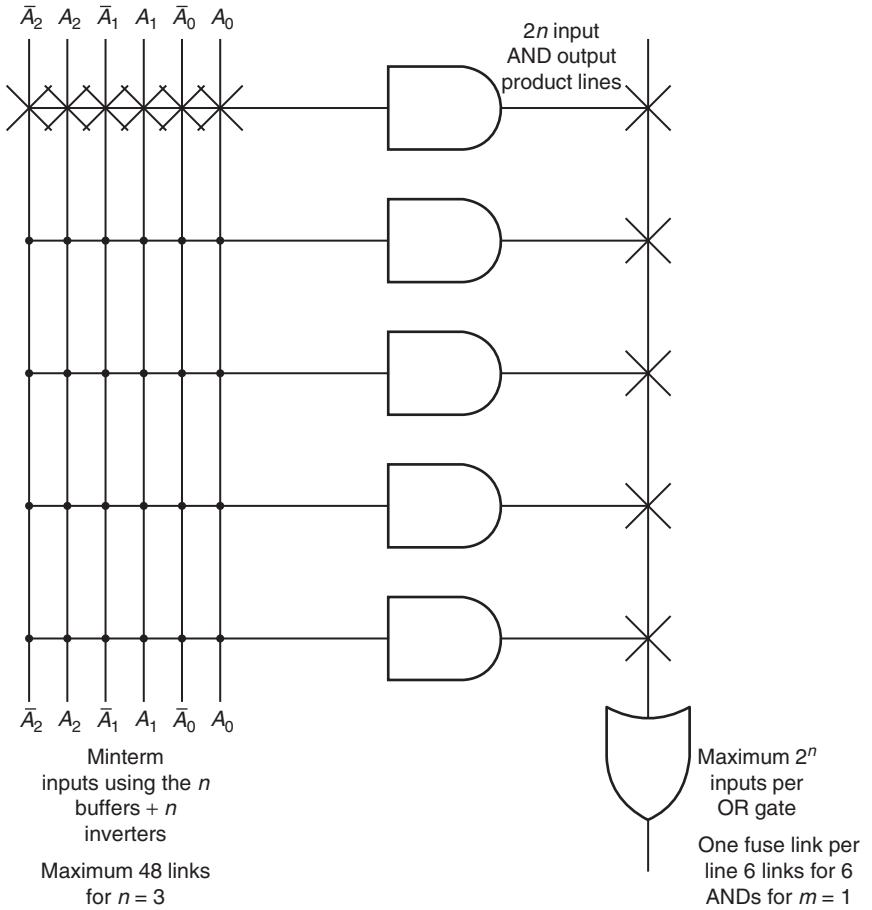


FIGURE 12.1 For the n inputs (2^n truth table rows) a matrix of $2n$ -input AND gates and m OR gates, when $n = 3$ and $m = 1$.

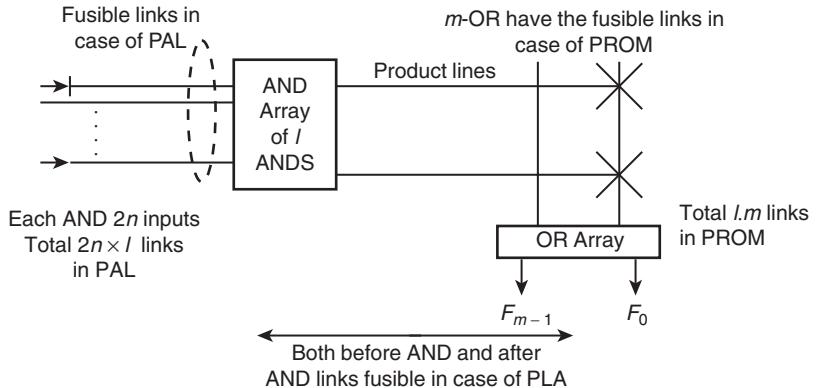


FIGURE 12.2 Circuit of the AND-OR arrays of l ANDs to implement any sum of the products expression for the case of n input variables with one output Y and the fusible links present for obtaining the F outputs as per required Y_0, \dots, Y_{m-1} when using a PAL or PROM of PLA.

Figure 12.3 shows circuit of AND-OR arrays to implement a sum of the products expression for the case of three input variables with one output Y .

Figure 12.3 also shows the intact links (by the crosses), which should be present in order to implement a function defined by SOP expression $C + \bar{B}\bar{C}$ and implement the logic states. The fused links for $Y_0 = C + \bar{B}\bar{C}$ are shown by the cross signs. In this expression, there are only two terms on right hand side. Therefore, there are only two present links to an eight input OR gate.

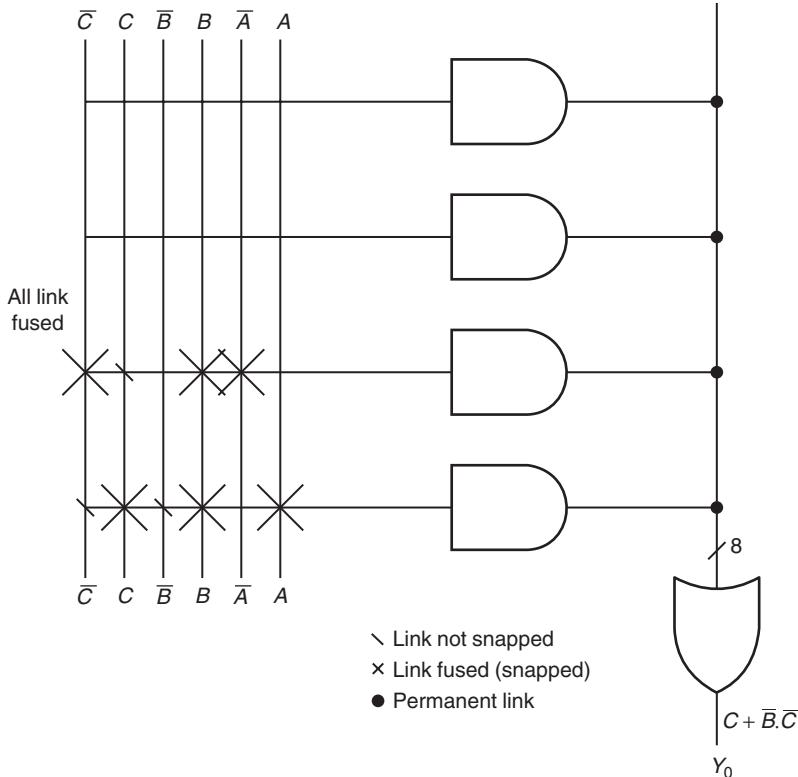


FIGURE 12.3 PAL circuit of AND-OR arrays to implement any sum of the products expression for the case of three input variables with one output Y_0 .

Figure 12.3 also shows the intact links (by the dots), which should be present in order to implement a function defined by SOP expression. In Figure 12.4 the present links at the AND arrays [in order to implement truth table function corresponding to the SOP expressions (12.2a and 2b)] are shown by the cross signs. In the expressions, there are only two and three terms on the right hand side for Y_1 and Y_0 . Therefore, there are only two and three present links to the 16 input OR gates are shown.

When n is large, this number becomes too big. Practically, the numbers of links are therefore limited in PROMs, PALs and PALs. PROM has only OR links and PAL only AND links. Figures 12.6, 12.13 and 12.14 shows applications of PROMs.

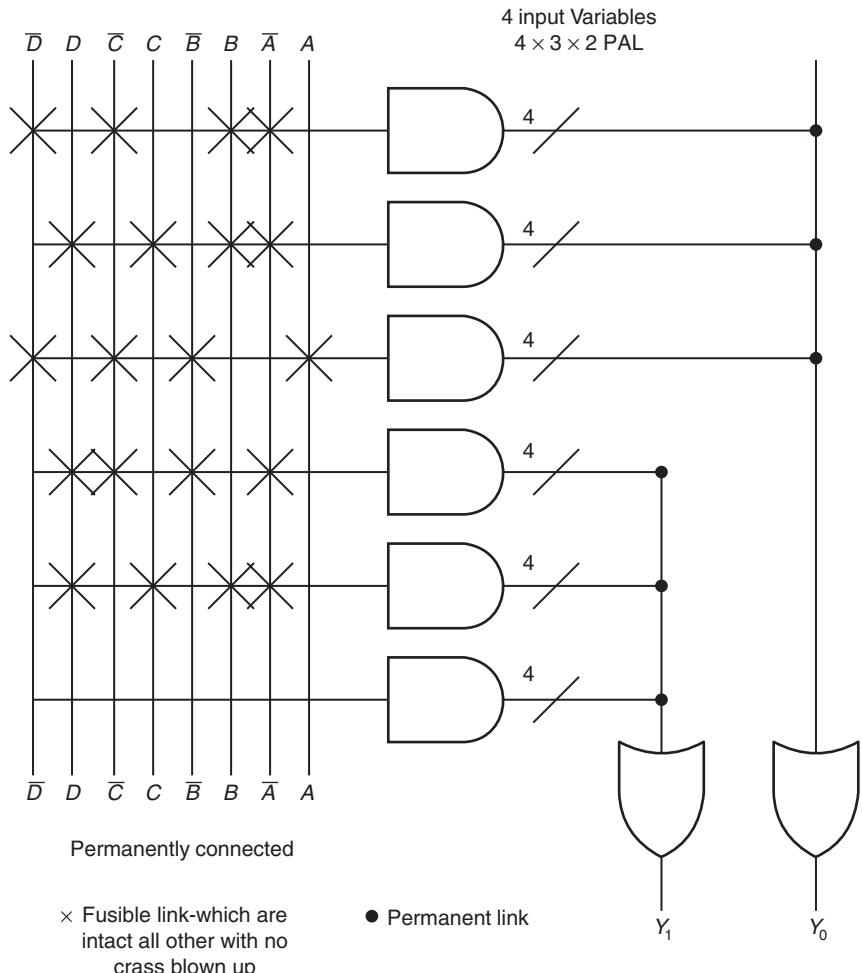


FIGURE 12.4 $(n \times l \times m)$ PAL circuit of AND-OR arrays to implement prime implicants at the SOP expressions with only $2n/l$ fusible links and generate m outputs.

Points to Remember for a PROM

1. Programmable ROM is a special programmable logic device (PLD) in which each input of the OR gate in the OR-arrays have the fusible links, which are fused as per the truth table or the Boolean expressions or Karnaugh maps requirements for the given set of outputs. (AND array inputs are not programmable in a PROM).
2. $(n \times m)$ PROMs have n input AND gates (2^n in number) and 2^n input m ORs. Each AND output corresponds to a minterm each in the SOP expression. Number of fusible links are $2^n \times m$ only in place of $2n \cdot 2^n \times m$ maximum possible.

3. An OR gate with no input given at one of its line is assumed to have input 0.
4. An AND gate with no input given at one of its line is assumed to have input 1.
5. Multiple n lines, each with a fusible link can be shown as a single line with the n fusible links.
6. A fusible link, which is not fused (snapped) if shown by a cross or slash sign at an intersection at the input from a previous stage output or from a source of input to the link, then the slash or cross sign is now equivalent of soldered link in the electrical circuit.
7. A fusible link that is fused (snapped) is shown by a missing cross sign at the intersections at the inputs from the previous stage outputs. A missing cross sign is equivalent of un-soldered (detached) link in the electrical circuit.
8. The AND gate or OR gate input line with all links fused with no cross sign anywhere is placed a cross sign in its center of AND or OR symbol, respectively.
9. A programming is a systemic hardware related procedure implemented by software at a programming system, called programmer or laboratory programmer or PLD programmer and the programming means executing the procedure for fusing (snapping) the needed links in the erased or fresh PLD.
10. Erased PLD means in which fusible links ready for programming.
11. Complete erasing occurs in EPROM by UV. Byte by byte erasing occurs in EEPROM and sector by sector in a flash memory by electrical means alone.
12. Programming of EEPROM or EPROM or flash is writing a byte one by one for fusing the appropriate links at the previously erased byte.

12.2 PAL (PROGRAMMABLE ARRAY LOGIC)

Each AND array corresponds to a SOP minterm or an implicant of Boolean expression. The output of each AND array is given to an m -input OR. An output of OR corresponds to output, Y . In general, there should be a 2^n input OR gate at each output variable, Y . However, in actual practice, these are limited as there are limited numbers of AND gates (only l number) used in the PALs. Let the number of AND gates be l . Each AND has $2n$ inputs. Therefore, the number of fusible links = $(2n.l)$.

PAL is a registered trade name of Monolithic Inc. USA, and Advanced Micro Devices. The OR links are fixed in a PAL, and are not programmable. Each OR gate has (l/m) fixed connections in the PALs. [When l is not an integer multiple of m , then one of the OR gate will have less number of OR gates and others $(l - 1)/(m - 1)$

or $(l - 2)/(m - 1)$ fixed connections. For example, if $p = 8$, $m = 3$, then two OR gates will have three each and one OR gate 2 fixed connections only.]

Programmable Array Logic (PAL) is a special programmable logic device (PLD) in which each input of the AND gate in the AND-arrays have a fusible link each, which is fused as per the minimum requirement after a suitable minimization and reduction procedure. The AND-array fusible links are fused as per the requirement of minimized form (prime implicants) after multi output minimization the number of the Boolean expressions (called multi output minimization) to be implemented.

Figure 12.4 showed a $(n \times l \times m)$ PAL A $(n \times l \times m)$ PAL has l number AND gates each with $2n$ inputs and each input has a fusible link and $n = 4$, $l = 6$ and $m = 2$. The fusible links number is just $(2.n.l)$ in place of $(2n.2^n.m)$ maximum possible and $(2^n.m)$ in PROMs.

A PAL needs to implement only the m' number of Boolean expressions by multi output minimization through a procedure the m number of Boolean expressions but generating the required m outputs. Reduction is by considering those expressions, which contain the expressions with a common set or complementary set of prime implicants.

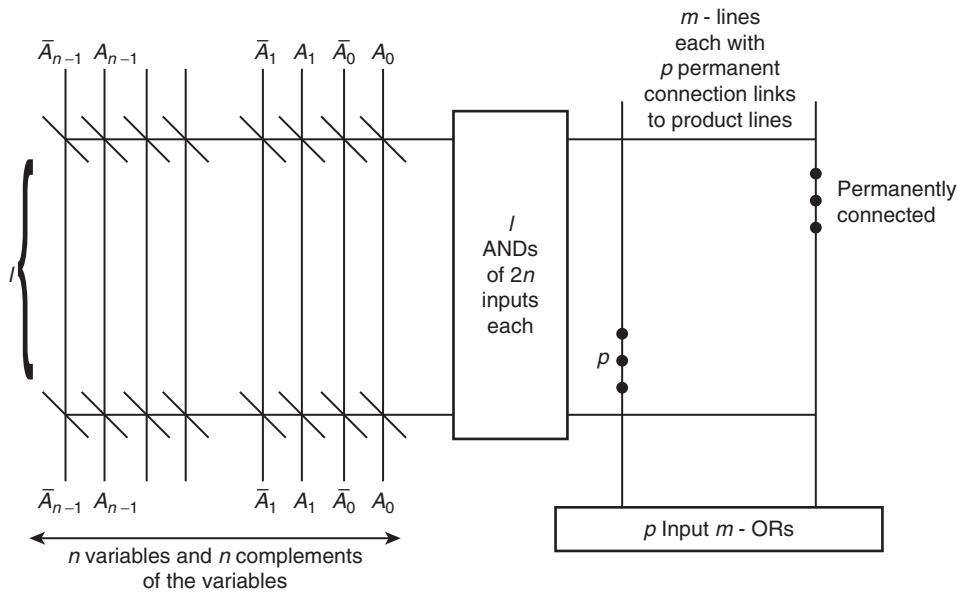
A unit called PAL programmer program a PAL. Using a PAL, any desired logic function to obtain an output bit Y can be implemented. A PAL is programmed like an E²ROM. A PAL is a general-purpose combinational circuit for the prime implicants of Boolean expressions. User needs to know only the truth table while using a PROM. For using a PAL, user need to know only the sets of the prime implicants of the logic functions needed to obtain the required outputs.

Why does a PAL have less number of ANDs l compared to 2^n in a PROM?

Reason is that usually $n = 10$ quite high, say 10 and there are computer based minimization procedures available, that makes it feasible to implement a combination logic circuit with prime implements only. Occurrence of common sets and complementary sets of prime implements in the Boolean expressions further reduce the requirements of the ANDs. So the PAL will have less cost (less number of gates) requirements and $l < 2^n$.

For example, a version of PAL called 16L8 contains eight ORs. Therefore, there are maximum eight outputs. Figure 12.5 shows its logic circuit. Each OR has seven inputs. There can be maximum 16 inputs in 16L8. The 16 inputs can be the Boolean variables and also the feedback of the ORs two outputs are dedicated and six outputs pins can be programmed so. There are 32 columns to link eight AND arrays. There are eight AND array as there are eight ORs. Each AND is an 8 inputs AND. Therefore, there are 64 rows total number of fusing links in 16L8 are, therefore, 2048 (=32 multiplied by 64) which are programmable by an external unit called PAL programmer. Inside this IC for PAL, between each OR output and an output pin, there is a tristate NOT meaning thereby, that the output pins are active 0s. An output pin represents function $Y = 1$ situation when at 0 and in standby (inactive) state, it is at 1.

An unregistered PAL acts like a combinational circuit. A registered PAL acts like a general-purpose sequential circuit (a circuit in which the past sequences

FIGURE 12.5 PAL 16L8 inputs and outputs and logic circuit. $m = 8$, $l = 7$, $n = 8$, $p = 7$.

(states) also matter). Instead of using discrete gates and FFs, we can use a registered PAL (FFs are described in next chapter). A registered PAL find applications in the state machines (the logic circuits generating the outputs states according to a sequence).

The PALs are less costly (fewer gates) than PROMs, give an advantage of smaller access times of ~ 30 ns or less compared to ~ 100 ns or more in PROMs. These devices simplify complicated circuits of the gates by at least five times or more. CMOS version of programmable logic devices (PLDs) like PALs have access times 0.015 μ s to 0.030 μ s more than the bipolar PLDs, but have smaller power consumption per PLD. The HCMOS (High Speed CMOS) versions are also available. An output from a PAL is, however, available within, typically, ~ 0.030 μ s in the HCMOS versions, and about 10 ns in bipolar transistor versions.

[Note: Electrically erasable PALs are also called field programmable PALs and are more costly. Masked PALs can be made like the masked ROMs at the manufacturing stages.]

Remember the Definition

PAL is a special programmable logic device (PLD) in which each input variable and its complement at an AND in the ANDs array have a fusible link, which is fused as per the prime implicants or the common and complementary sets of prime implicants and give the outputs as per the truth table for the combinational circuit.

Points to Remember

1. PAL is a special programmable logic device (PLD) in which each input and its complement at an AND has a fusible link. Total number of AND gates are limited in number to l with $l < 2^n$ maximum possible.
2. The AND-array fusible links are fused as per the requirement of minimized form (prime implicant form) after a multi output minimization of Boolean expressions for the outputs.
3. We implement m' number of Boolean expressions by multi output minimization of the m number of Boolean expressions. Reduction is by considering the expressions containing the common set or complementary set of prime implicants.
4. Inputs at AND array are programmable and inputs at OR array are not programmable in a PAL.
5. $(n \times l \times m)$ PAL has $2n$ input AND gates (l in number) and fixed input (number of inputs $< l$) m number OR gates. Each AND output corresponds to a prime implicant each in the SOP expression. Since number of ANDs are limited in number of fusible links 1 are $(2n.l)$ only in place of $(2n.2^n \cdot m)$ maximum possible.
6. At the AND inputs, a slash or cross sign shows a fused link, which is not snapped and none just an intersection shows a fused (snapped) link A dot shows the permanently connected link.
7. The AND gate, with all links fused with no sign anywhere, is placed a cross sign at the center of the AND symbol, respectively.
8. Programming is a systemic hardware related procedure implemented by software at a programming system, called programmer or laboratory programmer or PAL programmer and the programming means executing the procedure for fusing (snapping) the required links in a fresh (unprogrammed) PAL.
9. Electrical erasability in PAL is most often not available.

12.3 PLA (PROGRAMMABLE LOGIC ARRAYS)

Each AND array corresponds to a SOP minterm or implicant. The output of each AND array is given to an m -input OR. An output of OR corresponds to output, Y .

- (1) In general, there should be $(2.n)$ input 2^n number AND gates with OR gate for each output variable, Y . However, in actual practice, these are limited as there are limited numbers of AND gates used in the PLAs.
- (2) Similarly in general, there should be a 2^n input OR gate at each output variable, Y . However, in actual practice, these are limited as there are limited numbers 1 of AND gates used in the PLAs and therefore there are (l/m) input m number OR gates in the PLAs. $[(l/m)]$ if not an integer, then last OR gate will have $(l - 1)/m$ or $(l - 2)/m$ inputs and remaining $(l - 1)/(m - 1)$ or $(l - 2)/(m - 1)$ input OR gates with each OR gate having fusible links.

12.10 Digital Systems: Principles and Design

Let l is the number of AND gates present in a PLA. Each AND has $2n$ inputs. Therefore, the number of fusible links (like a PAL) is $(2n.l)$ at the AND plane of the PLAs. The number of fusible links is $l [= (l/m).m]$ at the OR plane of the PLAs. Therefore, the number of fusible links is $(2n.l^2)$.

OR links are also fusible in a PLA (like a PROM) and are programmable. Last OR gate has $(l - 1)/m$ or $(l - 2)/m$ inputs that have the fusible links and remaining $(l - 1)/(m - 1)$ or $(l - 2)/(m - 1)$ input OR gates with each OR gate having fusible links. For example, if $p = 8$, $m = 3$, then two OR gates will have three each and one OR gate two fusible links only.

Programmable Array Logic (PLA) is a special programmable logic device (PLD) in which each input of the AND gate array (like a PAL) as well as in the OR-array (like a PROM) have a fusible link each, which is fused as per the minimum requirement after a suitable minimization and reduction procedure. The AND-OR arrays fusible links are fused as per the requirement of minimized form (prime implicants) after multi output minimization of the number of the Boolean expressions.

Figure 12.6 shows a $(n \times l \times m)$ PLA. The PLA has l number of AND gates each with $2n$ inputs and each input and output of AND has the fusible links. The fusible links number is just $(2.n.l^2)$ in place of $(2n.2^n.m)$ maximum possible and $(2^n.m)$ in PROMs.

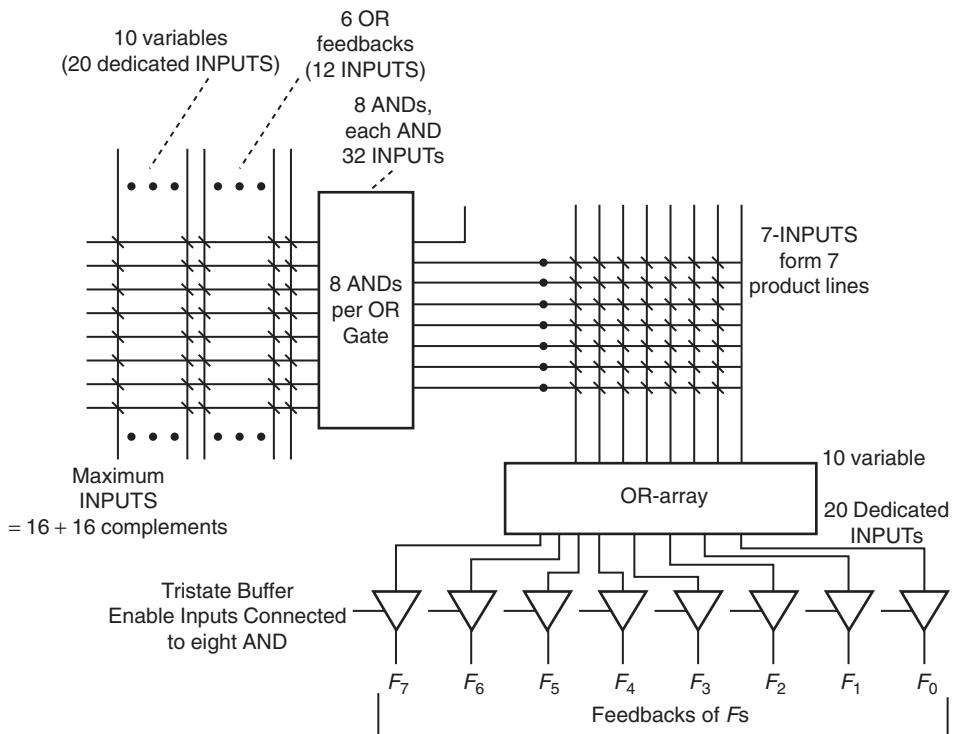


FIGURE 12.6 A $(n \times l \times m)$ PLA. A $(n \times l \times m)$ PLA has l number AND gates each with $2n$ inputs and each input has a fusible link ($l < 2^n$). ($l = 8$, $n = 10$ dedicated + 6 feed back = 16, $m = 7$).

PLAs are extremely suitable to implement the random or complex logic functions. If a PLA is made using bipolar transistors, as in a PAL, then the power and therefore current requirements are very high, typically 200 mA to 800 mA in a bipolar version. It is less than 200 mA in the CMOS versions of the PLAs.

A PLA needs to implement only the m'' number of Boolean expressions by multi output minimization through a procedure the m number of Boolean expressions but generating the required m outputs. Multi-output minimization is by considering prime implicants at the Boolean expressions and their products. If Y_1 and Y_2 are two Boolean expressions to be implemented then the problem is to find subset of the prime implicants that is common in Y_1 , in Y_2 and in $Y_1 \cdot Y_2$.

A unit called PLA programmer programs a PLA. Using a PLA, any desired logic functions to obtain the output bit Y_1, Y_2, Y_3, \dots can be implemented with minimal number of 'AND' and 'OR' gates. It has two types of inputs: One dedicated, which means directly connected logic inputs and other feedback, which means connected from the outputs of OR.

Why does a PLA have less number of ANDs l compared to 2^n in a PROM?

Reason is that usually $n =$ quite high, say 10 and there are computer based minimization procedures available for the individual Boolean expression minimization and multi output minimization. That makes it feasible to implement a combination logic circuit with fewer prime implement subsets only. So the PLA will have less cost (less number of gates) requirements and $l < 2^n$.

PLAs are extremely suitable to implement the random or complex logic functions. (Note: Electrically erasable PLAs are also called field programmable PALs and are more costly. Masked PLAs can be made like the masked ROMs at the manufacturing stages.)

Remember the Definition

PLA is a special programmable logic device (PLD) in which each input variable and its complement at the AND gate array as well as each input at the OR gate array have a fusible link, which is fused as per the subset of prime implicants needed to obtain the multi outputs realization after individual minimization and multi output minimization and circuit realization is as per the multiple outputs truth table for the combinational circuit.

Points to Remember

1. PLA is a special programmable logic device (PLD) in which each input, its complement at an AND gate and each gate output has a fusible link and total number of AND gates are limited in number to l with $l < 2^n$ maximum possible.
2. The AND-array and OR array fusible links are fused as per the requirement of minimized form (subsets of prime implicants) after multi output minimization of the number of the Boolean expressions to be implemented.

3. We implement m'' number of Boolean expressions by multi output minimization the m number of Boolean expressions. Minimization is by considering the expressions containing the common subset set or complementary set of prime implicants in the Boolean Expressions and their products.
4. AND array inputs are programmable and OR array inputs are also programmable in PLAs.
5. $(n \times l \times m)$ PLAs have $2n$ input AND gates (l in number) and fixed input (number of inputs $< l$) m number OR gates. Each AND output corresponds to a prime implicant each in the subsets of the SOP expressions and their products obtained after minimization. Since numbers of ANDs and ORs are the limited ones, the number of fusible links total is $(2n.l^2)$ only in place of $(2n.2^n.m)$ maximum possible.
6. At the AND inputs, a sign shows a fused link, which is not snapped and no sign just an intersection shows a fused (snapped) link A dot shows the permanently connected link.
7. The AND gate with all links fused with no sign anywhere is placed a cross sign in its center of AND symbol.
8. A programming is a systemic hardware related procedure implemented by software at a programming system, called programmer or laboratory programmer or PLA programmer and the programming means executing the procedure for fusing (snapping) the needed links in the fresh PLA.
9. Electrical erasing ability in PLA is most often not provided.
10. Programming of PLA means fusing the links after minimization and multi output minimization procedures, most often computer based.

In a general case of PLA, since both the AND and OR arrays are programmable, a PLA is more complex than a PAL or a PROM.

■ EXAMPLES

Example 12.1 Show how to program the fusible links to get a 4-bit Gray code from the binary inputs using a PAL. Also compare the design requirement with a PROM.

Solution

Refer to Example 12.6 for PROM based implementation of the Gray code converter. PAL is used when minimized Boolean expression(s) is available. Therefore, let us recall the Example 11.4. The expressions for code conversion are as follows:

Y_3 :

$$Y_3 = A \quad \dots(12.3)$$

Y_2 :

$$Y_2 = A \cdot \bar{B} + \bar{A} \cdot B \quad \dots(12.4)$$

 Y_1 :

$$Y_1 = B \cdot \bar{C} + C \cdot \bar{B} \quad \dots(12.5)$$

 Y_0 :

$$Y_0 = \bar{D} \cdot C + \bar{C} \cdot D = C \text{ XOR } D \quad \dots(12.6)$$

Step 1: Finding the PAL input bits and data bits and number of fused links to the ANDs that are needed.

Seven number eight-input (A, B, C and D) and ($\bar{A}, \bar{B}, \bar{C}$ and \bar{D}) ANDs are needed to get the four bit Gray code outputs Y_0, Y_1, Y_2 and Y_3 . This is because none of the seven prime implicants (one in Y_3 , and two each in Y_2, Y_1 and Y_0) has a common set in these four expressions. When we compare it with PROM, there were 16 AND gates, were needed for all the sixteen minterms (Figure 12.13(b)). We need 7×4 PAL in place of 16×4 PROM. There are $4 \times 2 \times 7 = 56$ fusible links compared to 64 fusible links in PROM. Let OR_0, OR_1, OR_2 and OR_3 give the outputs D_0, D_1, D_2 and D_3 , respectively. Let us number the fused links of AND0 as 1, 2, ..., 8 for $A, \bar{A}, B, \bar{B}, C, \bar{C}, D$ and \bar{D} . Let us number the fused links of AND1 as 9, 10, ..., 16 for $A, \bar{A}, B, \bar{B}, C, \bar{C}, D$ and \bar{D} . Similarly, we number up to 56 for AND6 \bar{D} link. (Figure 12.7 shows a demonstrative $4 \times 7 \times 4$ PAL).

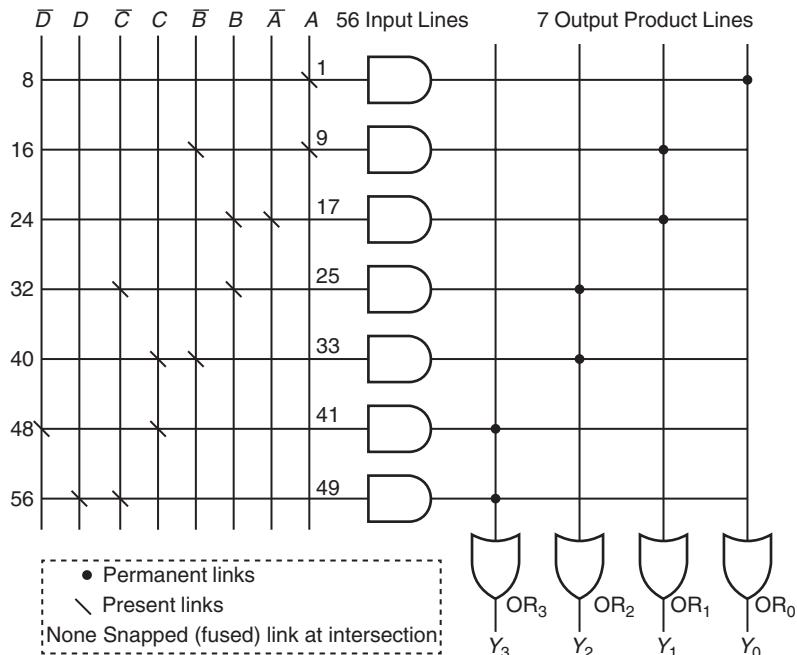


FIGURE 12.7 A demonstrative $4 \times 7 \times 4$ PAL that implements a Gray code converter ($n = 4, l = 7, m = 4$). Link numbers are also shown.

12.14 Digital Systems: Principles and Design

Step 2: Finding the PLA fused link numbers needed.

Table 12.1 gives the not fused fusible links to implement the Gray code. Figure 12.7 shows the links at the inputs of the ANDs and connections at the ORs that are present.

TABLE 12.1 Not fused links table

Bit	Not fused snapped link numbers							
	AND0	AAND1	AND2	AAND3	AND4	AAND5	AND6	AND7
Y_0	1							
Y_1			9, 12	18, 19				
Y_2					27, 30	36, 37		
Y_3							45, 48	55, 54

Example 12.2 Show how to program the fusible links to get the outputs as per Boolean expressions correspond to the given sets of combinational circuits implemented using PAL. Compare the number of fusible links and gates needed with respect to the PROM.

$$Y_0 = \Sigma m(2, 4, 5, 6, 7); Y_1 = \Sigma m(2, 6, 8, 9);$$

Solution

Recall Chapter 5. There are four variables as there are the terms $m(8)$ and $m(9)$ in Y_2 . Four variables Karnaugh Map (Tables 12.2) for maximum 16 minterms in an SOP expression is as follows:

TABLE 12.2 Sixteen minterms map

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

The Karnaugh map for Y_1 is therefore as in Table 12.3.

TABLE 12.3 Map for Y_1 for simplification

$A.B$	$C.D$	$\bar{C}.D$	$\bar{C}.D$	$\bar{C}.\bar{D}$	$C.\bar{D}$
	00	01	11	10	
$\bar{A}.\bar{B}$	00				1
$\bar{A}.B$	01	1	1	1	1
$A.B$	11				
$A.\bar{B}$	10				

$$Y_0 = \bar{A} \cdot B + \bar{A} \cdot C \cdot \bar{D} \quad \dots(12.7)$$

The Karnaugh map for Y_2 is as follows:

$A \cdot B \backslash C \cdot D$	$\bar{C} \cdot \bar{D}$ 00	$\bar{C} \cdot D$ 01	$C \cdot D$ 11	$C \cdot \bar{D}$ 10
$\bar{A} \cdot \bar{B}$ 00				1
$\bar{A} \cdot B$ 01				1
$A \cdot B$ 11				
$A \cdot \bar{B}$ 10	1	1		

$$Y_1 = A \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot C \cdot \bar{D} \quad \dots(12.8)$$

Step 1: Finding for the PAL number of fused links, ANDs and ORs needed.

We have a common subset of prime implicant, $\bar{A} \cdot C \cdot \bar{D}$. This can be taken as output Y_2 from an OR gate OR_2 . It can be implemented by one AND gate. The equations 12.7 and 12.8 have two other prime implicants. These can be implemented by two AND gates. Therefore, total three ANDs and three OR gates shall suffice. Let OR_0 , OR_1 and OR_2 and OR_3 give the outputs F_0 , F_1 and F_2 and F_0 give output Y_1 , F_1 give output Y_2 and F_2 give output Y_3 , respectively.

There are three feedback links also available at inputs E , F and G from OR_0 , OR_1 and OR_2 , respectively. Let us number the fused links of AND_0 , AND_1 up to AND_5 . The links are 1, 2, up to 14 for AND_0 A , \bar{A} , B , \bar{B} , C , \bar{C} , D , \bar{D} , E , \bar{E} , F , \bar{F} , G and \bar{G} . Let us number the fused links of AND_1 as 15, 16, ... 28 for A , \bar{A} , B , \bar{B} , C , \bar{C} , D , \bar{D} , E , \bar{E} , F , \bar{F} , G and \bar{G} . AND_2 link A up to AND_5 link \bar{G} are numbered from 29 up to 84. This PAL has four variables and three OR feedback input variables. OR_0 inputs permanently connect to AND_0 and AND_1 , OR_1 inputs to AND_2 and AND_3 and OR_2 inputs to AND_4 and AND_5 . (Figure 12.8 shows a demonstrative $7 \times 6 \times 3$ PAL).

Step 2: Finding the fused links needed.

Table 12.4 gives the PAL implementation. (i) Links 58, 61, 64 not fused (snapped) implements $\bar{A} \cdot C \cdot \bar{D}$. (ii) Links 2 and 3 implements $\bar{A} \cdot B$. (iii) Links 29, 32, 34 implements $A \cdot B \cdot \bar{C} \cdot Y_2 = G$ is feedback link and connects 55 and 27 links of AND_3 and AND_1 , respectively.

TABLE 12.4 Not fused links table

Bit	Not fused snapped link numbers					
	AND_0	AND_1	AND_2	AND_3	AND_4	AND_5
$F_0(Y_1)$	2, 3	27				
$F_1(Y_2)$			29, 32, 34	55		
$F_2(Y_3)$					58, 61, 64	

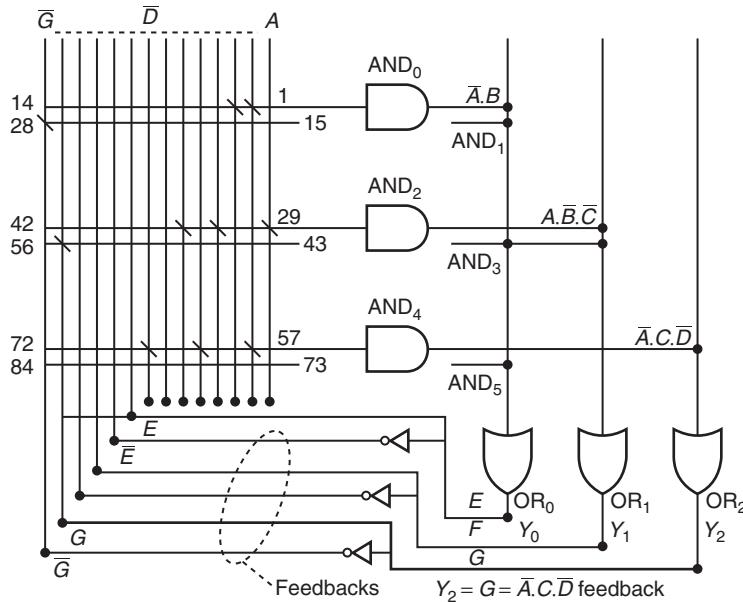


FIGURE 12.8 Demonstrative $7 \times 6 \times 3$ PAL with four input variables and three feedback input variables, six ANDs and three ORs ($n = 7$, $l = 6$, $m = 3$).

Example 12.3 Show how to program the fusible links to get a 4-bit Gray code from the binary inputs using a PLA. Also compare the design requirement with a PROM.

Solution

Refer to Example 12.6 for a PROM and Example 12.1 for PAL based implementation of the Gray code converter. PLA is used when minimized Boolean expression(s) is available, and both ANDs and ORs have fusible links. Therefore, let us recall the Example 5.4.

Y_3 :

$$Y_3 = A \quad \dots(12.9)$$

Y_2 :

$$Y_2 = A\bar{B} + \bar{A}B \quad \dots(12.10)$$

Y_1 :

$$Y_1 = B\bar{C} + C\bar{B} \quad \dots(12.11)$$

Y_0 :

$$Y_0 = \bar{D}C + \bar{C}D = C \text{ XOR } D \quad \dots(12.12)$$

Step 1: Finding the PLA input bits and data bits and number of fused links to the ANDs that are needed.

Seven number eight-input (A, B, C and D and $\bar{A}, \bar{B}, \bar{C}$ and \bar{D}) ANDs are needed to get the four bit Gray code outputs Y_0 , Y_1 , Y_2 and Y_3 . This is because none of the seven prime implicants (one in Y_3 and two each in Y_2 , Y_1 and Y_0) has a common set

in these four expressions we need seven. When we compare it with PROM there were 16 AND gates for all the sixteen minterms. We need $4 \times 8 \times 4$ PLA in place of 16×4 PROM. There are $(4 + 8) \times 8 = 96$ fusible links compared to 64 fusible links in PROM. Let OR_0, OR_1, OR_2 and OR_3 give the outputs D_0, D_1, D_2 and D_3 , respectively. Let us number the fused links of AND_0 as 1, 2, ..., 8 for $A, \bar{A}, B, \bar{B}, C, \bar{C}, D$, and \bar{D} , and fused links of AND_0 to OR_0, OR_1, OR_2 and OR_3 as 9, 10, 11 and 12. Let us number the fused links of AND_1 as 13, 14, ..., 24 for $A, \bar{A}, B, \bar{B}, C, \bar{C}, D, \bar{D}$ and 4 OR inputs. Similarly, we number up to 96 from AND_7 , OR_3 input link (Figure 12.9 shows a demonstrative $4 \times 8 \times 4$ PLA).

Step 2: Finding the PLA fused link numbers needed.

Table 12.5 gives the not fused fusible links to implement the Gray code. Figure 12.9 shows the links at the inputs of the ANDs and connections at the ORs that are present.

TABLE 12.5 AND-OR not fused links

Bit	Not fused snapped link numbers							
	AND_0	AND_1	AND_2	AND_3	AND_4	AND_5	AND_6	AND_7
Y_3	1, 12							
Y_2		13, 16, 23	26, 27, 35					
Y_1				39, 42, 46	52, 53, 58			
Y_0						65, 68, 69	78, 79, 81	

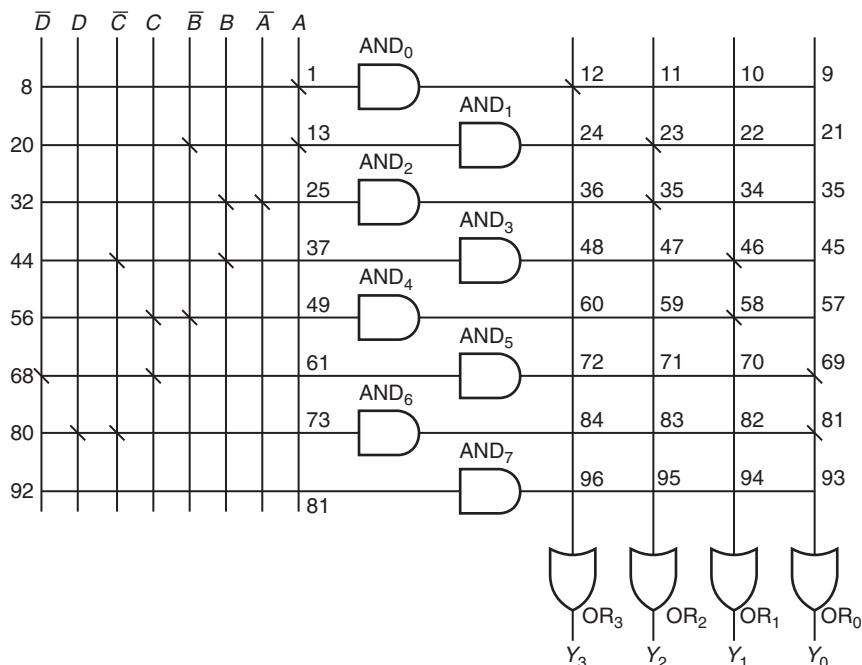


FIGURE 12.9 A demonstrative $4 \times 8 \times 4$ PLA that implements a Gray code converter ($I=7, n=4, m=4$ only used).

12.18 Digital Systems: Principles and Design

We note that $4 \times 7 \times 4$ PLA is needed to implement Gray code converter and $4 \times 7 \times 4$ PAL was needed for implementing the same using PAL. Total number of AND-OR gates links are 96 for PLA, 56 for PAL and 64 for a PROM.

Example 12.4

Show how to program the fusible links to get the four Boolean expressions correspond to the given sets of 4 combinational circuits implemented using PLA. Compare the number of fusible links and gates needed with respect to the PAL and PROM.

Solution

$$Y_0 = \Sigma m(2, 4, 5, 6, 7); Y_2 = \Sigma m(2, 4, 5, 6, 8, 9);$$

Recall Chapter 5. There are four variables as there are the terms $m(8)$ and $m(9)$ in Y_1 . Four variables Karnaugh Map (Table 12.6) for maximum 16 minterms in an SOP Expression is as follows:

TABLE 12.6 SOP minterms at map

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

The Karnaugh map for Y_1 is therefore as in Table 12.7:

TABLE 12.7 Map for Y_0

$A.B \backslash C.D$	$\bar{C}.\bar{D}$	$\bar{C}.D$	$C.\bar{D}$	$C.D$
$\bar{A}.\bar{B}$	00			1
$\bar{A}.B$	01	1	1	1
$A.B$	11			
$A.\bar{B}$	10			

$$Y_0 = \bar{A}.B + \bar{A}.C.\bar{D} \quad \dots(12.13)$$

The Karnaugh map for Y_2 is as in Table 12.8.

$$Y_2 = A.\bar{B}.\bar{C} + \bar{A}.C.\bar{D} + \bar{A}.B.\bar{C} \quad \dots(12.14)$$

Karnaugh map of the product $Y_0.Y_1$ is in Table 12.9

$$Y_2 = Y_1.Y_0 = \bar{A}.B.\bar{C} + \bar{A}.C.\bar{D} \quad \dots(12.15)$$

We find a subset of two prime implicants into Y_0 and Y_1 , both. Hence these can be implemented separately.

TABLE 12.8 Map for Y_1

$A.B \backslash C.D$	00	01	11	10
$\bar{A}.\bar{B}$	00			1
$\bar{A}.B$	01	1	1	1
$A.B$	11			
$A.\bar{B}$	10	1	1	

 TABLE 12.9 Map for $Y_0 \dots Y_1$

$A.B \backslash C.D$	00	01	11	10
$\bar{A}.\bar{B}$	00			1
$\bar{A}.B$	01	1	1	1
$A.B$	11			
$A.\bar{B}$	10			

$$Y_1 = A.\bar{B}.\bar{C} + Y_2; Y_0 = \bar{A}.B.C + \bar{A}.B.\bar{C} + \bar{A}.C.\bar{D} = Y_2 \quad \dots(12.16)$$

after rewriting it [because $\bar{A}.B = \bar{A}.B.(\bar{C} + C) = \bar{A}.B.\bar{C} + \bar{A}.B.C$].

Step 1: Finding the PLA number of fused links, ANDs and ORs needed—

We have a common subset of two prime implicant, $\bar{A}.C.\bar{D}$ and $\bar{A}.B.\bar{C}$. Two ANDs are needed for implementing it as well as Y_1 . The equations 12.15 and 12.16 have two other prime implicants, $\bar{A}.B.C$ and $A.B.\bar{C}$ respectively. These can be implemented by two other AND gates. Therefore, total four ANDs and two OR gates shall suffice in case of PLA.

Let us number the fused links of AND_0 , AND_1 up to AND_3 . The links are 1, 2, up to 8 for AND_0 , A , \bar{A} , B , \bar{B} , C , \bar{C} , D , \bar{D} . Two input OR gate links of AND_0 are 13 and 14. AND_1 , AND_2 and AND_3 can be numbered up to 56 (fourteen link with each AND product line). This PLA has four variables and two OR feedback input variables. (Figure 12.10 shows a demonstrative $4 \times 4 \times 2$ PLA circuit in a $4 \times 6 \times 2$ PLA).

Step 2: Finding the fused links needed—

Table 12.10 gives the PLA implementation. Links (13, 2, 5, 8) and (16, 17, 20, 27) give the prime implicants of expression 12.15 as well as for y_0 (subset in y_1). Links (29, 32, 34, 42) implements isolated term in equation 12.16. Link 53 and 56 implement y_0 feedback term in expression (12.16).

Example 12.5

Show how will we design a hardware lock from an eight input variable PLA having eight number 16 input AND gates to eight OR gates. Lock opens and generates

12.20 Digital Systems: Principles and Design

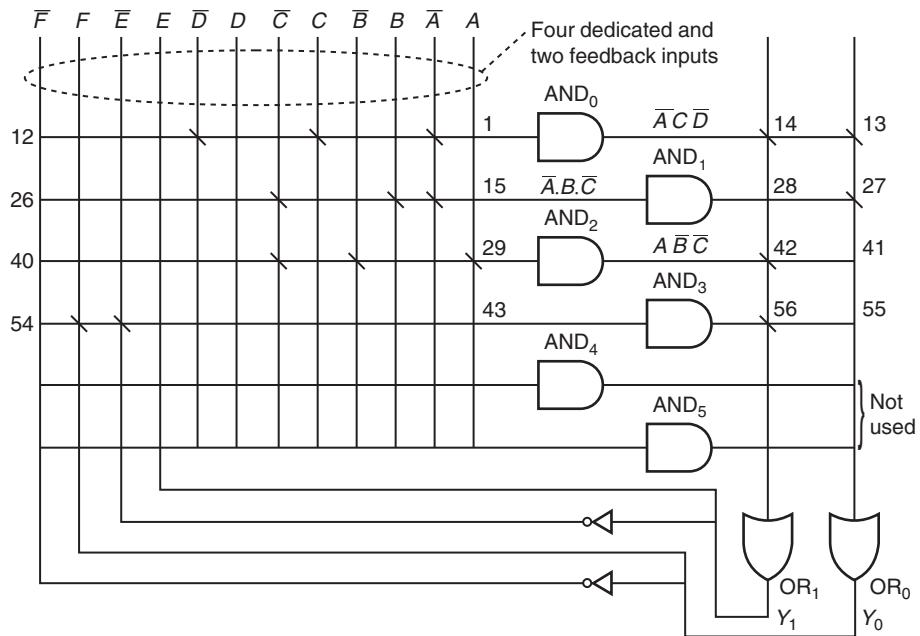


FIGURE 12.10 An implementation by $4 \times 4 \times 2$ PLA with four dedicated and two feedback input variables, 4 ANDs and 2 ORs ($I = 4$, $m = 2$, $n = 4$).

TABLE 12.10 Not fused links at PLA

Bit	Not fused (snapped) link numbers					
	AND ₀	AND ₁	AND ₂	AND ₃	OR ₁	OR ₀
$F_0(Y_0)$				53	42, 56	
$F_1(Y_1)$	2, 5, 8	16, 17, 20	29, 32, 34			13, 27

output = 1 when input variables are 10100111 and OR outputs Y_0 to Y_7 are as follows:

$$Y_0 = \sum m(0, 12, 38, 89); Y_1 = \sum m(10, 112, 138, 209); Y_2 = \sum m(0, 12, 117, 133, 1769); Y_3 = \sum m(37, 49, 74, 189); Y_4 = \sum m(53, 92, 108, 128); Y_5 = \sum m(8); Y_6 = \sum m(83, 1133); Y_7 = \sum m(1, 128, 138, 59); \text{ [Total 25 distinct minterms]}$$

Solution

Maximum minterm is $m(1769)^1 \cdot 2^{10} < 1769 < 2^{11}$. Hence, there are 11 literals maximum. Eight literals are for dedicated inputs. It means there are feedbacks from three ORs. Total number of inputs = 22. Number of ANDs and product lines are 25. Let us number the fuse links, 1 to 22 for AND_0 , 23 to 30 for the inputs OR_0 to OR_7 from AND_0 . Similarly, number the fuse links, 31 to 60 for AND_1 , 61 to 90 for the AND_2 and OR inputs. Now last AND_{24} will have links between 721 and 750. Which link remains not-snapped is left as an exercise to the reader. Figure 12.11 shows the link numbers that can be fused to implement the above.

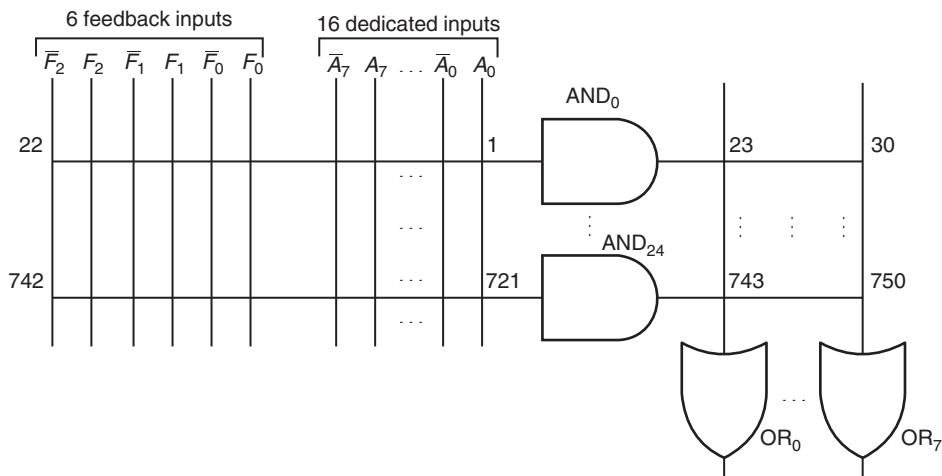


FIGURE 12.11 Hardware lock implementation using an eight input variable PLA having eight number 16 inputs AND gates to eight OR gates.

■ EXERCISES

1. Show how to program the fusible links to get a 4-bit 7536 code from the binary inputs using a PAL. Also compare the design requirement with a PROM.
2. Show how to program the fusible links to get a 4-bit 8421 code from the binary inputs using a PAL. Also compare the design requirement with a PAL.
3. Show how to program the fusible links to get a 4-bit 5421 code from the binary inputs using a PAL. Also compare the design requirement with a PAL.
4. Show how to program the fusible links to get the four Boolean expressions correspond to the given sets of combinational circuits implemented using PLA. Compare the number of fusible links and gates needed with respect to the PROM and PAL. The circuit Karnaugh maps are as in Tables 12.11 and 12.14.
5. Show how to program the fusible links to get the four Boolean expressions correspond to the given sets of four combinational circuits in Exercise 3 implemented using PROM.

TABLE 12.11 SOPs map in a circuit

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

12.22 Digital Systems: Principles and Design

TABLE 12.12 SOPs map in a circuit

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

TABLE 12.13 SOPs map in a circuit

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

TABLE 12.14 SOPs map in a circuit

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

6. Show how will we design a hardware lock from a four input variable PAL having four number eight input AND gates and four OR gates. Lock opens ad generates output = 1 when input variables are 10111101 and OR outputs Y_0 to Y_7 are all 1s.

■ QUESTIONS

1. Describe the fusible links in AND-OR arrays at a PAL. Describe 16L8 PAL
2. Describe the fusible links in AND-OR arrays at a PLA.

3. Why is it possible to work with l AND gates in case of n input variables when using a PAL? (Hint: Less number of minterms, common or complementary sets of prime implicants).
4. Why is it possible to work with l AND gates in case of n input variables when using a PLA? (Hint: Less number of minterms, subsets of prime implicants in Boolean expressions and their products).
5. Why do we need 2^n AND gates in case of a PROM in case of n input variables? (Hint: 2^n minterms).
6. What is the advantage of a PAL compared to PLA and PROM? (Hint: Simpler minimization, less number of fused links).
7. What is the advantage of a PLA compared to PAL and PROM? (Hint: Complex logic functions implementation with less number of AND-OR arrays).
8. What is advantage of a PROM compared to the PLA and PALs? (Hint: PROM also works as the memory elements for data tables, instructions, data sets, etc.).
9. How do you implement the multi outputs (Boolean expressions) with minimal fusible links in a PAL?
10. How do you implement the multi outputs (Boolean expressions) with minimal fusible links in a PLA?
11. Explain a hardware lock design using a PLA.
12. What is the advantage of tristate outputs provision in a PAL? What is the advantage of a registered output PAL? (Refer Section 14.1)

This page is intentionally left blank.

CHAPTER 13

Logic Gates

OBJECTIVE

In this chapter, we shall study RTL, DTL, TTL, ECL, ICL, HTL, NMOS and CMOS logic gates; the difference between the various families of gates and their speeds; propagation delay; operating frequency; power dissipated per gate; supply-voltage levels; and operational-voltage levels that define logic states 1 and 0.

In Chapter 2, we learnt six digital electronics logic operations used as basic operations in the complex circuits. These are NOT, NAND, AND, OR, NOR and XOR. We also learnt that an interesting combination is NOT-XOR (recall the Figures 2.1 and 2.2). There are the different types of circuits of the logic gates for implementing these operations; RTL, DTL, TTL, ECL, ICL, HTL, NMOS and CMOS. How are these circuits made? We shall learn this in this chapter. How does the different family of gates differ? These questions are answered in this chapter.

Let us first revise and learn the following concepts to have ease in learning the topics of this chapter.

13.1 REVISION OF THE IMPORTANT GATES

(A) NOT

NOT gate has the unique property that its output is 1 if the input is at 0 logic level and the output is 0 if input is at 1 logic level.

13.2 Digital Systems: Principles and Design

NOT logic is used when an output is desired to be the complement of the input. If all inputs of NAND gates are joined then that will also act as a NOT gate. NOT gate is also called inverting logic circuit. It is also called a complementing circuit.

Two DeMorgan theorems help a digital circuit designer to implement all the other logic gates with the help of either NOR gates only or NAND gates only. A NOT gate is implementable by a NAND or a NOR by joining all of their inputs.

(B) AND

A NAND gate followed by a NOT gate gives us an AND gate. Its symbol differs from NAND only by the omission of a bubble (circle). Its unique property is that its output is 0 unless all the inputs to it are at the logic 1. Two inputs AND gate has the following way of writing its operations.

$$F = A_1 \cdot A_2 \quad \dots(13.1)$$

Dot between two states indicate AND logic operation using these.

Remember

AND gate unique property is that output is 1 only if all of the inputs are at 1 logic level.

(C) OR

A NOR gate followed by a NOT gate (made by the common input NOR) gives us OR gate. Its symbol differs from the NOR only by the omission of a bubble (circle).

OR gate unique property is that output is ‘1’ if any of the inputs are at ‘1’ logic level.

(D) NOR

Its unique property is that its output is 0 if any of its inputs is 1. A NOR gate is a basic building block for other types of the logic gates other than TTLs (Section 13.4.3). In the TTL circuits, a NOR is fabricated in an IC by the several NANDs.

(E) XOR

An XOR gate (is called ‘Exclusive OR’ gate). Its unique property is that the output is 1 only if odd numbers of the inputs at it are 1s.

(F) NOT-XOR

A NOT-XOR gate has a unique property that the output is 0 only if odd numbers of the inputs are 1. It is like a NOR gate but differs in the output when even number of its inputs are 1.

(G) BUFFER (Non Inverting)

A BUFFER gate simply gives same state at output as that at the input. It can also be made by connecting two inverting circuits (NOTs) in series. It gives one advantage that we can connect to an output a greater number of other gates logic inputs with a non-inverting buffer in between than without the BUFFER, of course, at the expense of an additional propagation delay time which depends upon speed of the BUFFER logic gate internal circuit.

13.2 DIODE CIRCUIT

A *p-n* junction diode, shown by a triangle with a line for the *p* and *n* end, respectively, is used in the logic circuits as follows. (refer Figure 13.1(a)). It is used in making logic circuits due to the following factors.

1. When a silicon *p*-end connects through a resistance ~ 2200 Ohm (~ 2.2 k Ω) to the supply of 5 V it will flow $(5\text{ V} - 0.7\text{ V})/2.2\text{ k}\Omega = \sim 2$ mA current and its *p*-end will be at ~ 0.7 V (called threshold voltage) with respect to *n*-end connected to the supply ground. This *p*-end at low voltage now represents the logic state 0. (refer Figure 13.1(b)).
2. When the *p*-end connects through a resistance ~ 2200 Ohm (~ 2.2 k Ω) to the supply of 5 V but the *n*-end also connects to a high voltage ~ 2.4 V to 5 V, the current through the diode and resistance circuit will be negligibly small and the diode voltage is below the threshold. The *p*-end will also be at high voltage with respect to the supply ground. The both *p*-end and *n*-end now represents the logic state 1. (refer Figure 13.1(c)).

Figures 13.1(d) and (e) shows how the two diodes can be used to create two-input AND and OR gate, respectively, and can also be used to give functionally the AND and OR operations within another logic circuit.

13.3 BIPOLAR JUNCTION TRANSISTORS AND MOSFETS

13.3.1 *N-P-N* Transistor Common Emitter Circuit

An *n-p-n* junction transistor, also called *n-p-n* bipolar junction transistor (*n-p-n* BJT) can be used in logic circuits as follows: Figure 13.2(a) shows an *n-p-n* BJT symbol and its common emitter circuit. It also shows the D.C. state currents I_C , I_B , I_E through (i) V_{CC} supply-collector resistance, (ii) base-emitter and (iii) emitter, and V_{EE}^- supply ground, respectively. It also shows the D.C. state voltages V_R , V_{BC} , V_{BE} across (i) V_{CC} supply and collector resistance, (ii) collector and base and (iii) base and emitter and V_{EE}^- supply ground, respectively.

When collector-end connects through a resistance between 0.6 k Ω to ~ 4 k Ω) to the supply of 5 V it will flow the current as per the base current I_B at that instant and the logic state (output) at the collector shall also depend on that. This circuit in its normal mode of operation is called inverting current amplifier and a small change in

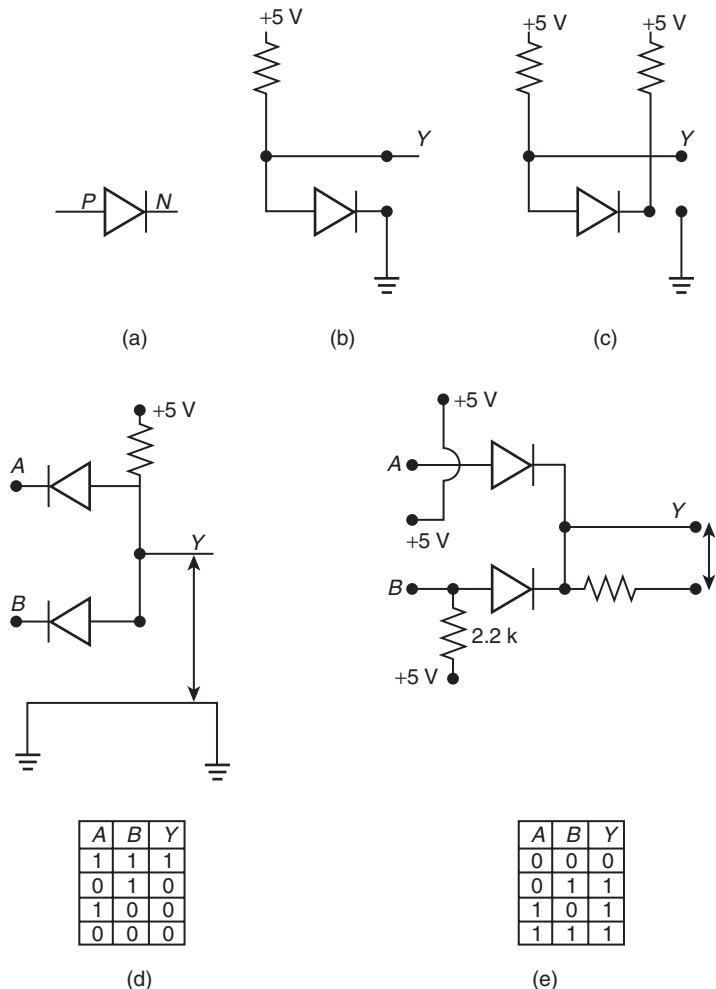


FIGURE 13.1 (a) A diode symbol. (b) A silicon diode at logic 0 (c) The diode at logic 1 (d) Two diodes creating a two-input AND (Truth table in inset). (e) Two diodes creating a two-input OR (Truth table in inset).

base current causes a large change in collector current. It is called current switching transistor circuit when it is operated in one of the two—*cut-off* mode or *saturation* mode. In logic circuits, its use as a current switching circuit is of interest. The cut-off and saturation modes are switched by the base current changes such that the logic outputs are 1 and 0, when the logic inputs are 0 and 1, respectively.

1. **Cut-Off Mode Operation:** Base-emitter $p-n$ junction is reverse biased ($0 < V_{BE} < V_{BE(on)}$) and there is negligible current I_B flowing across it. The base-collector p -junction is also reverse biased ($0 < V_{BC}$) and there is negligible current I_C through it. Voltage drop through R is negligible due to this transistor and collector is thus at high voltage of supply voltage ~ 5 V. The logic output represents 1 till other circuits or logic gates connected to it through R_B decrease this voltage at the collector below a limit. Note: V_{BE} is

base-emitter bias and $V_{BE(on)}$ is the bias needed to make base-emitter junction forward biased. We must remember $V_{CE(sat)} = \sim 0.5$ V, $V_{BE(cutoff)} < 0.7$ V in cut-off mode of operation. (refer Figure 13.2(b)).

- 2. Saturation Transistor Circuit Operation:** Base emitter $p-n$ junction is forward biased ($V_{BE(sat)} = \sim 0.7$ V) and there is significant current I_B flowing across it. The base collector $p-n$ junction is also forward biased ($V_{CE(sat)} = \sim 0.2$ V, base current large enough to give $I_C < \beta I_B$, where β is the forward current gain). There is significant current I_C (\sim mA) through it. Voltage drop through R is large due to I_C this transistor and collector is thus at very low voltage ($V_{CE(sat)} \sim 0.2$ V). The logic output represents 0 till other circuits or logic gates connected to through this transistor may also sink the current through it. We must remember, that when $I_C \approx \beta I_B$, the $V_{CE(sat)} = \sim 0.2$ V, $V_{BE(ON)} = V_{BE(sat)} = \sim 0.7$ V and $V_{BC(sat)} = V_{BC(ON)} = \sim 0.5$ V and there is saturation mode of operation (refer Figure 13.1(c)).

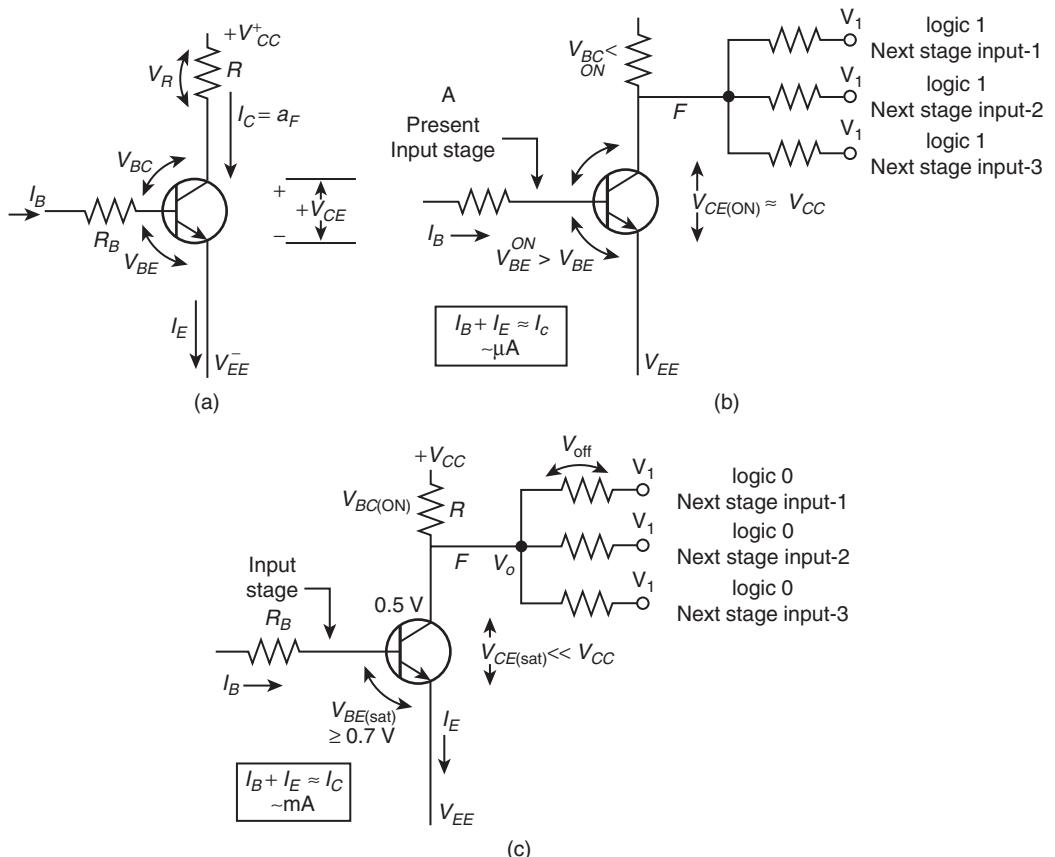


FIGURE 13.2 (a) The $n-p-n$ BJT symbol and its common emitter circuit (b) The transistor collector negligible current corresponding to the OFF state of the transistor, which give functionally the 1 logic output state = 1 from the transistor collector (c) The transistor collector high current (\sim mA) corresponding to the saturation state of the transistor, which give functionally the 0 logic output states, respectively, for giving the input(s) 0 to another logic circuit(s).

13.3.1.1 Important Equations for the Circuit

Important equations are $I_B = (V_i - V_{BE(ON)})/R_B$... (13.2)

where V_i is input voltage applied to R_B . This is because potential difference R_B equals the term in the bracket. In both saturation and normal modes, equation holds. Also,

$$I_C = \beta I_B = \beta(V_i - V_{BE(ON)})/R_B \quad \dots(13.3)$$

The output at F is given by

$$V_o = V_{CC} - I_C \cdot R = V_{CC} - R \cdot \beta(V_i - V_{BE(ON)})/R_B \quad \dots(13.4)$$

When the V_i increases the V_o decreases till $V_o = V_{CE(\text{sat})}$.

13.3.1.2 Important Feature of the $n-p-n$ Common Emitter Circuit in Application in the Logic Circuits, and the Operational Voltage Levels that Define Logic States 1 and 0

1. V_i for the first stage can vary within a range provided the transistor remains in cut-off mode and the output is constant ($\sim V_{CC}$), which decreases with the next stage loading circuit. We can assign a range for V_o (between the V_{OH} maximum = ($\sim V_{CC}^+$) and V_{OH} minimum) and logic is said to be 1. The next stage(s) V_i can be between V_{OH} and (V_{OH}) minimum permitted drop in—between F and next stage(s) (Right side circuit of Figure 13.2(c)).
2. V_i can vary within a range provided the transistor remains in saturation mode and the output is constant ($\sim 0.2V$), which changes due to the currents from the next stage loading circuit(s). We can assign a range for V_o ($V_{OL} = V_{OL}$ maximum to supply ground, V_{EE}^-) and logic is said to be 0. The next stage(s) V_i can be between supply ground V_{EE}^- and (V_{OL} maximum + permitted drop in between F and next stage(s)).

13.3.2 MOSFET Circuits

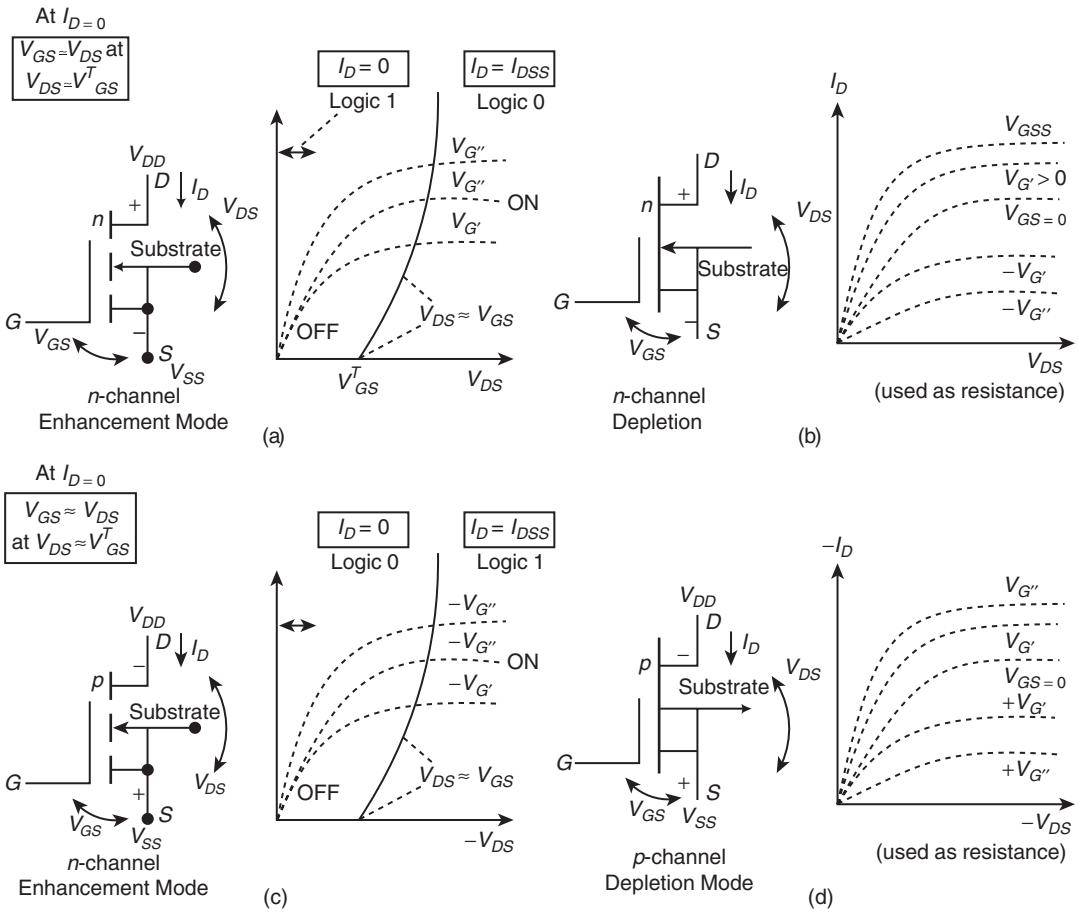
MOSFET (Metal-Oxide Semiconductor Field Effect Transistor) logic circuits are most used circuits today. These are small and simple to fabricate on VLSI (very large scale integrated) chips. All memories, microprocessors, and computers therefore use these.

13.3.2.1 MOSFET Circuit Operation

A MOSFET is a three-layered structure; a metal (gate), a very thin oxide layer and a semiconductor (channel), in four types: n -channel, enhancement mode n -channel depletion mode, p -channel and enhancement mode p -channel depletion mode. (Figures 13.3(a) to (d) gives the symbol and gate-characteristics of four types of MOSFETs).

Enhancement mode n -channel MOSFET shows the characteristics as follows. When voltage between gate and source, $V_{GS} = 0$, the current I_D between the drain and source is 0 and is independent of voltage between drain and source, V_{DS} .

Small V_{DS} Region: MOSFET is OFF and is in cutoff state below a positive threshold Voltage V_{GS}^T . After a threshold, when $V_{GS} > V_{GS}^T$, a channel is formed (established for I_D flow) between drain and source and the I_D increases and linearly varies with V_{DS} . MOSFET is said to be in ON state. Further, when the V_{GS} increases, the slope of change in I_D is steeper and steeper with respect to V_{DS} . Channel resistance decreases steeply with $(V_{GS} - V_{GS}^T)$. MOSFET is said to be in ON state. Channel width (along with electrons flow) is constant is function of $(V_{GS} - V_{GS}^T)$ (Figure 13.3(a)) left side logic 0 region characteristic).



$$R_{Ch} \text{ Channel Resistance} \propto \text{Channel length and}$$

$$R_{Ch} \propto \frac{1}{\text{Channel width}}$$

Channel width is a function of V_{GS}
(= 0 when $V_{DS} < V_{GS}^T$)

FIGURE 13.3 (a) to (d) Symbols of four types of MOSFETs. Also are the linear region characteristics for n-channel enhancement, n-channel depletion, p-channel depletion, and p-channel enhancement respectively. (e) Change in R_{Ch}

Constant I_D with V_{DS} Region: Now, $V_{DS} = V_{GS} + V_{GD}$. Increase in V_{DS} and therefore V_{GD} causes the channel narrowing, decrease in channel resistance. At a voltage called pinch-off voltage, V_{GS}^T , the channel resistance near drain is 0. At a given $V_{GS} = V_{GSS}$ the I_D does not change significantly with V_{DS} in ON state when $V_{DS} > V_{GSS}^T$. The variation is no longer linear but is non-linear with V_{DS} and becomes nearly constant. Current I_{DSS} continues to flow in this region. (Figure 13.3(a) right side logic 1 region characteristic).

Enhancement mode needs $V_{GS} > V_{GS}^T$. A depletion mode *n*-channel MOSFET conducts at $V_{GS} = 0$ also and conduction starts from a negative threshold gate-source voltage, $V_{GS} > -V_{GS}^T$. MOSFET is OFF and is in cutoff state below this voltage. When voltage between gate and source, $V_{GS} = 0$, the current I_D between the drain and source is 0 and is independent of voltage between drain and source, V_{DS} (Figure 13.3(b)).

Enhancement mode *p*-channel MOSFET and Depletion mode *p*-channel MOSFET works similar to *n*-channel MOSFETs in these modes. The current is carried by holes; therefore the direction of current and voltages is reverse in *p*-channel with respect to *n*-channel.

For turning the MOSFET in ON state, an enhancement mode *p*-channel needs $V_{GS} < -V_{GS}^T$. Depletion mode *p*-channel needs $V_{GS} < +V_{GS}^T$. (Figure 13.3(c) and (d))

Enhancement mode MOSFET as a Resistance: If we join the drain and gate, $V_{GS} = V_{DS}$. When $I_D = 0$, $V_{DS} = V_{GS}^T$. The resistance is non-linear. Very high at $V_{DS} = V_{GS}^T$ and drops sharply with V_{DS} change. (Figure 13.3(e))

Depletion mode MOSFET as a Resistance: If we join the drain and gate, $V_{GS} = V_{DS}$. When $I_D = 0$, $V_{DS} = -V_{GS}^T$. The resistance is non-linear. Very high at $V_{DS} = -V_{GS}^T$ and drops sharply with V_{DS} change. At $V_{GS} = 0$ also it is not very high.

13.3.2.2 MOSFET Circuit Features

Static region resistance between input at gate and the channel is very high and the MOSFETs when at cut-off show very high input D.C. impedance. (There is only capacitive impedance between gate and source.) When at saturation, MOSFET (ON region) resistance between the drain and source is low (when drain source voltage is below the pinch off). The saturated region is above a pinch-off voltage, (the resistance between drain and source is more than between a BJT's collector and emitter).

13.4 RTL, DTL, TTL LOGIC GATES

Section 13.3 explained how a BJT transistor or a MOSFET can be used in two regions cut-off region and saturation region. This fact can be used to design the logic gates. Followings are the logic gates built on this concept.

13.4.1 Resistor–Transistor Logic (RTL)

Both input and output stage circuits in a NOR logic gate is shown in the Figure 13.4.

- 1. Input Stage:** Two inputs to two *n-p-n* transistors through two 450 Ohm resistances.
- 2. Output Stage for the next Input Stage:** The common output *F* connects the collector(s) of the transistors with the input stages given to other resistances at the next stage RTL gates.

Logic Operation

When input voltage at logic input *A* is low, the transistor T_A does not conduct (in cutoff stage). Similarly, when input *B* is low, the transistor T_B does not conduct. Output of these transistors is from their collectors, which are joined together and given supply voltage of 3.6 V through a 640 Ohm resistance. When both T_A and T_B do not conduct, the output at *F* is high voltage, close to the supply voltage of 3.6 V ($V_o = V_{CC} - I_C \cdot R$).

When input voltage at logic input *A* is high, the transistor T_A conducts. Similarly, when input *B* is high, the transistor T_B conducts. Output of these transistors is from their collectors, which are joined together and given supply voltage of 3.6 V through a 640 ohm resistance. Therefore, when either of T_A and T_B conducts, the output at *F* is low voltage, closer to the supply ground voltage ($V_{CE(\text{sat})} = \sim 0.2$ V). *F* connects to *M* number *R-T* stage (resistance transistor stage) transistors, T_1 , T_2 ... T_M ($M = 4$ in the figure)

NOR gate gives output = 1 when its all inputs are equal to 0. It gives output = 0, when any of the input = 1. Therefore, the circuit of Figure 13.4 works as two-input NOR gate.

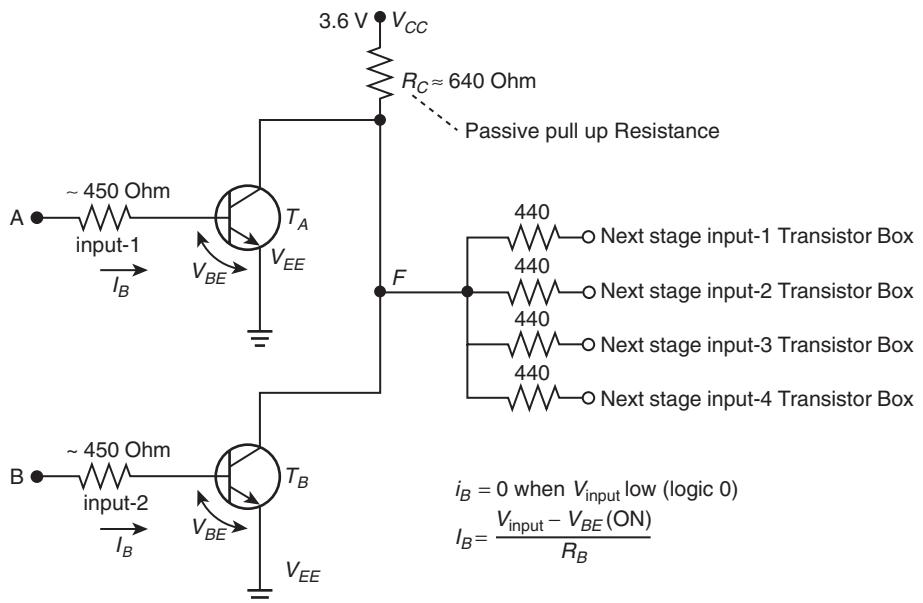


FIGURE 13.4 An RTL circuit for a NOR gate the output of which given to four gates.

Case of Input Stage Output Connected to m Output Stage Transistors as Load

When the output from common collector junction is to m number $R-T$ stage (resistance transistor stage) transistors, each drives through 450 Ohm resistance as the load, the current from input stage 640 Ohm resistance will be divided into m parts.

Base-Emitter voltage, when a transistor is ON (conducting in saturation region), is $V_{BE(sat)} \approx 0.8V$. Base current for each output stage transistor T_j ($j = 1$, will be $I_B = I_B' = (V'_i - V_{BE(ON)})/R_B = (1/m) (3.6 V - 0.8V)/(640 \text{ Ohm} + (450/m) \text{ Ohm}) = V'_i$ is the input to T_j . If $m = 4$, I_B will be 0.93 mA.

(Point, you must note and practice: When solving a numerical problem, at each stage don't forget to put the units at numerator and denominator both).

The collector current when this transistor is conducting (is in saturation) will be $I_C = (3.6V - 0.2 V)/640 \text{ ohm} = 5.3 \text{ mA}$. A T_j transistor gain h_{fe} must be equal or greater than I_{CO} (saturation stage current)/ I_B . For $m = 4$, $h_{fe} >= (5.3 \text{ mA}/0.93 \text{ mA})$, $h_{fe} >= 5.8$. [h_{fe} = small signal current gain for the common emitter configuration, and is assumed here close to the D.C. current gain β in equations (13.2) to (13.4)].

We can have more output stage transistors (j can be higher) if h_{fe} is higher. More output stage transistors means more output stages can be driven from the output F .

For logic output of 1 at F , the output voltage depends on m and m maximum permissible number depends on the h_{fe} value of T_j transistor. Higher h_{fe} permits higher m .

For logic output 0 at F , the output voltage does not depend on m . This is because of the transistor T_A or T_B conducting and in saturation. The output at F is $\sim 0.2 \text{ V}$ when input at A or $B >= 0.5 \text{ V}$ (cut-off voltage). Cut-off voltage is the voltage, below which the transistor cut-off the current and above which the transistor is in saturation and is conducting.

Definition of Fan Out

Number of logic gates at the next stage(s) that can be loaded to a given logic gate output so that voltages for each of the possible logic state remain within the defined limits (refer for example, for 1 between V_{OH} minimum and V_{CC}^+ and for 0 between V_{OL} maximum and V_{EE}^-).

Note: V_{OH} is the voltage for next stage when logic = 1, V_{OL} when = 0, V_{OH} minimum and V_{OL} maximum are the permissible limiting values as per logic definition.

Propagation Delay

Definition of Propagation Delay

Propagation delay for a logic output from a logic gate means the time interval between change in a defined reference point input voltage and reflection of its effect at the output. It can also be defined as the time interval between changes in a defined logic level input and reflection of its effect at the output logic level.

(Note: The slightly different values of the delays are obtained when the input change from 0 to 1 and change from 1 to 0. We can take average propagation delay. Also a propagation delay is also subject to variations in power supply and temperature. We can then define a statistical deviation and an average).

Calculation of Propagation Delay: Let base-emitter capacitance = C nF (nF means nanoFarad). If $m = 4$, the total capacitance all T_j in parallel being = $4.C$. Resistance = $640 \text{ Ohm} + (450/m) \text{ Ohm} = 752.5 \text{ Ohm}$. Propagation delay = $(640.m + 450) C \text{ ns.}$ ($\text{nF} \times \text{Ohm} = \text{ns}$).

Definition of Noise Margin

Noise margins for the logic outputs 1 and 0 means the permitted worst case voltage levels variations of the logic output 1 and 0, respectively, when an output from a stage is the input at the next stage(s). The margins are permitted due to expected internal temperature variations and power supply variations. Permitted noise margins reflects the digital circuit immunity and worst case performance consistency in the presence of an induced noise.

Calculation of the Noise Margins: For logic state 0, the output at F can be $\sim 0.2 \text{ V}$ and maximum 0.5 V , else the T_j will start conducting and go in saturation. Hence noise margin of logic state 0 in RTL based logic circuit is 0.3 V .

We have seen that the output voltage depends on m . For logic state 1, the output at F can be calculated as follows. Collector current at $T_j = 5.3 \text{ mA}$ for each transistor T_j base current is $5.3 \text{ mA}/h_{fe} = 0.265 \text{ mA}$ assuming $h_{fe} = 20$.

For $m = 4$, total base current needed from $F = m \cdot 0.265 \text{ mA} = 1.06 \text{ mA}$. Voltage at F = voltage drop between T collector and emitter + total base current multiplied by total base resistance $(450/m) \text{ Ohm}$. Thus voltage at $F \geq 0.8 \text{ V} + 1.06 \text{ mA} \cdot (450/m) \text{ Ohm} \geq 0.92 \text{ V}$.

Available output voltage at F can be calculated as follows:

$V_O = 3.6 \text{ V} - \text{voltage drop at } (450/m) \text{ ohm collector resistance} + \text{voltage drop at } 0.8 \text{ V base-emitter.} = 3.6 \text{ V} - (640 \text{ Ohm}/(640 \text{ Ohm} + 450/m \text{ Ohm}) \cdot (0.8) \text{ V} = 1.2 \text{ V}$ for $m = 4$.

Hence the logic 1 at F can be between the 1.2 V and 0.92 V for $m = 4$ and $h_{fe} = 20$. For logic 1, noise margin will be 0.28 V when $m = 4$ and $h_{fe} = 20$.

Wired Logic (Wired OR or Implied OR) Connection

If outputs F and, F' at two RTL gates are made common (connected), the output can be considered as AND operation between the logic outputs. Because when both the outputs correspond to cutoff stages of the transistors, the output will remain unaffected and will be 1. When any of the outputs correspond to saturation condition $\sim 0.2 \text{ V}$, the output from common point will become 0.2 V . If A, B are the inputs at one RTL NOR gate and C, D are inputs at another, NOR the output will be as follows:

$$F = (\overline{A+B})(\overline{C+D}) = \overline{(A+B+C+D)} \quad (\text{Using DeMorgan theorem}).$$

13.4.2 Diode–Transistor Logic (DTL)

A DTL circuit in Figure 13.5 works as three input NAND gate.

1. **Input Stage:** It is based on the applying of two or more inputs to two or more *n*-ends of the *p-n* junction diodes in place of passive ~ 450 Ohm resistance in an RTL circuit (Figure 13.4). The *p*-ends are common and connect to a 5000 Ohm resistance R_D , which connects to supply voltage of 5 V. The common point *P* also connects to a transistor-base through two forward biased diodes. The transistor (*T*) base also connects to the emitter through 5000 Ohm R_B connect. The emitter of *T* connects to supply ground. The collector of *T* connects to supply through a resistance R_C of 2200 Ohm.
2. **Output Stage for the next Input Stage:** A common output from the transistor *T* at *F* is given to other diodes at the other next stage(s) logic gates (DTL gates), which will get the input from the transistor, *T*.

Both input stage diodes and transistor form a NAND gate and next input stage circuit(s). The DTL based NAND and its connection to next stages is shown in the Figure 13.5.

13.4.2.1 Logic Operation for the Output at *F*

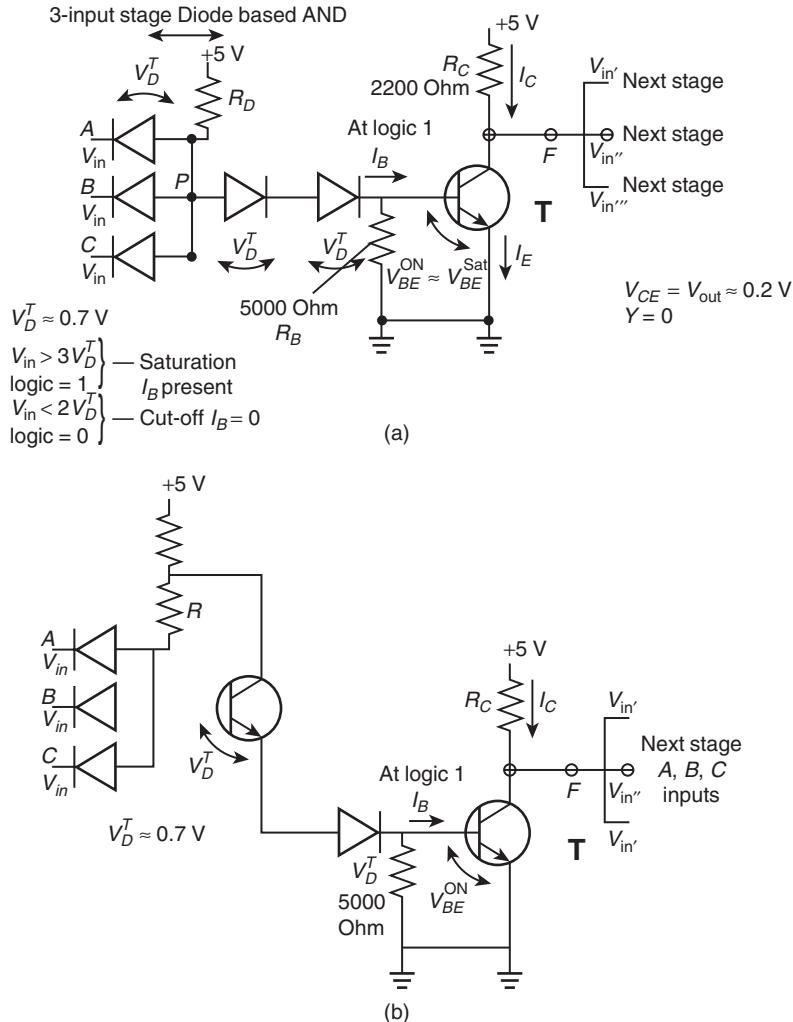
If input *A* or *B* or *C* is low ~ 0.2 V (near supply ground), the diode in the path of *A* or *B* or *C*, respectively will start conducting. Current through the diode will be approximately equal to $(5\text{ V} - 0.7\text{ V} - 0.2\text{ V})/(5000\text{ }\Omega) = \sim 0.8\text{ mA}$. (Voltage drop across a conducting *p-n* junction diode is threshold voltage $V_D^T \approx \sim 0.7\text{ V}$). Voltage drop of 0.9 V ($V_{in} + 0.2$) connects to the base of *T* through 2-diode pair, each needing threshold voltage of 0.7 V and total 1.4 V to turn ON and conduct which is much more than 0.9 V available. Hence the transistor base is at $\sim 0\text{ V}$, (below cutoff voltage). When *T* is not conducting the output *F* = 1, high ($\sim 5\text{ V}$). *T* does not conduct when any (or both of the inputs) is low because voltage at the base drops below the cut-off voltage needed at the base.

If all inputs *A*, *B* and *C* are high ($>0.7\text{ V}$), the voltage at common *p*-ends will start exceeding 1.4 V and the 2-diode pair to the base will start conducting. When each input *A*, *B* and *C* exceeds 1.4 V and the voltage at the common *p*-ends exceeds $(1.4\text{ V} + V_{BE(ON)}) = 2.1\text{ V}$, the base-emitter junction starts conducting. When each input exceeds 1.4 V , the diode at the input stops conduction and when exceeds 2.1 V , becomes reverse biased. $V_{BE(ON)}$ remains at 0.7 V . If transistor *T* base-emitter current exceeds a limit, the *T* goes in saturation mode and it will start conducting current I_C through R and $V_{CE} \approx 0.2\text{ V}$. Therefore, *F* = 0 when *A* and *B* both 1.

Property of a NAND is that its output is 0 when both the inputs are 1. Therefore, the circuit in Figure 13.5 works as a three-input NAND gate.

13.4.2.2 Case of Input Stage Output *F* Connected to *m* Output Stage Transistors as Load

When the output from common collector junction connects *m* number *D-T* stage (diode-transistor logic stage) transistors, each drives through one input diode and 2-series diodes (similar to first stage *T*), the current from input stage is 0 from the *F* when all the next stage inputs are high.



FIGURES 13.5 (a) The DTL based NAND and its connection to next stages (b) Modified DTL circuit by replacing 2-diode combination by one diode and one transistor.

When a transistor T is conducting (logic 0 at F), the $V_{CE} = \sim 0.2$ V (in saturation stage) and $V_{BE}^{sat} = \sim 0.8$ V. The currents from each output stage diode with the transistors T_j ($j = 1, \dots, j$) will be $m \cdot I_C$. The T will remain in saturation until the condition, $mI_C < \beta I_B$ (Refer Section 13.3) remains true. $I_B = (2.1\text{V})/5000 \text{ Ohm} = 0.4 \text{ mA}$ (the condition is also called *current-sink logic* condition). [$3V_D^T = 2.1 \text{ V}$] We can have more input $D-T$ stages (j can be higher) if $h_{fe} (\approx \beta)$ is higher. More input stages means more input stages can be driven through the output F . For logic output of 1 at F , the output voltage does not depend on m .

Calculation of Fan out: Number of logic gates at the next stage(s) that can be loaded to a given logic gate output is the fan-out. Fan-out $m = \beta I_B / I_C$.

Calculation of Propagation Delay: Let base-emitter capacitance = C nF (nF means nanoFarad). If $m = 4$, the total capacitance being all T_j in parallel = $4 C$. Resistance is very small between base and emitter in logic 1 state. Therefore, transistor turn-on delay is small. Resistance in logic 0 state is 5000 ohm, therefore turn-off propagation delay = $(5000) mC$ ns. ($nF \times \text{Ohm} = \text{ns}$). Typically, the turn-on delay is 30 ns and turn-off delay is 80 ns.

Increasing Fan-out by a Modified DTL circuit: We have seen that $I_B = 0.4$ mA in 2-diode base-input based DTL. Figure 13.2(b) shows a modified DTL circuit by replacing 2-diode combination by one diode and one transistor. This improves the fan-out due to increase in I_B .

Wired Logic (Wired AND or Implied AND) Connection: If outputs F and F' at two DTL NAND gates connected, the output can be considered as AND operation between the logic outputs. Because when both the outputs correspond to cutoff stages of the transistors, the output will remain unaffected and will be 1. When any of the outputs correspond to saturation condition ~ 0.2 V, the output from common point will become 0.2 V. If A, B are the inputs at one DTL NAND gate and C, D are inputs at another, NAND the output Y on joining F and F' at common terminal will be as follows:

$$Y = (\overline{A \cdot B}) \cdot (\overline{C \cdot D}) = \overline{(A \cdot B + C \cdot D)} \quad (\text{Using DeMorgan theorem}).$$

13.4.3 Transistor–Transistor Logic (TTL)

13.4.3.1 TTL NAND Gate

Since 1964, the digital electronic circuits based on the TTL have been introduced. TTL means that the circuit is based-upon transistor-transistor logic (TTL). The transistors in it are bipolar junction transistors (BJTs). In the TTL digital circuit, a voltage level V_{CC}^+ of about +5 V with respect to the ground potential (GND) defines ‘high’ i.e. 1, and the level below about 0.65 V with respect to the V_{EE}^- GND defines ‘low’ i.e. 0 (Table 1.1). A typical TTL digital circuit is shown in Figure 13.6(a). Figure 13.6(b) shows passive pull up for using the circuit for the wired OR logic TTL output Figure 13.6(c) shows an open collector gate output. Figure 13.6(d) shows the symbol of four inputs NAND gate.

Figure 13.6(a) shows the TTL NAND gate electronic circuit using two transistors T and T' . Figure 13.6(b) gives the symbol of 4-input NAND gate shown in Figure 13.6(a). [NAND has state 1 most of the times as its output F is 0 only when with all the inputs (for example, A_1 and A_2) are 1.] A NAND gate can be said to be basic building block of the all-digital TTL logic gates and other digital circuits.

This NAND circuit differs from the DTL NAND circuit in Figure 13.5(a) as follows:

1. There is multi-emitter transistor at the input in place $p-n$ diode at the inputs A and B . (Figure 13.5(a)). Four inputs multi emitter-base junction is like a collection of four $p-n$ diodes. Multi-emitter junction at transistor T forms a

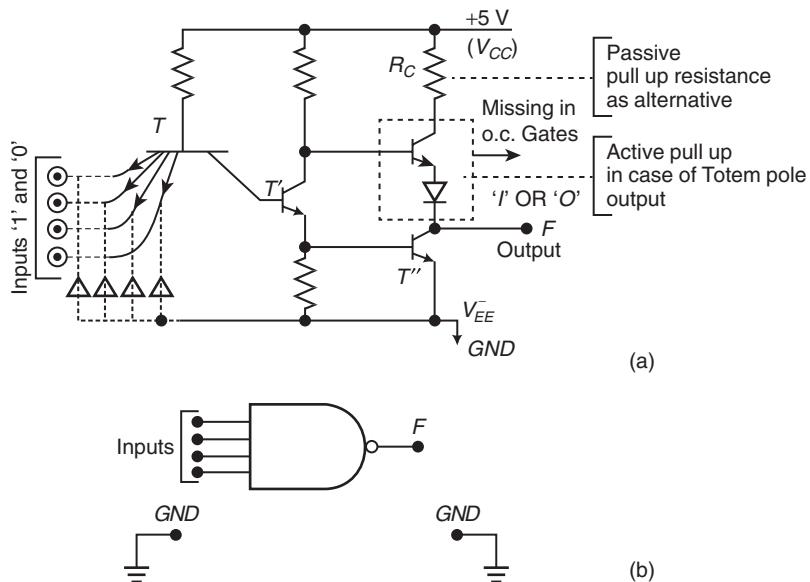


FIGURE 13.6 (a) TTL NAND Circuit with Active pull up (Totem Pole Output) (b) The symbols of four inputs NAND gate.

circuit equivalent to a diode based AND gate circuit at input stages in Figure 13.1(d).

2. The base-collector junction at T and base-emitter junction at T' are common, which performs as the replacement of 2-diodes before T in circuit of Figure 13.5(a). The output at T' emitter is therefore input to the base of the output stage transistor T'' .
3. In case of a TTL with a totempole output (Section 13.4.3.4) there is active pull up of the collector of T'' and output F in place of passive pull up by 2200 Ohm in Figure 13.5(a) DTL NAND circuit. Advantage of active pull up is to reduce the Joule loss (heat loss) within the resistance in passive case. Active pull up increases the loading fan-out (Note: When the active pull up circuit is externally connected the TTL circuit will be called open-collector TTL gate (Section 8.1)).

13.4.3.2 TTL Circuit Working

The TTL circuit working can be explained as follows:

1. **The both inputs 1:** Two or more inputs to two or more n -ends (multi-junction emitter) form multi $n-p$ junctions with the base. When all inputs are 1, all the junctions are reverse biased. The transistor T base-collector is forward biased. The collector +5 V voltage makes the base-emitter of T also forward biased. Since T emitter connects to T'' base, T'' is also forward biased. Both T' and T'' works in saturation mode. $V_{BE}(\text{sat})$ at $T'' \approx 0.7 \text{ V}$ and $V_{BE(\text{sat})}$ at T' is $\approx (0.7 \text{ V} + 0.7 \text{ V})$. Voltage at T base is $\approx (0.7 \text{ V} + 0.7 \text{ V} + 0.5 \text{ V})$. The input voltages at the TTL gates if remain $> 2.4 \text{ V}$, the multi-emitter

junctions will remain reversed biased. T'' collector output at F is 0 in saturation mode of T'' . The output is ~ 0.2 V. The collector of T'' connects to supply through the active pull up. This is expected from a NAND gate that the output is low when all inputs are 1.

2. **Any or both inputs 0:** Two or more inputs to two or more n -ends (multi-junction emitter) at T form the multi $n-p$ junctions with the base. When any of the input is low, the emitter-base junctions will be at threshold voltage and base voltage V_{BE} is $\sim (0.2\text{ V} + 0.7\text{ V})$, transistor T goes to saturation and the output at collector of T is low enough to make base-emitter of T' and base emitter of T'' , which are in series, conduct any significant current. The base currents at T' and T'' are negligibly small. The transistor T'' is in cutoff mode. T'' collector output at F is 1 in cutoff mode. The output is close to the supply voltage 5 V as the collector of T'' connects to supply through an active pull up. This is expected from a NAND gate that the output is high when any input is low.
3. **Output stage for the next input stage:** A common output from the transistor T'' at F is given to an input $n-p$ junction at next TTL stage(s) multi-emitter junction.

13.4.3.3 Wired Logic (Wired AND or Implied AND) in Case of a TTL Gate with Passive Pull Up Connection

If output F 's at two four-inputs TTL gates can be connected, provided there is passive pull up ($\sim 4\text{ k}\Omega$) like $2.2\text{ k}\Omega$ in Figure 13.5(a) in place of active pull up like in Figure 13.6(a) are connected, the output can be considered as AND operation between the logic outputs. Because when both the outputs correspond to cutoff stages of the transistors, the output will remain unaffected and will be 1. When any of the outputs correspond to saturation condition ~ 0.2 V, the output from common point will become 0.2 V. If A, B, C and D are the inputs at one TTL NAND gate and E, F, G and H are inputs at another NAND, the output will be as follows:

$$F = \overline{(A \cdot B \cdot C \cdot D)} \cdot \overline{(E \cdot F \cdot G \cdot H)} = \overline{(A \cdot B \cdot C \cdot D + E \cdot F \cdot G \cdot H)} \quad (\text{Using DeMorgan theorem}).$$

Current dissipation is logic 0 state will increase when two TTL gates with passive pull-ups are ANDed by wired logic. The TTL gates with missing pull up circuit at the collector are also called open collector gates. These are more suitable for the wired logic connections.

13.4.3.4 Totem Pole Output in Case of a TTL Gate with Active Pull Up Connection

Figure 13.6 (a) TTL NAND Circuit is a Totem Pole Output circuit when it has active pull up circuit. Figure 13.6 (a) shows active pull up by combination of a circuit with in a dotted square and a resistance R_C ($\sim 4\text{ k}\Omega$) between the supply and collector of T'' (output F).

Let C is the capacitance between collector of T'' (output at F) and emitter of T'' (supply ground).

- When T'' is in cutoff mode (output high) and changes to output low, the time constant for discharging when T'' goes to saturation mode (output ~ 0.2 V) when all logic inputs becomes 1, *the time constant is very small ($= T''$ output impedance $\times C$) as the T'' conducting stage impedance is very low ($\sim 20\ \Omega$).*
- When T'' goes to off mode (output 1) from the saturation (output 0), the time constant for charging when T'' goes to 1 when any of the logic input becomes 0, *the time constant is high ($= R \times C$) as the T'' is non-conducting stage impedance is very high ($\sim 20\ \Omega$) (charging is through resistance R ($\sim 4\text{ k}\Omega$) when passive pull up exists instead of active pull shown in Figure 13.6(a)).*

Output F when there is active pull up circuit between output F and supply (Figure 13.6(a)) is called Totem Pole Output. It gives an advantage of very low time constant in both cases of transitions (cutoff to saturation and saturation to cutoff) 1 to 0 and 0 to 1 A totem-pole circuit cannot used as wired AND logic (Section 13.4.3.3). Totem pole action is as follows:

- When T'' output is low, the base current is high. The output at collector of T' is at low (~ 0.2 V + 0.7 V). It is not sufficient to cause large current through T'' because F is at ~ 0.2 V and diode between emitter of active pull up transistor and F itself needs ~ 0.7 V. Voltage V_{BE} at active pull transistor T''' is less than the forward bias threshold. Therefore, the time constant is still controlled by ($= T''$ output impedance $\times C$) which is very low. The discharge on high to low transition takes place through T'' only.
- When T'' output is high, the base current is very low in cutoff stage. The output at collector of T' is at high. It is not sufficient to cause large current through T'' because F is also at high and the diode between emitter of active pull up transistor and F itself needs ~ 0.7 V. Voltage V_{BE} at active pull transistor T''' is less than the forward bias threshold. The transition of the diode from above threshold to below threshold state will take time. However the time constant on charging is still small controlled by {(diode impedance in the beginning conduction state + base-emitter saturation state resistance in the beginning) $\times C$,} which is low. These two resistance collectively are $\sim 150\ \Omega$ ($\ll 4\text{ k}\Omega$ in case of use of passive pull up). This is because the when there is a transition from low to high at output F , the diode and T''' will take time to change their conducting state to non-conducting states.

13.4.3.5 TTL Circuit Features

The TTL circuits have the following features:

- Less area is needed on the silicon wafer during its fabrication, which results in its faster speed of operation.
- In totem pole TTL circuit version, there are effectively low impedances during both the output transitions; 1 to 0 transition and 0 to 1 transition, provides us an ability to connect its output to a capacitor. We have the effectively low time constants when connecting to next stage(s) (a capacitor charges as well as discharges slowly if charging and discharging is through high impedance). Such ability is also called totem-pole output ability.

- (iii) In open collector version (Section 8.1), the pull up can be given externally and wired AND logic can be used. The external pull up helps in operation at higher currents or at higher voltages. The external pull up can also be passive or active.

13.4.3.6 Standard TTL Circuit Parameters

TTL circuit parameters in 7400 series are as follows:

Supply $V_{CC} = 5.0 \pm 0.5$ V $V_{EE} = 0$ V

V_{OL} (Voltage Output at logic 0) = 0.4 V and I_{OL} (Current Sink at logic 0) = 4 mA

V_{OH} (Voltage Output at logic 1) = 2.4 V and I_{OH} (Current Output at logic 1) = 0.04 mA

V_{IL} (Voltage Input at logic 0) = 0.8 V

V_{IH} (Voltage Input at logic 1) = 2 V

V_{TH} (Threshold Input Voltage) = 1.3 V [A voltage where the transistor in the circuit changes the mode of working between saturation and cutoff].] Note: Actually used inputs are as specified above to ensure correct performance.

Noise Margin at 1 = 0.4 V (between 2.8 V and 2.4 V)

Noise Margin at 0 = 0.4 V (between 0.4 V and 0 V)

13.4.3.7 Unconnected Input Case

If any input is not connected, the multi-emitter junction of transistor T the base-emitter at that particular junction does not conduct. As the base connects to the supply V_{CC} through 4 K, the corresponding emitter is also at the potential of the base. Hence, the unconnected input in the TTL logic circuit will behave as logic input = 1.

13.4.3.8 TTL Families

TTL circuits are available in different families. Each family has different speeds and power dissipation standards. Six families are Standard, Low power, High speed, Schottky, Low-power Schottky, and advanced low power Schottky. These families have speed corresponding the 10 ns, 33 ns, 6 ns, 3 ns, 10 ns and 5 ns per gate-propagation-delay times, and the power dissipations per gate of 10 mW, 1 mW, 24 mW, 19 mW, 2 mW and 1 mW. The latter five families are abbreviated as L , H , S , LS , and ALS respectively. A reciprocal of a propagation delay is a measure of the speed. The delay time is defined by time taken in the logic transition at the output after the change at the inputs.

If a Schottky diode connects a base-collector in a transistor, the capacitive effects within the transistor when operating in saturation mode reduces significantly. The Schottky diode has a metal end (acting as p -end). It is connected to the base. The n -end connects the collector. This diode switches much faster than the base-collector of the $n-p-n$ transistor. (Figure 13.6(a)). The Schottky diode starts conducting at 0.3 V threshold. Schottky diode based TTL dissipates more power. Therefore, by increasing R from the to the active pull up transistor, we get the low power Schottky version of TTL. We can also have the advanced Schottky (AS) TTL gates to have low power but high speed. We can have advanced low power (ALS) gates to speed but lower power dissipation from AS-TTL.

13.4.4 TTL other than NAND Gate

TTLs other than NAND gates can be made using NAND as a basic building block.

13.5 Emitter Coupled Logic (ECL)

13.5.1 ECL OR/NOR Gate

A Schottky diode gives high speed of operation by faster switching of 0 to 1 transitions in a TTL circuit. Transistors are prevented from becoming saturated during the transition period, for example, by adding Schottky diode between base and collector. ECL is a fast speed solution, which of course needs greater power dissipation.

ECL circuit gives a *current mode logic*, which is based upon the current switching actions. Figure 13.7(a) shows the ECL OR/NOR gate circuit. Figure 13.7(b) shows ECL circuit logic level voltages. Figure 13.7(c) shows the conditions of T and other transistors in different cases. Figure 13.7(d) shows the symbol of two inputs OR/NOR ECL gate.

13.5.1.1 ECL Circuit Internal Connections

This ECL circuit differs from the RTL/DTL/TTL circuits as follows:

1. Each input of the gate gets the inputs at the base of the individual transistors, T_A , T_B , or T_C .
2. There are the ‘OR’ output stage transistor (T_{OR}) and ‘NOR’ output stage transistor (T_{NOR}). The outputs are taken from the emitters of each.
3. There is transistor T_{ref} , which forms a differential amplifier pair with a T in the parallel circuits of T_A , T_B and T_C . The T_{ref} gets the input reference voltage ($V_{REF} = -1.15$ V) from a reference supply circuit. The pairs amplify the difference in the voltages (base currents) between the voltages at the bases of T_A , T_B , T_C and reference T_{ref} .
4. The emitter of T_{ref} and (T_A, T_B, T_C) couples together. (Hence the logic circuit name is emitter-coupled logic, ECL).
5. The emitters of the differential amplifier pair T_{ref} and (T_A, T_B, T_C) connect through a common resistance R_E (≈ 1.18 k Ω) and to the –ve of supply’ V_{EE} (≈ -5 V).
6. The collectors of (T_A, T_B, \dots) are also common. Common-collectors of the differential amplifier pairs connect through a resistance R_{C1} (~ 267 Ω) to the GND (which is +ve with respect to the –ve supply). This part of the circuit operates in common collector amplifier (emitter-follower) mode. The collector of T_{ref} connects to GND through a resistance R_{C2} (~ 300 Ω).
7. Collector of T_{OR} connects to GND in the common collector amplifier mode (also called emitter-follower mode). The emitter gives the output, which also connects to $-V_{EE}$ through a resistance R'_{out} (≈ 1.5 k Ω).
8. Collector of T_{NOR} connects to GND in the common collector amplifier mode. The emitter gives the output, which also connects to V_{EE} through a resistance R''_{out} (≈ 1.5 k Ω).

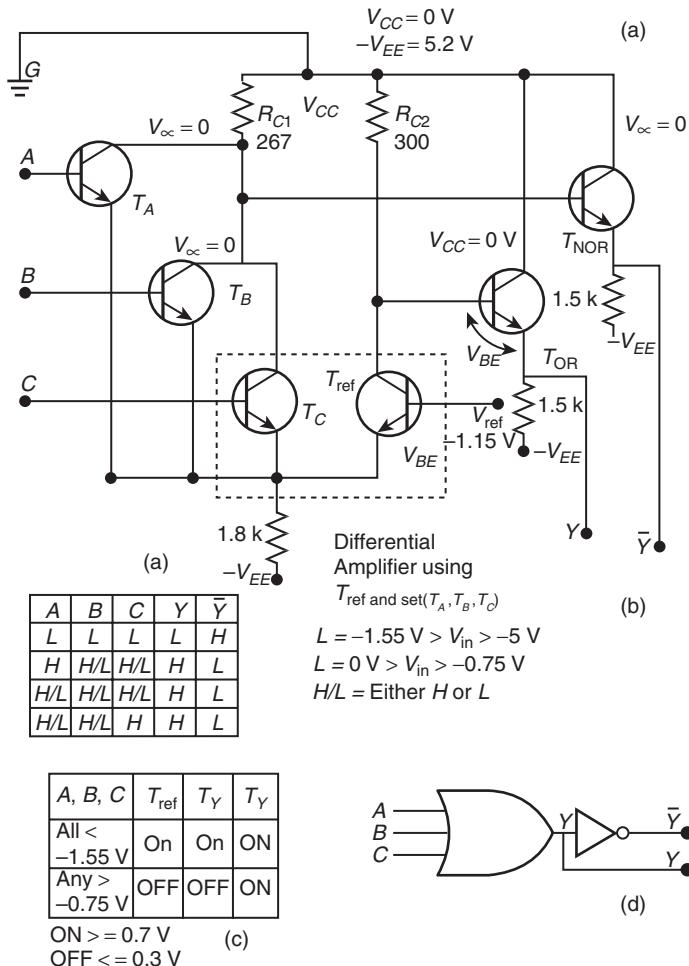


FIGURE 13.7 (a) ECL OR/NOR gate circuit (b) ECL circuit level voltages (c) Conditions of T and other transistors in different cases. (d) Symbol of two-input OR/NOR ECL gate.

13.5.1.2 ECL Circuit Working

The ECL circuit working can be explained as follows:

- Consider a differential amplifier pair between T_{ref} and one of the input-stage transistors. If V_{REF} at base of $T_{ref} = -1.15 \text{ V}$ and input at base of T_i (V_{in}) is -1.6 V (low) T_{ref} emitter has the voltage 0.7 V less than the $V_{REF} = \approx -1.85 \text{ V}$. Since potential difference between the emitter and base of T_i is only 0.25 V , T_i is in cutoff region, while T is in normal inverting amplifier mode. Hence, the collector of T is at -0.9 V and gives sufficient input base current to T_{OR} . The output from T_{OR} emitter is low ($-1.55 \text{ V} - 5 \text{ V}$).
- Consider a differential amplifier pair. If V_{in} at base of $T_i = -0.7 \text{ V}$ (higher than previous case). The transistor T_i is in normal inverting amplifier mode.

The T_i will now be in the cutoff region. Since potential difference between the collector and ground will be 0 V. The output from T_{OR} emitter is high ($0 \text{ V} > 0.75 \text{ V}$).

3. Consider the differential amplifier pairs again. If V_{in} at base of any one or all are at -0.7 V (high). The transistors with high inputs are in normal mode the currents simply add up. The T_i will again be in the cutoff region. Since potential difference between the collector and ground will be 0 V. The output Y from T_{OR} emitter is high.

The ECL circuit works as OR gate at the output of T_{OR} . Since Step 1 and 2 show that the working of T_{ref} and T_A and T_B are opposite. If T_{ref} is in cutoff, then T_p is in normal mode and when all inputs are low. If T_i is in cutoff, then T is in normal mode and when any or several or many inputs are high. Therefore, if collector of T_i is used to drive the T_{NOR} , we shall be obtaining the complementary output \bar{Y} . ECL circuit of Figure 13.7(a) therefore, gives OR as well as NOR outputs.

13.5.1.3 ECL Circuit Logic Voltage Level Parameters

ECL circuit parameters in 10 K series are as follows:

Supply $V_{EE} = -5.2 \text{ V}$

V_{OL} (Voltage Output at logic 0) = -1.7 V

V_{OH} (Voltage Output at logic 1) = -0.9 V

V_{IL} (Voltage Input at logic 0) = -1.4 V

V_{IH} (Voltage Input at logic 1) = -1.2 V

V_{TH} (Threshold Voltage) = -1.29 V (a voltage where the transistor in the circuit changes the mode of working between saturation and cutoff). Note: Actually used inputs are as specified above to ensure correct performance.

Noise Margin at 1 and 0 = 0.3 V (difference of -1.7 V and -1.4 V)

13.5.1.4 ECL Circuit Wired OR Logic

ECL circuit permits wired OR logic if (i) output of transistor T_{NOR} of one ECL circuit in Figure 13.7(a) connects to another ECL circuit output of transistor T_{OR} and (ii) output of transistor T_{OR} of one ECL circuit in Figure 13.7(a) connects to another ECL circuit output of transistor T_{NOR} gates give output $(Y_1 + Y_2)$.

13.5.1.5 ECL Input Unconnected Input Case

If any input is not connected, the transistor T_i base-emitter will be at cutoff. Therefore, it will be taken as low logic level.

13.6 INTEGRATED INJECTION LOGIC (I²L)

A resistance needs larger silicon area on a chip than a transistor or diode. Further, there is a Joule heat loss I^2R in a resistance R carrying current I . If base resistance of 450 Ohm in RTL logic circuit of Figure 13.4(a) is removed, we get a circuit called directly-coupled-transistor-logic (DCTL).

13.6.1 I^2L Circuit Internal Connections

The I^2L circuit (Figure 13.8(a)) differs from the DCTL and RTL circuits as follows:

1. A transistor T or T' base is given the input through a current source within dotted square (Figure 13.8(a)) in place of through the resistor or diode in RTL and DCTL logic circuits.
2. The T and T' emitters are at the GND potential and the collectors connect to a supply potential V_{BB} through an external resistance R_B to each T .
3. The T collector also connects to another transistor T' base.
4. T' base is given another input through another current source.

The transistor T and T' for an I^2L inverter pair.

Figure 13.8(a) shows the basic circuit called Integrated Injection Logic (I^2L).

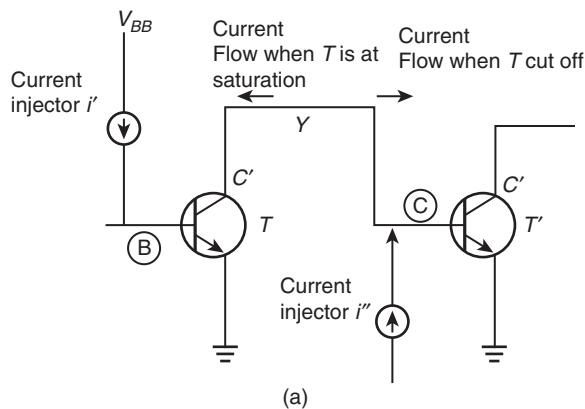
13.6.2 I^2L Circuit Working

The I^2L circuit working is as follows:

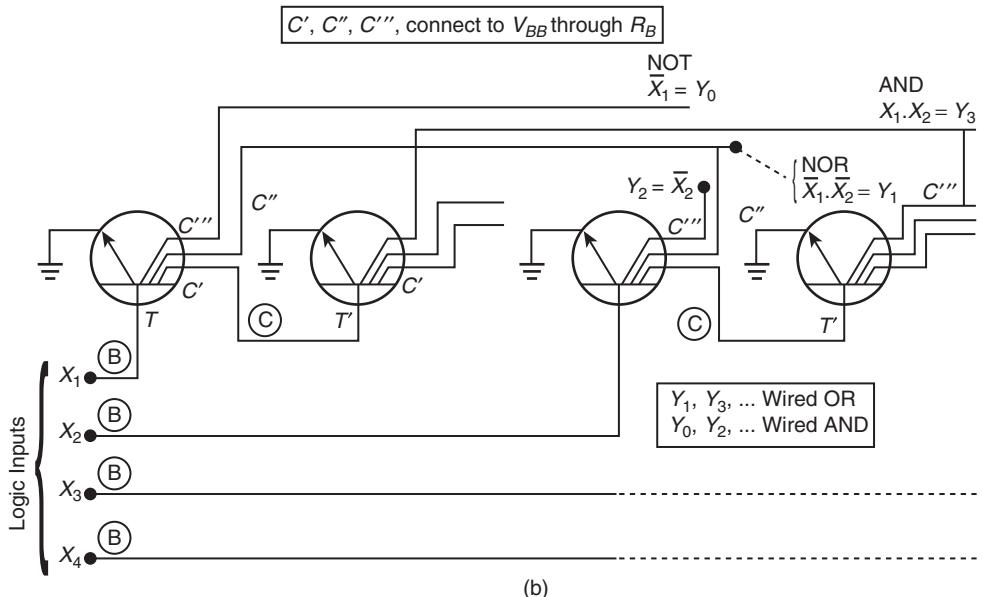
1. When the transistor T base is at logic 0, the injected current i' is 0 and the T is in cutoff mode. Therefore, input current source sinks at the input terminal of next stage T' . Because V_{BE} and thus collector C is at 1 and the current source's current at another input current terminal T' flows (injects) as i'' of second transistor T' . V_{BE} at $T' \approx 0.8$ V. V_{CE} at $T' \approx 0.2$ V. Therefore, logic states at T' collector = 0 and at $T = 1$.
2. When the V_{BE} transistor T base is at logic 1 ~ 0.8 V, the i' injects through T . V_{BE} at $T = 0.8$ V and the T is in saturation mode. V_{CE} at $T = 0.2$ V. Therefore, input current source at T collector sinks the current at T and i'' at T' is 0. V_{BE} at $T' = 0$ V. The collector of T' is at 1 and the T' is in cutoff state.
3. Both transistors T and T' can be made on a silicon as one pair of the multi collector junction transistors. The current sources at the input terminals are made using a $p-n-p$ circuit for each. The emitter of a grounded base $p-n-p$ transistor act as a current injector or sink (if input terminal is at logic 0). The $p-n-p$ collector gets the current through an external resistance connected to $V_{BB} + \text{supply}$. Figure 13.8(b) also shows the input terminals and current source circuits and three collector junctions transistor in place of T and T' and fabrication of the I^2L circuit. Figure 13.8(c) shows the physical placements of the junctions on the silicon.

13.6.3 I^2L Circuit Switching Speed, Delay Times and Power Dissipation

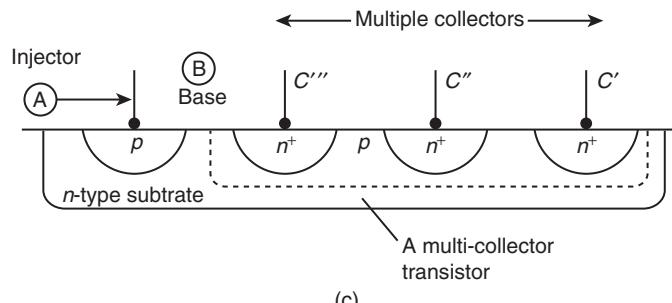
The current source charging current i flows when a base-emitter junction of one of the multi collector junction transistor flow the saturation current. Let base-emitter capacitance is C and saturation resistance is $R_{(\text{sat})}$. (It is very low). The charging time constant is $R_{(\text{sat})}$. Also power dissipation is very small as $i^2R_{(\text{sat})}$ is very small and $R_{(\text{sat})}$ also depends on charging current source. The delay time is inversely proportional to the power dissipation. Smaller delay time means larger power dissipation per gate. Therefore, I^2L gate has high speed high power dissipation.



(a)



(b)



(c)

FIGURE 13.8 (a) Basic circuit of Integrated Injection Logic (βL) (b) An implementation of AND and NOR using the βL circuit by using the multiple collector-junctions in a same transistor (c) The physical placements of the junctions on the silicon for a multicollector transistor.

13.7 HIGH THRESHOLD LOGIC (HTL)

HTL is a high threshold logic based on the modified DTL circuit shown in Figure 13.5(a). The following are the changes done to make an HTL circuit. Figure 13.9 shows a three input HTL NAND circuit obtained after the following modification in the modified DTL.

13.7.1 HTL Connections for the Output at F

- Operational voltages:** The V_{CC} supply is of 15 V in place of 5 V.
- Input stage:** It is based on the application of two or more inputs to two or more n -ends of the $p-n$ junction diodes in place of passive 450 Ohm resistance in RTL circuit. The p -ends are common and connect to 12 k Ω and 3 k Ω resistances R_D and R'_D , which connects to supply voltage of 15 V. The common point of R_D and R'_D , also connects to the collector of a transistor (T). The common point of p -ends also connects to the T 's base. The emitter of the T connects to one Zener diode of 6.9 V breakdown voltage. The p -end of the Zener connects to the transistor (T') base, which also connects to the emitter through 5 k Ω R_B . The emitter of T' is at GND of the supply. The collector of T connects to supply through a resistance R_C of 15 k Ω .
- Output stage for the next input stage:** A common output from the transistor T' at F is given to other diodes at the other next stage(s) logic gates (HTL gates), which will get the input from the transistor, T .

Three input stage diodes and transistor form the load of the next input stage circuit(s). The HTL based NAND and its connection to next stages is shown in the Figure 13.9.

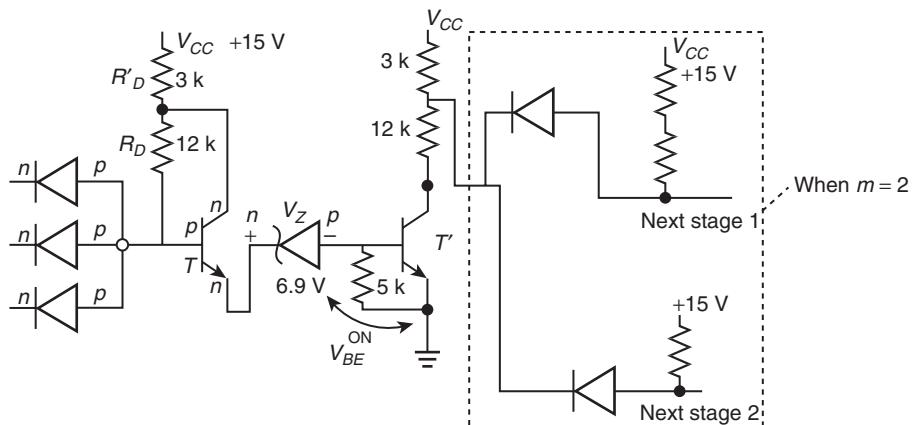


FIGURE 13.9 The HTL based NAND and its connection to next stages.

13.7.2 Logic Operation for the Output at F

If input A or B or C is low ~ 0.2 V (near supply ground), the diode in the path of A or B , respectively will start conducting. Current through the diode will be approximately equal to $(15 \text{ V} - 0.7 \text{ V} - 0.2 \text{ V})/(15000 \text{ Ohm}) = \sim 0.9 \text{ mA}$. (voltage drop across a conducting p - n junction diode is threshold voltage $= \sim 0.7 \text{ V}$.) Voltage drop of 0.9 V connects to the base of T' through two diodes, one formed by base-emitter of T and other formed by Zener needing threshold voltage of 6.9 V and total 7.6 V to turn ON and conduct. Hence the transistor T' base is at $\sim 0 \text{ V}$, below cutoff voltage. When T' is not conducting, the output $F = 1$, high ($\approx 15 \text{ V}$). T' does not conduct when any (or both of the inputs) is low because voltage at the base drops below the cut-off voltage needed at the base.

If all the inputs A , B and C are high ($> 0.7 \text{ V} + 7.6 \text{ V}$), the voltage at common p -ends will start exceeding 7.6 V and the Zener diode circuit to the base will start conducting. When input A , B and C exceeds 8.3 V and the voltage at the common p -ends exceeds $(8.3 \text{ V} + V_{BE(ON)}) = 9 \text{ V}$, the base-emitter junction of T' starts conducting. When the A and B inputs exceeds 7.6 V , the diode stops conduction and when exceeds 8.3 V , becomes reverse biased. $V_{BE(ON)}$ remains at 0.7 V . If transistor T' base-emitter current exceeds a limit, the T' goes in saturation mode and it will start conducting current I_C through R and $V_{CE} = \sim 0.2 \text{ V}$. Therefore, $F = 0$ when A , B and $C = 1$.

Property of a NAND is that its output is 0 when all the inputs are 1. Therefore, the HTL circuit in Figure 13.9 works as a NAND gate.

Calculation of Propagation Delay: Let base-emitter capacitance $= C \text{ nF}$ [nF means nanoFarad.] If $m = 4$ stages, which connects to F , the total capacitance being all T next stages in parallel $= 4C$. Resistance is very small between base and emitter in logic 1 state. Therefore, transistor turn-on delay is small. Resistance in logic 1 state is 15000Ω , therefore turn-off propagation delay $= (15000) mC \text{ ns}$. [$\text{nF} \times \Omega = \text{ns}$.] Typically, the turn-on delay is 90 ns and turn-off delay is 240 ns . Circuit temperature sensitivity is small compared to DTL as the Zener has very small temperature coefficient. The high threshold voltages give a higher noise margin a characteristic of HTL gates.

HTL circuit is appropriate suitable for industrial environment.

13.8 NMOS

There are two types of MOSFETs, n -channel and p -channel. Circuits based on n -channel MOSFETS are called NMOS circuits.

13.8.1 NMOS Circuit Connections and Working

Figure 13.10 shows a NMOS inverter. Enhancement mode or deletion mode n -channel MOSFET T' acts as an active pull up load (in place of R , which occupies larger silicon area). An enhancement mode MOSFET T acts as a inverter and gives an output V_o at F .

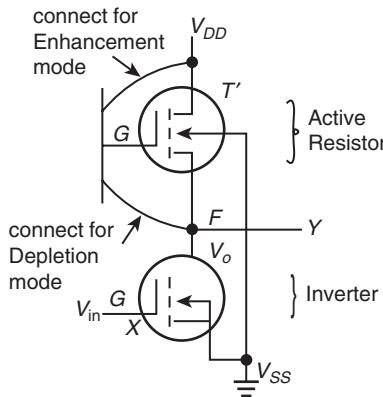


FIGURE 13.10 An NMOS inverter.

Figure 13.11(a) shows a two input NMOS NOR. A depletion mode n -channel MOSFET T'' acts as an active pull up load (in place of R , which occupies larger silicon area). Two enhancement-mode n -channel MOSFETs T_X and $T_{X'}$ are the logic drivers in parallel. These give an output V_o at F through a common point connected to drains of T_X and $T_{X'}$. If both T_X and $T_{X'}$ are off, the output equals supply voltage V_{DD} . If any one is ON, the output at F equals the V_{SS} (the supply GND).

Figure 13.11(b) shows a two input NMOS NAND. A depletion mode n -channel MOSFET T'' acts as an active pull up load (in place of R , which occupies larger silicon area). Two enhancement mode n -channel MOSFETs T_X and $T_{X'}$ are in series. These give an output V_o at F through the upper NMOS drain of $T_{X'}$ which also connects the NMOS MOSFET pull up. If any T_X and $T_{X'}$ are off, the output equals the supply voltage V_{DD} . If both are ON, the output at F equals the V_{SS} (the supply GND).

13.8.2 Calculation of Fan Out

Numbers of logic gates at the next stage(s) that can be loaded are very high due to high input impedance between the gate and channel.

13.8.3 Calculation of Propagation Delay

Let gate-source capacitance = C nF [nF means nanoFarad.] If $m = 40$, the total capacitance being all T_j in parallel = $40 C$. Resistance is very high between gate and source in both logic 0 and logic 1 states. Therefore, MOSFET turn-on delay is large. Now the technology has been developed to get the very small C to get high speeds compatible with the TTLs.

13.8.4 Calculation of Power Dissipation

NAND gate is ON in one of the 4 conditions of inputs defined in NAND truth table. Therefore like TTL, the NMOS NAND dissipates less average power compared to NMOS NOR.

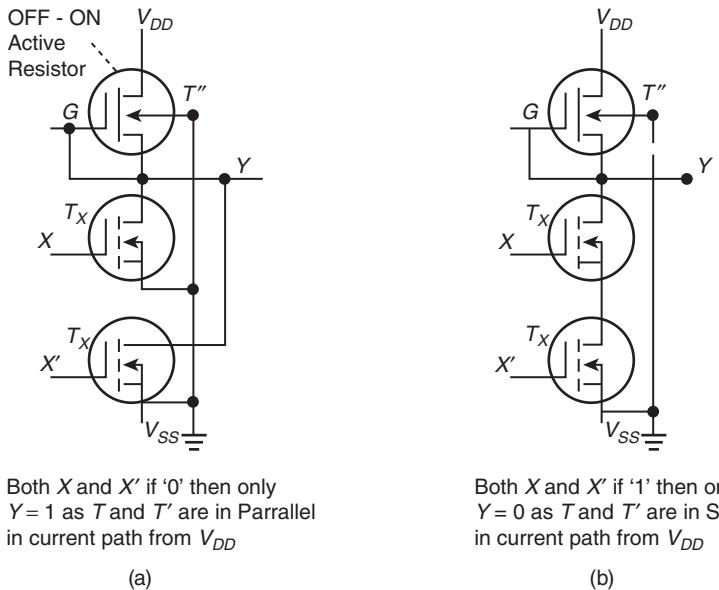


FIGURE 13.11 (a) Two-input NMOS based NOR gate. (b) A two-input NAND NMOS gate.

13.8.5 NMOS Circuit Voltage Levels

NMOS circuit parameters in the 8086 Microprocessor circuit are as follows:

Supply $V_{DD} = 5.0 \pm 0.5$ V $V_{SS} = 0$ V

V_{OL} (Voltage Output at logic 0) = 0.45 V and $I_{OL} = 2$ mA

V_{OH} (Voltage Output at logic 1) = 2.4 V and $I_{OH} = -400$ μ A

V_{IL} (Voltage Input at logic 0) = 0.8 V

V_{IH} (Voltage Input at logic 1) = 2 V

Input Leakage current = ± 10 μ A

Output Leakage current = ± 10 μ A

Noise Margin at 1 = 0.4 V

Noise Margin at 0 = 0.4 V

(Note: Refer Section 13.4.3.6. These parameters are TTL compatible).

13.8.6 Unconnected Input(s) not Permitted

Due to very high gate-source impedance, a small static charge built up can drive a gate voltage higher than the drain. Therefore, the unconnected inputs are not permitted in MOSFET based gates.

13.9 CMOS

13.9.1 Importance and Features of CMOS Logic Circuits

CMOS is most important logic circuit due to the following.

- An enhancement mode p -channel acts as pull up and n -channel enhancement mode MOSFET act as a logic driver (pull down) When n -channel is

ON, the *p*-channel is OFF and when *n*-channel is OFF, the *p*-channel is ON. It means the D.C. (steady state) current dissipation between the supply ends is very small as the series resistance is always very high. Current will flow only during the transitions from 0 to 1 or 1 to 0. Power will dissipate only during the transitions from 0 to 1 or 1 to 0. This feature makes it possible to fabricate large or very large or very-very large-scale integrated circuits (LSI or VLSI or VVLSI) using the CMOS pairs.

- (ii) Since at an instant one of the MOSFET in the pair is ON and has low resistance, the switching speed at charging and discharging is rapid and turn ON delay and turn OFF delay are nearly same.

13.9.2 Operations as Inverter (NOT), NOR and NAND

Figure 13.12(a) shows a CMOS inverter. CMOS (Complementary MOSFET) logic circuit uses an enhancement mode *p*-channel MOSFET T' acts as an active pull up load (in place of R , which occupies larger silicon area) and an enhancement mode *n*-channel MOSFET T acts as a driver of the output V_o at F .

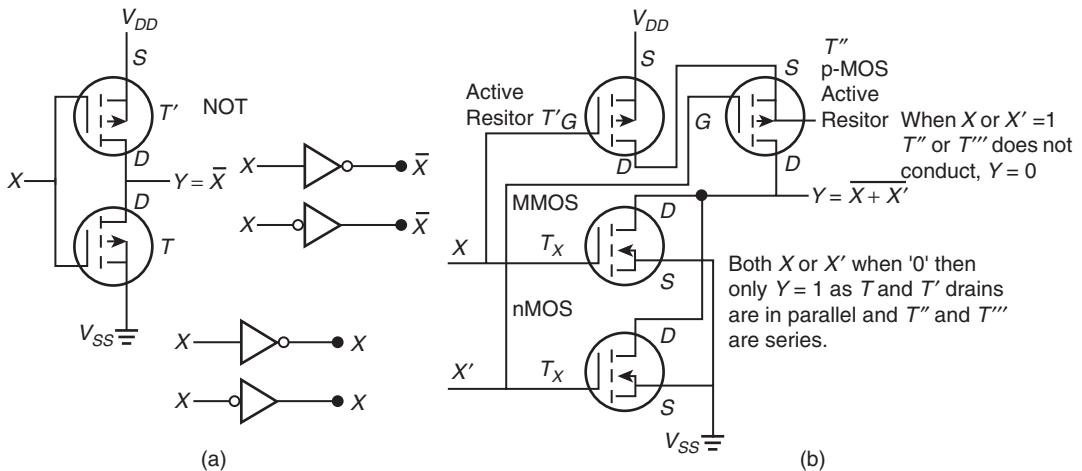


FIGURE 13.12 (a) A CMOS inverter (b) Two input CMOS based NOR gate.

Figure 13.12(b) shows a two input CMOS NOR. Two enhancement mode *p*-channel MOSFETs T' and T'' give the active pull up loads (in place of R , which occupies larger silicon area) to the drains of two *n*-channel logic driver circuits. These two enhancement-mode *n*-channel MOSFETs T_x and $T_{x'}$ are the logic drivers in parallel. These give an output V_o at F through a common point connected to drains of T_F and $T_{F'}$. If both T_F and $T_{F'}$ are off, the output = supply voltage V_{DD} . If any one is ON, the output at F equals the V_{SS} (the supply GND).

Figure 13.13 shows a two input CMOS NAND. Two enhancement *p*-channel MOSFET T' and T'' give an active pull up load (in place of R , which occupies larger silicon area). Two enhancement mode *n*-channel MOSFETs T_x and $T_{x'}$ are in series. These give an output V_o at F through the upper *n*-MOSFET drain of $T_{x'}$, which also

connects the *p*-MOSFET pull up. If any T_X and $T_{X'}$ are off, the output equals supply voltage V_{DD} . If both are ON, the output at F equals the V_{SS} (the supply GND).

13.9.3 Calculation of Fan out

Numbers of logic gates at the next stage(s) that can be loaded are very high due to high input impedance between the gates and the channels at the logic drivers. There are only capacitive driving loads. Steady state D.C. power dissipation is extremely small. Impedance between gate and source is capacitive. Current flows at the transitions only.

13.9.4 Calculation of Propagation Delay

Let gate-source capacitance = C nF (nF means nanoFarad). If $m = 40$, the total capacitance being all T_j in parallel = $40C$. Resistance is very high between gate and source in both logic 0 and logic 1 states but the charging and discharging occurs from the input logic gate rapidly in both 0 to 1 and 1 to 0 transitions (since at an instant one of the MOSFET in the pair is ON and has low resistance, the switching speed at charging and discharging is rapid and turn ON delay and turn OFF delay are nearly same). Therefore, CMOS MOSFET logic circuit turn-on and turn off delay is little compared to NMOS turn-on delay. Now the technology high speed CMOS (HCMOS) has been developed to get very small C to get the large speeds.

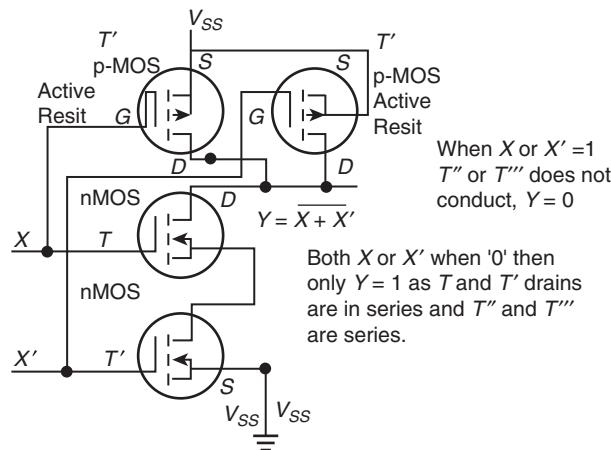


FIGURE 13.13 Two-input NAND CMOS gate.

13.9.5 Calculation of Power Dissipation

When *n*-channel is ON, the *p*-channel is OFF and when *n*-channel is OFF, the *p*-channel is ON. It means the D.C. (steady state) current dissipation between the supply ends is very small as the series resistance is always very high. Current will flow only during the transitions from 0 to 1 or 1 to 0. Power will dissipate only during the transitions from 0 to 1 or 1 to 0. Therefore, unlike like TTL and NMOS, the CMOS logic gates dissipates very little power compared to NMOSs and TTLs and dissipate power only at the transitions of logic states, not in d.c. state.

13.9.6 CMOS Circuit Voltage Levels

CMOS circuit parameters are as follows:

Supply $V_{DD} = 5$ V operating version or 3 to 16 V operating version and $V_{SS} = 0$ V

V_{OL} (Voltage output at logic 0) = $(1/3) V_{DD}$ and $I_{OL} = 0$

V_{OH} (Voltage output at logic 1) = V_{DD} and $I_{OH} = 1$ mA (4000B series)

V_{IL} (Voltage input at logic 0) = $V_{SS} = 0$ V

V_{IH} (Voltage input at logic 1) = $(2/3) V_{DD}$

V_{TH} (Threshold Voltage) = $(1/2) V_{DD}$ (A voltage where the MOSFET in the circuit change the mode of working between ON and OFF). Note: Actually used inputs are as specified above to ensure correct performance.

Noise Margin at '0' and '1' = $-[(1/3) V_{DD} - (1/2) V_{DD}]$.

13.9.7 MOS Logic Circuits (CMOSs) and Their Relative Advantages with Respect to TTLs

Table 13.1 gives a comparison of a CMOS with that of a TTL digital circuit.

For a complex integrated circuit (IC) which incorporates millions or more CMOS transistors, the MOSFETs being field effect devices dissipates a total power which is very small compared to the ICs based on the BJTs. This is very significant property of CMOS digital circuits. In place of BJTs based TTLs in an integrated circuit on a silicon chip, the CMOS digital circuits are used in the complex ICs. For this reason including the reasons mentioned in various rows of Table 13.1 the CMOS IC logic gates and digital circuits gained popularity since 1975 at the expense of the TTL circuits. TTL circuits offer the only benefits of (i) higher speed of operation (Tables 13.2 and 13.1) and (ii) the permission to float the inputs (Table 13.1). Very recent advances in the MOSFET fabrication technology (HCMOS) have made it feasible to get the high-speed operations from the CMOS gates also.

13.10 MEANINGS OF SPEED, PROPAGATION DELAY, OPERATING FREQUENCY, POWER DISSIPATED PER GATE, SUPPLY VOLTAGE LEVELS, OPERATIONAL VOLTAGE LEVELS THAT DEFINE LOGIC STATES 1 AND 0

- Speed:** It means how many times per second, a logic gate is able to respond to the change in the inputs and give the output as per specified logic.
- Propagation Delay:** Propagation delay for a logic output from a logic gate means the time interval between change in a defined reference point input voltage and reflection of its effect at the output. It can also be defined as the time interval between changes in a defined logic level input and reflection of its effect at the output logic level. [Note: The slightly or significantly different values of the delays may be obtained when the input change from 0 to 1

TABLE 13.1

CMOS	TTL
At an input or output, 1 means a state of a potential difference (p.d.) between $(V_{DD} - V_{SS})$ and $0.66(V_{DD} - V_{SS})$. 0 means a state of a p.d. between $0.33(V_{DD} - V_{SS})$ and V_{SS} . $(V_{DD} - V_{SS}) = 3$ V to 16 V. Typical values; $V_{DD} = 5$ V, V_{SS} GND Potential (0 V). Battery operations are feasible.	1 means a p.d. between 5 V and 2.4 V at an output or between 5 V and 2.0 V at an input. 0 means a p.d. between 0.8 V and GND at an input or between 0.4 V and GND at an output. V_{CC} (Figure 13.6) is $5\text{ V} \pm 0.25$ V. Power supply impedance is to be less than one tenth Ohm up to the operating frequencies. Battery operations therefore difficult.
Negligible dissipation at steady state of its inputs. Only about 1 mW per gate per MHz in 4000 series. Power consumption depends upon number of times an input changes i.e. upon the operating frequency.	Power consumption per gate as per Table 13.2.
An enhancement mode <i>p</i> -channel acts as pull up and <i>n</i> -channel enhancement mode MOSFET act as a logic driver. When <i>n</i> -channel is ON, the <i>p</i> -channel is OFF and when <i>n</i> -channel is OFF, the <i>p</i> -channel is ON. It means the d.c. (Steady state) current dissipation between the supply ends is very small as the series resistance is always very high. Current will flow only during the transitions from 0 to 1 or 1 to 0 . Power will dissipate only during the transitions from 0 to 1 or 1 to 0 . This feature makes it possible to fabricate large or very large or very-very large scale integrated circuits (LSI or VLSI or VVLSI) using the CMOS pairs.	An <i>n-p-n</i> transistor and a <i>p-n</i> diode form the pull up circuit in totem pole output TTLs. The <i>n-p-n</i> BJT act as a logic driver. At each saturation mode operation of the BJT, the current will flow. Significant power will dissipate during the transitions from 0 to 1 or 1 to 0 as well during steady state with output logic state = ' 0 '. This feature inhibits to use the TTLs to fabricate large integrated circuits (LSI) and use is restricted to the interfacing of the logic signals between the processor and memories, and peripherals.
Since at an instant one of the MOSFET in the pair is ON and has low resistance, the switching speed at charging and discharging is rapid and turn ON delay and turn OFF delay are nearly same.	Since in cutoff region the resistance in the charging path is large, the turn ON delay and turns OFF delay are widely different in passive pull up TTL and almost same in totem pole TTL with active pull up circuit at the output.
Gained popularity after 1975 and now almost universally used.	Very popular up to 1985. Loosing popularity exponentially after 1975.
Inputs are not permitted to float (disconnected), and must be connected to state 1 or 0 , or to V_{DD} or to V_{SS} .	Inputs are permitted to float (disconnected) as the transistor T' (Figure 1a) always has a complete circuit (internally) for the currents. Unconnected input logic state is 1 .
Output can be as per p.d. $(V_{DD} - V_{SS})$ (Figure 13.12(a)) i.e. fixed at state 0 or 1 as per $(V_{DD} - V_{SS})$. Any Volt age either up to one third of $(V_{DD} - V_{SS})$ or above two third of it can be there.	Output can be below V_{CC} of $(5\text{ V} \pm 0.25\text{ V})$ at state 1 and can be down to 2.4 V. For state ' 0 ', the output does not exceed 0.8 V.
Inputs and outputs are permitted to be kept at state 1 or 0 as per user wish as steady state current is negligibly small. Direct connections to V_{DD} or V_{SS} at input permitted.	It is better to keep a state 1 because in that case the margin of Voltage fluctuation is $(5\text{ V} - 2.8\text{ V})$ i.e. 2.2 V at an output, and is 2.6 V at an input. At a state 1 , the steady state current drawn by a transistor at the output is nominal around 50 mA, while at ' 0 ' it is much higher. Direct connection is not done to V_{CC} but only through a resistance above $1.1\text{ k}\Omega$.

13.32 Digital Systems: Principles and Design

IC logic gate chips start with 40, 41 or 45 with no following nonnumeric character or characters. IC logic gates also start with prefix 74HC in a high speed CMOS version that is followed by numeric numbers analogous to that of 74 series of TTL logic circuit. HCT is used when TTL compatibility is desired from the CMOS logic gates.	IC chips based upon TTL logic start with prefix 74 can be followed by an abbreviation as per Table 13.2.
Maximum operating frequency may be often small, for example for a 40...B it is 2.5 MHz. (Very high in HCMOS)	It depends upon TTL family as per Table 1.2. It can be as high as 100 MHz.
Propagation (speed): For 40...B it is 200 ns, for 74C it is 90ns and for 74HC it is 12ns.	It depends upon TTL type as per Table 1.2
Larger Voltage changes permitted, noise immunity very high compared to TTL.	Small Voltage changes permitted i.e. noise immunity is less than CMOS.

and change from 1 to 0. We can take average propagation delay. Also a propagation delay is also subject to variations in power supply and temperature. We can then define a statistical deviation and an average.]

3. **Operating Frequency:** It means how many times logic levels changes per second are permitted without affecting the logic gate characteristics outside the limits, which have been set for the propagation delays, voltage levels and power dissipation per gate.
4. **Power Dissipation Per Gate:** It means how much average power dissipates per gate when a logic circuit is operated within the specified operating frequency.
5. **Voltage Levels that Define Logic States 1 and 0:** A logic level 1 at output is defined by voltage levels V_{OH} maximum and V_{OH} minimum. A logic level 1 at input is defined by voltage levels V_{IH} maximum and V_{IH} minimum. A logic level 0 at output is defined by voltage levels V_{OL} maximum and V_{OL} minimum. A logic level 0 at input is defined by voltage levels V_{IL} maximum and V_{IL} minimum.
6. **Threshold Logic Input Voltage:** A voltage where the transistor in the circuit changes the mode of working between saturation and cutoff. *Actuary used inputs are as specified above to ensure correct performance.*

13.11 SPEED, PROPAGATION DELAY, OPERATING FREQUENCY, POWER DISSIPATED PER GATE, SUPPLY VOLTAGE LEVELS, OPERATIONAL VOLTAGE LEVELS THAT DEFINE LOGIC STATES '1' AND '0' FOR VARIOUS FAMILIES OF GATES

Table 13.2 gives the abbreviation for a family, measure of speed per gate and power dissipation of per gate for these families of the lowlogic gates.

TABLE 13.2

S. No.	'Family name'	Abbreviation for the family (other than standard)	Propagation delay ns	Power dissipated per gate mW	Maximum operating frequency MHz
1	Standard TTL	-	10	10	30
2	Low Power TTL	L	33	1	3
3	High Speed TTL	H	6	24	50
4	Schottky TTL	S	3	19	100
5	Low Power Schottky TTL	LS	10	2	30
6	Advanced Low Power Schottky TTL	ALS	5	1	35

Table 13.3 gives the propagation delay, power dissipated per gate in mW, speed-power dissipation product in pW.s (pJ), fan-out and maximum operating frequency in MHz.

TABLE 13.3

S. No.	Type of gate	Propagation delay ns	Power dissipated per gate mW	Speed power product pW.s = pJ (Pico Joule)	Fan out	Maximum operating frequency MHz	Noise immunity
1	RTL	12	12	144	5	8	Average
2	DTL	30	10	300	8	72	Good
3	I^2L	25-250	0.006 to 50 mW	$\approx 1\text{pJ}$ or less Current Source Dependent	Injection	Very High	Small
4	HTL	90	50	5000	10	100	Excellent
5	ECL (10K Series)	2	50	100	25	≈ 75	Small
6	NMOS in 8085 type VLSI	300	0.2 to ~ 10	60	20	2	Good
7	Standard TTL	10	10	100	10	35	Very good
6	CMOS in 74HC series	18	2.5 mW				
	static*, 600 mW/MHz	20	60	Very Good			
	static, ~ 10 at 1 MHz						

*In steady state, only leakage current flows. It is $\sim 10^{-5}\text{mA}$

■ EXAMPLES

Example 13.1

In an RTL logic circuit, $V_{BE(ON)}$ is 0.7 V and input Voltage is 2.4 V. What is the base current when the resistance in base-emitter circuit is 1 k Ω ?

Solution

We use the equations $I_B = (V_i - V_{BE(ON)})/R_B = (2.4 \text{ V} - 0.7 \text{ V})/1000 \Omega = 1.7 \text{ mA}$.

Example 13.2

In an RTL logic circuit, $V_{BE(ON)}$ is 0.7 V and β forward current gain is 20. What will be collector current in saturation state if $I_B = 1 \text{ mA}$?

Solution

We use the equation $I_C = \beta I_B = \beta(V_i - V_{BE(ON)})/R_B = 20 \times 1 \text{ mA} = 20 \text{ mA}$

Example 13.3

In an RTL logic circuit, find voltage output at the logic gate if $V_{BE(ON)}$ is 0.7 V and β forward current gain is 10 and collector circuit resistance is 0.5 kW. Assume V_{CC} is 5 V and base resistance is 5 k Ω at logic Input = 3.7 V.

Solution

We use the equation

$$\begin{aligned} V_o &= V_{CC} - I_C R = V_{CC} - R \cdot \beta(V_i - V_{BE(ON)})/R_B \\ &= 5 \text{ V} - (500 \text{ Ohm} \times 10 \times (3.7 \text{ V} - 0.7 \text{ V})/5000 \text{ Ohm}) \\ &= 5 \text{ V} - 3 \text{ V} = 2 \text{ V}. \end{aligned}$$

Example 13.4

A triplet of diode has p -ends common at Y' and the n -ends are for the inputs A , B and C (Figure 13.5(a)). The triplet p -end connects to 5.0 V supply through 1 k Ω . It also connects to voltage output Y through a $p-n$ diode in series with n -end of it towards Y . Find the output at Y under different conditions given in the truth table (Table 13.4). Y connects to other loading circuits, the effect of which can be assumed to be negligible. Assume 1 => 4.4 V input and 0 = 0.2 V. Assume all diodes as silicon diodes with 0.7 V threshold voltage.

Solution

TABLE 13.4 Table for logic inputs and voltages after solving the problem

Output Y' at common of all 3 diodes				
A	B	C	Supply current through 1 k Ω	Output at Y
0	0	0	(+ .2 V + .7 V)	.2 V
0	0	1	(+ .2 V + .7 V)	.2 V
0	1	0	(+ .2 V + .7 V)	.2 V
0	1	1	(+ .2 V + .7 V)	.2 V
1	0	0	(+ .2 V + .7 V)	.2 V
1	0	1	(+ .2 V + .7 V)	.2 V
1	1	0	(+ .2 V + .7 V)	.2 V
1	1	1	> 4.4 V	4.3 V to 3.7 V

We draw the circuit for the problem and analyze the circuit as under:

When any of the input = 0 [0.4 V] the diode connected to it is above threshold and voltage at the p -end clamps to $(0.7 \text{ V} + 0.2 \text{ V}) = 0.9 \text{ V}$ as 0.7 V is the drop across

this diode and 0.2 V is the input. The output Y will be 0.7 V less than the voltage at the common p -ends. Therefore $Y' = 0.2$ V.

When all of the inputs = 1 [> 4.4 V] the diode connected to it is cut-off and voltage at the p -end clamps to 5.0 V as 0.6 V is less than the threshold voltage 0.7 V for any of the triplet diodes to conduct. The output Y' will be 0.7 V less than the voltage at the common p -ends. Therefore, voltage at $Y' = > 4.4$ V and at Y is between 1.3 V to 3.7 V.

Current through the resistance is 0 in case of all inputs = 1 as there is no conduction path available. Current is $(5\text{ V} - 0.9\text{ V})/1\text{ k}\Omega = 4.1\text{ mA}$ when conduction path available when the inputs = 0.

Using above solution, now we can fill the columns 4 and 5 Table 13.4 as Table 13.5 as the answer (Table 13.4 also shows Y' and Y potentials).

TABLE 13.5

A, B and C input states	Output at common of all 3 diodes	Supply current in mA through 1 kΩ	Output at Y in Volt
Any input = '0' = 0.4 V	0.9 V	4.1	0.2 V
All input = '1' = > 4.4 V	5.0 V to 4.4 V	0	4.3 V to 3.7 V

Example 13.5 Find how the current will distribute in the diodes of the triplet.

Solution

When any of the input is at 0 = 0.4 V and any diode of triplet is conduction and in case more than one input is at 0, the current through R will distribute among those diodes which connect to input 0. Hence Table 13.4 can be redesigned as following Table 13.6.

TABLE 13.6 Table for input stages and voltages

A	B	C	Output at common of all 4 diodes in Volts	Through A current in mA	Through B current in mA	Through C current in mA	Supply Current in mA through 1 kΩ	Output at Y in Volt
0	0	0	1.1	1.3	1.3	1.3	3.9	0.4
0	0	1	1.1	1.95	1.95	0	3.9	0.4
0	1	0	1.1	1.95	0	1.95	3.9	0.4
0	1	1	1.1	3.9	0	0	3.9	0.4
1	0	0	1.1	0	1.95	1.95	3.9	0.4
1	0	1	1.1	0	3.9	0	3.9	0.4
1	1	0	1.1	0	0	3.9	3.9	0.4
1	1	1	5.0	0	0	0	0	4.3

Note: 1.1 V because $0.4\text{ V} + 0.7\text{ V} = 1.1\text{ V}$. 1.3 mA because $(5\text{ V} - 1.1\text{ V})/1\text{ k}\Omega = 3.9\text{ mA}$ and $3.9\text{ mA}/3 = 1.3\text{ mA}$ and $3.9\text{ mA}/2 = 1.95\text{ mA}$.

13.36 Digital Systems: Principles and Design

Example 13.6 Find what will be output Y if the next stage circuits do not load the output.

Solution

When Y is not loaded, the diode between Y' and Y will not conduct. Hence output will be same as that at common triplet p -end Y . Table 13.7 gives the output.

TABLE 13.7

A, B and C state	Output at common of all 4 diodes in Volt	Output at Y in Volt
Any input = 0	1.1	1.1
All inputs = 1	5.0	5.0

Example 13.7 If in Example 13.4 another triplet of diodes (A' , B' , C') connect and give the output Y'' , what will the output if we connect Y and Y'' .

Solution

This will correspond to wired AND logic. Therefore the output will

$$Y \cdot Y'' = (A + B + C) \cdot (A' + B' + C')$$

Example 13.8 Consider DTL logic circuit of Figure 13.5(a) with a fan-out of 8. If effective Capacitance effect of next stage DTL is 0.04 nF, what will be maximum possible propagation delay in the output change at the next stage?

Solution

Base-emitter capacitance = 0.04 nF. Assume that maximum number of next stages permitted is present. Therefore, $m = 8$. When the output = 1, the next stage transistor-ON delay is small and is $0.04 \text{ nF} \times 8 \times 150 \Omega = 48 \text{ ns}$. When logic state output becomes 0, the discharging (due to 1 to 0 transition) resistance is 5000 Ohm, therefore turn OFF propagation delay = $(5000 \text{ Ohm}) \times 8 \times 0.04 \text{ nF} = 1600 \text{ ns}$. Average delay = $(1648/2) = 824 \text{ ns}$. Average delay per gate next stage gate added = $(824/8) = 1103 \text{ ns}$.

Example 13.9 Consider DTL logic circuit of Figure 13.5(a) with a fan-out of 8. What is V_{CE} at saturation? What is V_{CB} at saturation?

Solution

In saturation, the output at F is 0.2 V. Hence $V_{CE} = 0.2 \text{ V}$. Therefore, $V_{CB} = V_{BE(\text{sat})} = -0.8 \text{ V} - 0.2 \text{ V} = -0.6 \text{ V}$.

Example 13.10 Consider circuit of Figure 13.5(a)/Let β (forward common emitter gain) is 20 and $I_B = 0.4 \text{ mA}$. Fan out = 8. Whether will the collector current keep the output stage transistor in saturation or not?

Solution

Let us recall current-sink logic condition. T will remain in saturation if collector current is $I_c < (20 \times 0.4)/8 \text{ mA} = 1 \text{ mA}$. Actual collector current is $(5 \text{ V} - 0.2 \text{ V})/5000$. This is less than 1 mA. Therefore, fan out of 8 is permitted.

Example 13.11

Consider circuit of Figure 13.5(a). If a circuit sink $I_C = 4 \text{ mA}$ when output is 0 and source $400 \mu\text{A}$ when '1', what is the average collector current and average power dissipated.

Solution

1. Average Current = $(4 \text{ mA} + 0.4 \text{ mA})/2 = 2.2 \text{ mA}$.
2. Average Power Dissipated = $5 \text{ V} \times (\text{Average current}) = 5 \text{ V} \times (4 \text{ mA} + 0.4 \text{ mA})/2 = 11 \text{ mW}$.

Example 13.12

Consider TTL circuit of Figure 13.6(a). What is the value of active pull totem pole circuit resistance between supply and collector, if logic stat 0 sink current is 2 mA. What is the value of active pull totem pole circuit resistance between supply and collector, if logic state 1 total current to next stage is maximum 0.04 mA/gate ? Assume fan-out = 10.

Solution

1. At logic 0 output, the transistor is in saturation and is at the voltage $\sim 0.2 \text{ V}$. Total Current = $2 \text{ mA} = (5 \text{ V} - 0.2 \text{ V})/R$. Therefore, $R = 4.8 \text{ V}/2.0 \text{ mA} = 2.4 \text{ k}\Omega$.
2. At logic '1' output, the transistor is cutoff and is at voltage between 5 V and $\sim 2.4 \text{ V}$. Total Current = $10 \times 0.04 \text{ mA} = (5 \text{ V} - 2.4 \text{ V})/R$. $R = 3.6 \text{ V}/0.4 \text{ mA} = 9 \text{ k}\Omega$.

Example 13.13

In the above example, what is turn-on 0 to 1 transition delay time? What is turn-off 1 to 0 transition delay time? What is average delay time? Assume $C = 0.04 \text{ nF}$.

Solution

For 0 to 1 transition, the R is to be taken as $2.4 \text{ k}\Omega$ only because the active pull up activates only after a turn-on transition to '1'. So delay time = $0.04 \text{ nF} \times 10 \times 2.4 \text{ k}\Omega = 9.6 \text{ ns}$. For 1 to 0 transition, the R is also to be taken as $2.4 \text{ k}\Omega$. So delay time when discharging of next stage input terminals is again = $0.04 \text{ nF} \times 10 \times 2.4 \text{ k}\Omega = 9.6 \text{ ns}$. Average delay time remains same.

Example 13.14

Consider the ECL logic circuit of Figure 13.7(a). If two input stage transistors are given the p.d of -1.5 V each, and reference to base of other differential pair transistor is -1.15 V , show that the input stage transistor are in the cutoff region. If one of the input stage now become at -0.5 V , what will be changes in the outputs.

Solution

Consider a differential amplifier pair between T_{ref} , one of the input-stage transistors, T_i and another input stage transistor T_j . V_{REF} at base of $T_{\text{ref}} = -1.15$ V. Input at base of T_i and T_j are -1.5 V. T_{ref} emitter has the voltage 0.7 volt less than the V_{REF} . Hence voltage at the emitter of T_{ref} is -1.85 . Therefore, the potential difference between the emitter and base of T_i is only 0.35 V so T_i is in cutoff region. Similarly, T_j is in cutoff region while T is in normal inverting amplifier mode. Hence, the collector of T_{ref} is at -0.9 V and gives sufficient input base current to T_{OR} . The output from T_{OR} emitter is low (Figure 13.7(b)).

When one of the inputs (to T_j) becomes -0.5 V, the p.d. between the base and emitter is now greater than 0.7 V, hence the corresponding transistor operates in normal input mode. The voltage at the emitter of T_j will now drive T into the cutoff region. Hence, the collector of T is at does not give sufficient input base current to T_{OR} . The output from T_{OR} emitter is high (Figure 13.7(b)).

Example 13.15

Consider a circuit of Figure 13.8. (1) What will be injected current if V_{BB} is 2 V, V_{EB} is 0.7 V and R external is 100 k Ω . (2) If logic input to the base of inverting transistor amplifier is 0 , what will be collector current in transistor T ?

Solution

1. Since p.d. across R external is $(V_{BB} - V_{EB})$, the injected current $= (2 \text{ V} - 0.7 \text{ V}) / 100 \text{ k}\Omega = 0.013 \text{ mA}$.
2. Since logic input is 0 , the current of 0.013 mA just sinks and base current at T is 0 . Hence, Collector current at T is negligible because T is at cut-off.

Example 13.16

Draw a circuit to implement AND operation using n -channel enhancement MOSFETs alone.

Solution

Refer Figure 13.11(b), which shows an n -MOS NAND. AND operation is complement of NAND and complement is obtained by common input NAND. Therefore if the output F of first NAND in this figure is connected to a NOT or another identical NAND with both the inputs connected to F , the output of that NAND will be AND operation of two inputs at the first NAND. Figure 13.14 shows a NAND-NOT based circuit.

Example 13.17

Draw a circuit to implement AND operation using CMOS pairs.

Solution

Refer Figure 13.13(b), which shows a CMOS NAND. AND operation is complement of NAND and complement is obtained by common input NAND. Therefore if the output F of first NAND in this figure is connected to NOT or another identical NAND with both the inputs connected to F , the output of that NAND will be AND operation of two inputs at the first NAND. Figure 13.15 shows a NAND-NOT based circuit.

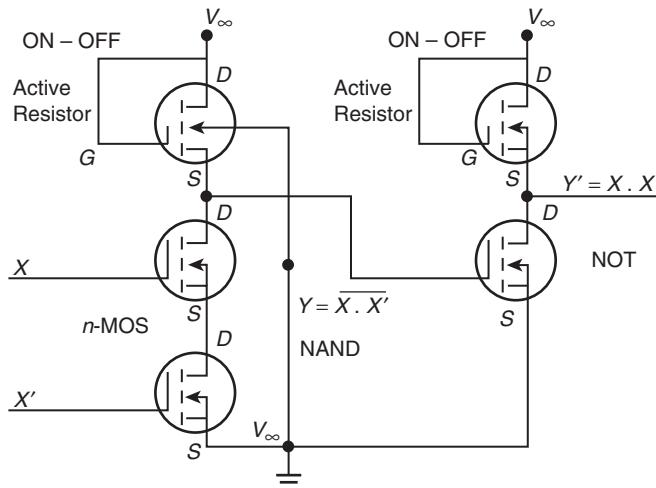


FIGURE 13.14 Two-input NMOS based AND gate using a NAND-NOT.

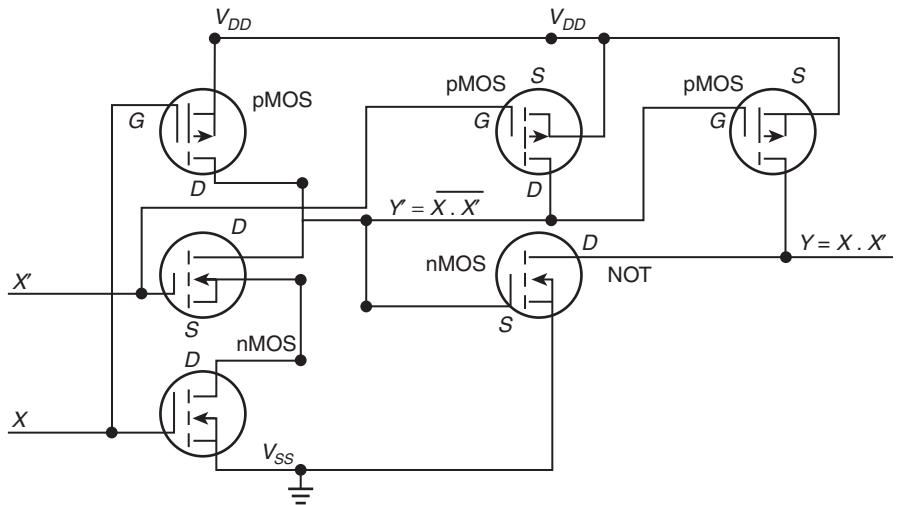


FIGURE 13.15 TWO-input CCMOS based AND gate using a NAND-NOT.

Example 13.18

A typical CMOS logic circuit dissipates $2.5 \mu\text{W}$ static, $600 \mu\text{W}/\text{MHz}$. What is the average power dissipation assuming maximum operational frequency of 10 MHz ? What will be the power dissipated at 500 kHz operation?

- At 100 MHz , the power dissipation will be $(2.5 \mu\text{W} + 6000 \mu\text{W})$. Average dissipation assuming 50% time static and 50% maximum rate operation = $3001.5 \mu\text{W}$.
- Since power dissipation is $600 \mu\text{W}/\text{MHz}$, at 500 kHz , it will be half. It means $300 \mu\text{W}$.

Example 13.19 What are the correct characteristics among the DTL, ECL, CMOS and TTL gates? (A) or (B) or (C) or (D) [A GATE (2003) Competition Examination Question].

(A) Minimum Fan-out DTL, Minimum Power Dissipation TTL and Minimum Propagation Delay CMOS (B) Minimum Fan-out DTL, Minimum Power Dissipation CMOS and Minimum Propagation Delay ECL (C) Minimum Fan-out TTL, Minimum Power Dissipation ECL and Minimum Propagation Delay TTL (D) Minimum Fan-out CMOS Minimum Power Dissipation DTL and Minimum Propagation Delay TTL

Solution

Using Table 13.3, we find that (B) is correct.

Example 13.20 What are the correct characteristics among the I^2L , ECL, CMOS and TTL gates? (A) or (B) or (C) or (D)

(A) (i) Noise immunity excellent TTL, (ii) Ten MHz operational case minimum power dissipation CMOS and (iii) Minimum propagation delay I^2L (B) (i) Noise immunity excellent I^2L , (ii) Ten MHz operational case minimum power dissipation TTL and (iii) Minimum propagation delay CMOS (C) (i) noise immunity excellent CMOS, (ii) Ten MHz operational case minimum power dissipation ECL and (iii) minimum propagation delay TTL (D) (i) Noise immunity excellent CMOS, (ii) Ten MHz operational case minimum power dissipation I^2L and (iii) Minimum propagation delay ECL

Solution

Using Table 13.3, we find that (D) is correct.

■ EXERCISES

1. In an RTL logic circuit, if input resistance $R = 5 \text{ k}\Omega$ (in place of 450Ω in Figure 13.4) and collector path resistance $= 1 \text{ k}\Omega$ (in place of 640Ω). Calculate the noise margins at 1 and at a 0 assuming $\beta_F = h_{fe} = 20$, $V_{CE(\text{ON})} = 0.2$, $V_{BE(\text{sat})} = 0.7 \text{ V}$. Also calculate the fan-out of the circuit and propagation delay assuming a next stage effective capacitance $= 40 \text{ pF}$.
2. If output stage collector path resistance $= 2 \text{ k}\Omega$ in circuit of exercise 1, what will be the effect on the fan-out.
3. In a DTL logic circuit, (Figure 13.5) if 5 V to the input stage diodes there is an $R = 4 \text{ k}\Omega$ and there are 2 diodes in place of three diodes feeding the next stage base current, what will be collector-emitter currents at T' at logic output of 1 and 0 when $\beta_F = h_{fe} = 20$, $V_{CE(\text{sat})} = 0.2$, $V_{BE(\text{ON})} = V_{BE(\text{sat})} = 0.7 \text{ V}$ with no next stage connection.

Also Calculate (i) the fan-out of the circuit, (ii) propagation delay assuming a next stage effective capacitance $= 40 \text{ pF}$ assuming maximum possible next stages connected, (iii) power dissipated per gate at output $= 0$ and when output $= 1$.

4. If output collector path resistance = $2 \text{ k}\Omega$ in circuit of exercise 3, what will be the effect on the fan-out.
5. (i) What will be logic outputs if the 3 circuits of RTL logic Exercise 1 circuit connect in wired logic configuration (ii) What will be logic outputs if the 3 circuits of DTL logic Exercise 2 circuit connect in wired logic configuration.
6. A triplet of diode p -ends in a multi-emitter input transistor of a TTL circuit (Figure 13.6) are used for the inputs A , B and C . What are the output stage collector currents and voltages under different conditions? What will be the current directions if the output connects to a next stage TTL? Show these in Table 13.8.

TABLE 13.8

A	B	C	Output voltage	Output current in supply circuit of the output stage	Current direction when output connects to a next stage TTL
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

7. If reference Voltage in ECL logic circuit (Figure 13.7) changes from -1.15 V to -1.25 V , what will be the changes in the following specifications? $V_{EE} = 5.2 \text{ V}$ $V_{CC} = 0 \text{ V}$, $V_{OL} = -1.7 \text{ V}$, $V_{OH} = -0.9 \text{ V}$, $V_{IL} = -1.4 \text{ V}$ and $V_{IH} = -1.2 \text{ V}$.
8. What will be output Y if another ECL circuit is (i) wired AND and (ii) wired OR?
9. Which are the data not representing the correct situation in Table 13.9?
10. Consider TTL circuit of Figure 13.6(a). (i) What is the value of active pull totem pole circuit resistance between supply and collector, if logic state 0 sink current is 1 mA . (ii) What is the value of active pull totem pole circuit resistance between supply and collector, if logic state 1 total current to next stage is maximum 0.01 mA/gate . (iii) what is turn-on 0 to 1 transition delay time? What is turn-off 1 to 0 transition delay time? What is average delay time? Assume $C = 0.05 \text{ nF}$. Assume fan-out = 20.
11. Consider the ECL logic circuit of Figure 13.7(a). If two input stage transistors are given the p.d of -1.0 V each, and reference to base of other differential pair transistor is -1.15 V , show that the input stage transistor state whether cutoff or saturation or normal region. If one of the input stage now become at -0.9 V , what will be changes in the outputs.

TABLE 13.9

S. No.	Type of gate	Propagation delay ns	Power dissipated per gate mW	Speed *Power Product pW.s = pJ (Pico Joule)	Fan out	Maximum operating frequency MHz	Noise immunity
(A)	NMOS operating at 50 MHz	300	0.2 to \approx 10	60	20	50	Small
(B)	Standard TTL	10	10	100	10	35	Very good
(C)	CMOS in 74HC series static*, 600 mW/MHz 0.045 static, \sim 10 at 1 MHz		18	2.5 mW			
(D)	ECL (10K Series)	2	50	Very Good	25	\sim 75	Small

13. Consider a circuit of Figure 13.8. (1) What will be injected current if V_{BB} is 2 V, V_{EB} is 0.7 V and R external is 20 k Ω . (2) If logic input to the base of inverting transistor amplifier is 1, what will be collector current in transistor T ?
14. Consider the circuit of Figure 13.11(a). What will be the circuit and logic function(s) possible if (i) One depletion mode n -channel MOSFET T' is replaced by two in series (ii) Two enhancement-mode n -channel MOSFETs T_X and $T_{X'}$ are replaced by two logic drivers in parallel and one another logic driver in series?
15. Consider the circuit of Figure 13.13(a). What will be the circuit and logic function(s) possible (i) Four enhancement mode p -channel MOSFETs T' and T'' forms a pull up network (ii) There are two enhancement mode n -channel MOSFETs T_X and $T_{X'}$ in series and there is another pair of the logic drivers in parallel at the pull down network?
16. A typical VLSI CMOS logic circuit dissipates 50 nW static, 0.6 nW/MHz. What will be the power dissipated at 500 MHz operation if at a given instant maximum 1000 gate circuit is operational?

■ QUESTIONS

1. Why is the DTL circuit considered as better logic circuit than RTL?
2. Compare the fan-out, turn-on delay times, turn-off delay times and power dissipated per gate in circuits of RTL, DTL, HTL and TTL?
3. What are the advantages that can be obtained by an open collector TTL?
4. An I^2L circuit needs minimum silicon area in an IC? Why?
5. Compare modified DTL circuit operation with that of DTL.

6. Draw into four circuits a TTL NAND circuit in Figure 13.6(a) that can be used to make AND, OR, XOR and NAND gates.
7. Draw the four circuits for CMOS NOR circuit in Figure 13.13(a) that can be used to make AND, OR, XOR and NAND gates.
8. What are the advantages of using CMOS over NMOS?
9. What are the advantages of using TTL in place of DTL logic?
10. A typical VLSI logic circuit dissipates 1 nW static, $0.06 \mu\text{W}/\text{MHz}$. What will be the power dissipated at 1MB RAM ($1 \text{ MB} = 1024 \times 1024 \times 8 \text{ bits}$) if each bit cell has 4 MOSFETs?

This page is intentionally left blank.

CHAPTER 14

CPLDs and FPGAs

OBJECTIVE

We will learn in this chapter about CPLDs and FPGAs. These are advanced high-density field-programmable devices.

14.1 CPLDS

The PROM, PLA and PAL (PLDs) are programmable logic devices for combinational circuits. Following are the needs observed when designing the various application circuits:

1. The output from the OR gate(s) is sometimes needed in complement form.
2. Sometimes an output is needed through a tristate or tristate NOT form also. [Figure 14.1(a)]. Tristate output(s) enables the output availability on a common bus from several sources.
3. Sometimes an output with or without complementing (or both) is needed to be feedback to an input of the same stage to a previous stage or next stage. [Refer lower part of the circuit in Figure 14.1(a)].

PALs for these applications as per need are available in the various versions. CPLDs are the complex programmable logic devices of high logic gate-densities, that has many blocks made of PLAs, PALs or GALs (Sections 14.2 and 14.3). A block is a unit which repeats in rows or columns or in a hierarchical structure.

14.2 Digital Systems: Principles and Design

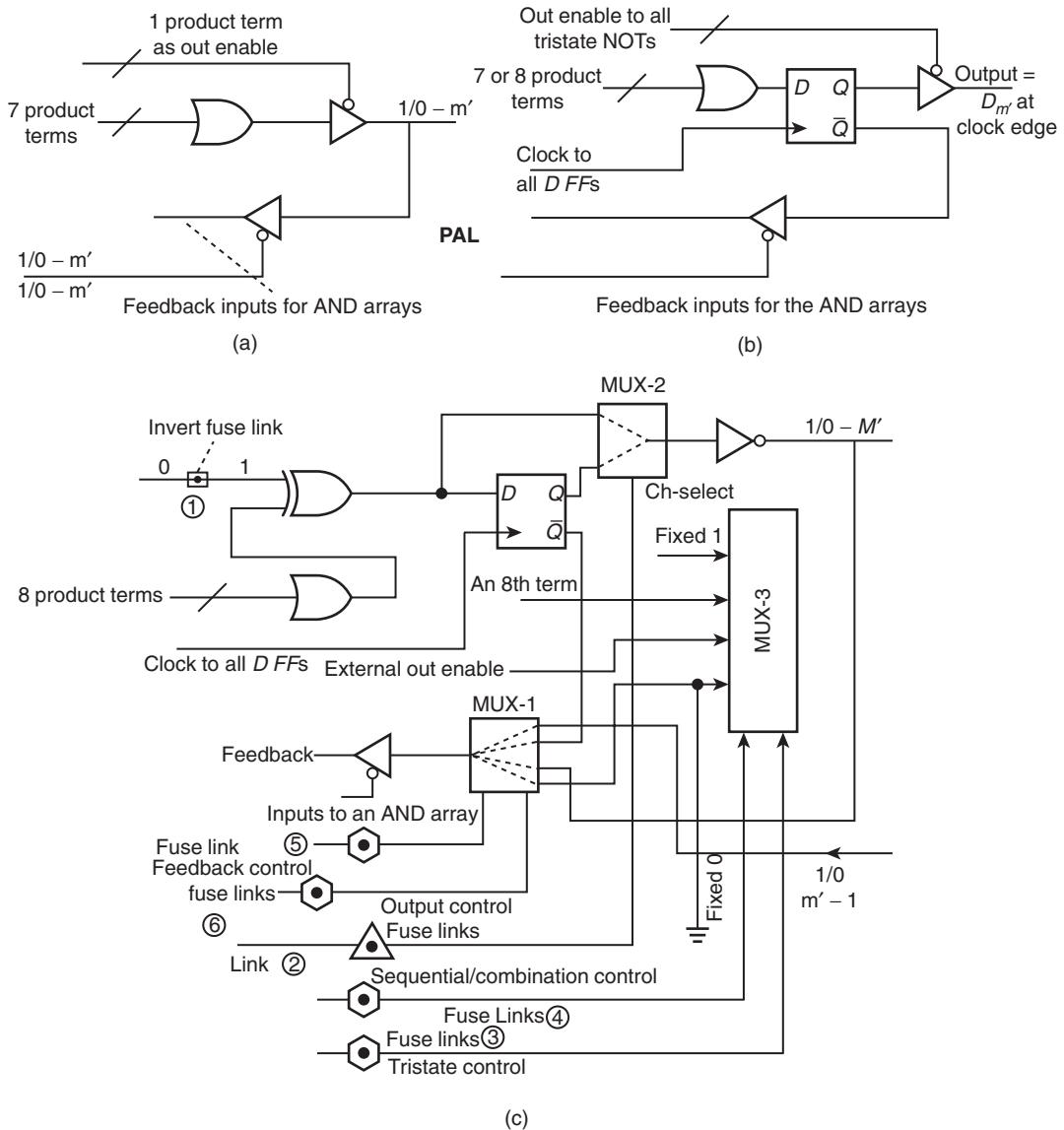


FIGURE 14.1 (a) Output stage cell of a PAL (b) Output stage cell of a registered PAL (c) Macrocell of a GAL.

14.2 REGISTERED PAL

Many a times, the output bit(s) needs to be stable like in a memory cell and is according to the inputs at the instance of a clock edge or clock pulse (master-slave) transition. For example, refer Figure 8.2(a) sequential logic circuit for a PIPO register. In that circuit an X_i registers at Q_i output. Other example is a counter, on the count pulse the output $Q_D Q_C Q_B Q_A$ changes and remains stable till next clock edge. The process of transition at a defined instance is called registering the input(s)

as per the in-between logic. The output is the registered output at the clock edge. [Registering means remembering the state even after withdrawal of the inputs.]

A form of PAL is *registered PAL*, for example, an IC 16R8. In addition to a tristate NOT at each of OR output [Figure 14.1(a) upper part] as in 16L8, there exists a output stage cell, which consists of the *D*-flip flop (*FF*) [Figure 14.1(b) upper part] also before the tristate NOT and each *D*-*FF* has an edge triggered clock input pin, then the PAL device becomes a *registered PAL*. The input(s) can now be withdrawn after the application of a clock pulse, which registers the programmed output from the given set of inputs.

Point to Remember

An unregistered PAL acts like a combinational circuit while a registered PAL also performs as a programmable general-purpose sequential circuit. (PAL has two planes—AND plane, which is programmable and fixed OR-plane.)

14.3 ARRAY LOGIC CELL

Consider the following questions:

1. Can there be a more flexible output stage than simply a tristate NOT or a *D*-*FF* plus tristate NOT like in the 16L8 and 16R8, respectively? [Figures 14.1(a) and (b).] Flexible means programmability for the output stage either as (i) active 1, or (ii) active 0, or (iii) standby state either 0 or 1 or tristate or tristate not, or (iv) the output feedback as such or with complement, or (v) the output feedback to the present stage input (iv) the output of previous stage given as a succeeding stage input as given in a ripple counter or ring counter or Johnson counter. (Figures 8.7 to 8.10)
2. Can there be a device, which can be erased like an E²PROM and so that it becomes E²PROM array that is reconfigurable again?
3. Can there be a programmable logic device (PLD), which can work both as PAL as well as a registered PAL?
4. Can there be a flexibility to convert a PLD from a combinational circuit device, like PAL 16L8 to a sequential circuit device, like PAL 16R8 (or 21R8 where maximum inputs can be 20 corresponding to 40 columns)?

Lattice (a company) gave the solution of above questions in what is now known as GAL (generic array logic). Output stage of a GAL is also called a macrocell.

Points to Remember

GAL is a PAL like device with an E²PROM like erasing ability along with the programmable output stage and feedback stage for designing a combinational or sequential circuit.

Figure 14.1(c) shows, what a macrocell is. A popular GAL version is GAL20V8. S.G. Thomson, Lattice and other leading companies manufacture the GALs. Example 14.1 will explain the use of fuse links to control output and input stages.

Due to use of the CMOS gates and flip-flops, the these PLDs consumes less power and current is ≈ 45 mA with access time for obtaining an output from a change at the inputs is < 75 ns. These PLDs are called quarter power GALs. These new CMOS based GALs have made PALs and registered PALs less practical because of its excellent low power dissipation feature. Advanced Micro Devices made a 0.01 mA standby power dissipation version of GAL. A CPLD may contain approximately 5000–12000 gates and 50 GALs/PALs.

Example 14.2 give an example of a CMOS PLD. Table 14.1 gives the features of PAL, registered PAL, GAL and CMOS-array logic. PLDs. In the table, the abbreviations are as under:

n'_i = number of internal inputs to each OR from the AND arrays.

m'_o = number of non-tristatable dedicated outputs.

m'_o = number of tristatable dedicated output.

n_i = number of external dedicated inputs to an AND array.

max. m_o = maximum number of outputs.

max. n_i = maximum number of inputs.

n_Q = number of Q outputs which are programmable as feedback inputs to an AND array.

TABLE 14.1 PLAs and array-logics

Type	n'_i, m'_o, n_i and n_Q	No. of ORs	Output stage of each OR	Max. m_o	Max. n_i	Designable circuits
16L8	7, 2, 10 and 6 from the tristate NOT	8	A tristate NOT	8	16	Combinational (C)
16R8	8, 8, 8 and 8 from \bar{Q} of each D-FF	8	Synchronous clocked. An edge trigger D-FF before a tristate NOT	8	16	Sequential (S)
20V8 (E ² PROM based)	8, - 12@ and 8*	8	Macrocell ¹	12*	20	Both C and S
85C224 (EPROM based)	8, -, 14 ¹ and 8	8	Macrocell ¹	8	22	Both C and S

* 8 of these through the macrocell. @ means a global clock and a global out-enable are in addition. ¹ means including clock and out-enable (non-global) – means not available.

Points to Remember

A PAL implements a combinational circuit of the multiple inputs, n , with multiple outputs, m , total $n + m$, say, 20. It essentially fuses the sum of products (SOPs) functions. A GAL implements a combinational as well a sequential circuit of $(n + m)$ inputs and outputs. The GAL has a macrocell at each output stage of the SOPs. The macrocell makes it possible to select one or more options as follows:

- (i) a feedback to input as addition input from a present stage or a neighbouring stage,

- (ii) a complementary output,
- (iii) a tristate output, and
- (iv) a registered output.

A GAL provides a flexible output stage for each input stage implementing a SOP function. It does not provide for multiple inputs at its macrocell and the multiple outputs from its macrocell. Total number of input and output pins are 16, 20 or 22. Further, there is only one common clock input to all the output stage flip-flops.

14.4 FIELD PROGRAMMABLE GATE ARRAYS (FPGAs)

In a field programmable gate array (FPGA), there are no distinct input and output stages as in the AND-OR array and macrocell, of a PAL and GAL, respectively. Consider an exemplary FPGA with logic cell (for example, Figure 14.2 right side cell) with all the logic cells arranged in the array form (Figure 14.2 left side for an arrangement), 12×8 or 24×32 matrix (array) are examples of the array structures. A logic cell is also called CLB (configurable logic block). A 12×8 FPGA possess total 96 logic cells. A 24×32 FPGA possess 768 logic cells.

A typical logic cell consists of the followings:

- (i) The gates to implement SOP function,
- (ii) A D -*FF* with preset and clear, and
- (iii) Data path selector multiplexer (MUX) at the input.

Xilinx has done pioneering work. There are 100 to 750 cells in a single EPLD IC for implementing FPGA circuit with features as follows:

- (i) The number of IO (input and output) pins could be 50 to 200,
- (ii) Number of clock inputs can be 8, and
- (iii) Number of gates could be from 1000 to 10,000.

[Note: A latest FPGA XC2VP125 has 125136 logic cells.]

Point to Remember

The logic cell or a CLB can be thought of a combination circuit input stage for SOPs plus a macrocell designed in a much more flexible way with (i) Feedback possibilities not only from a neighbouring IO stage but from other stages as well, and (ii) provision of multiple outputs from a macrocell.

A typical logic cell shown at Figure 14.2 right side has (i) two number six inputs AND gates, four number two input AND gates, three number 2 to 1 MUXs and a clock input edge triggered D *FF* with the preset and clear inputs, and (ii) three number outputs from the ANDs, one output from the MUX and one output from the D *FF*. Per logic cell, the total number of inputs are 21 and outputs are 5. We can not only obtain SOP functions on the inputs but also the multiplexing and decoding functions at the inputs at the logic cell.

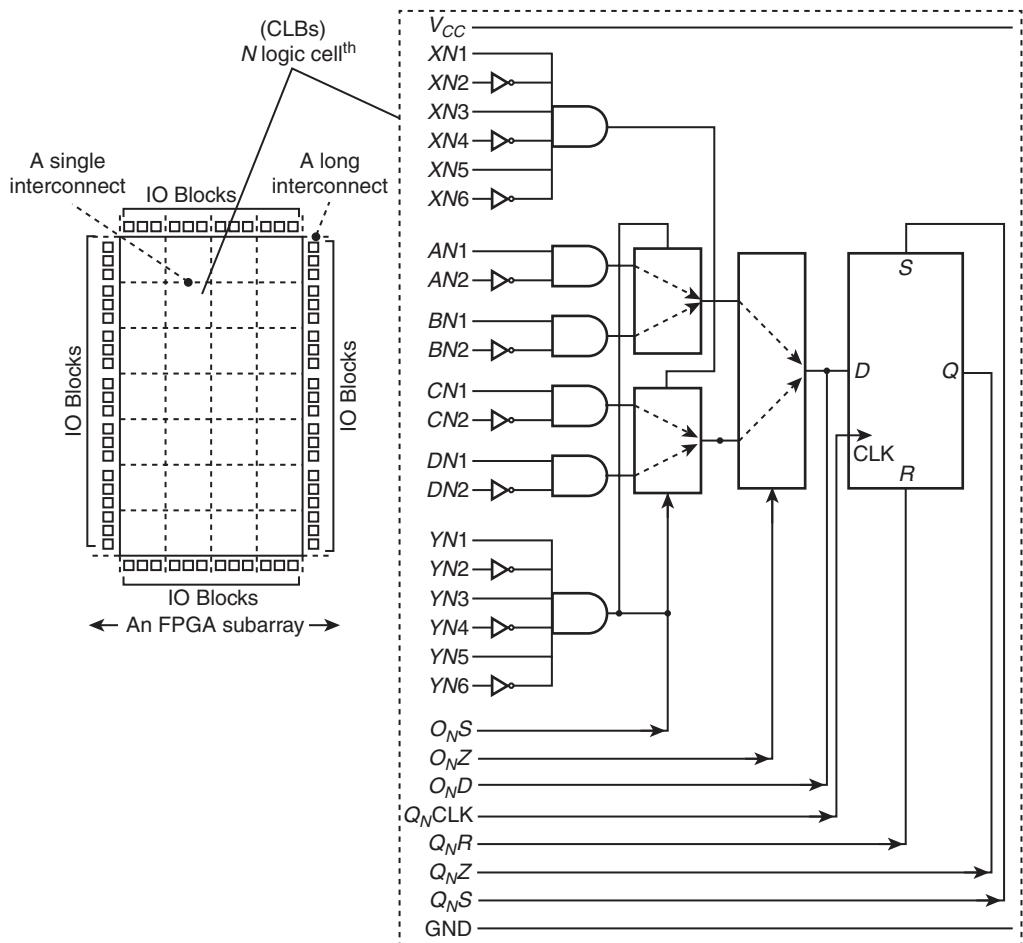


FIGURE 14.2 FPGA (Logic cell (right side) and the array structure of the logic cells (left side) (Dotted lines are single interconnects with programmable switches) (Dashed lines are long interconnects) logic cells from a CLB (Configurable logic block).

The above described logic cell design, with each cell arranged like an element in a matrix inside an FPGA, facilitates implementation of a multiple bits (16 or 32) complex combinational or sequential circuit. The examples are an adder or other arithmetic unit, a state machine, a shift register or an auto reloadable counter. The $D\text{ FF}$ in a cell of the FPGA is configurable as $JK\text{ FF}$, $RS\text{ FF}$ or $T\text{ FF}$, which facilitates an implementation of the 16 or 32 bit circuit for the various types of the shift registers and counters. Example 14.6 will give the forms of programmable links. The number of logic cells can be very large. For example, 125 to 136 in Xi link Xc 2 VP 125. It has four power PC processors also.

An FPGA finds the exemplary applications; data acquisition logic, plant automatic operation controller, graphic or image or voice processor, mouse interface, disk cache controller or parallel processor controller or encryption device,

decryption device, pattern recognizer, DNA sequence storage, signature recorder. An FPGA can have 1000–50000 logic gates (NANDs or NORs).

■ EXAMPLES

Example 14.1 Give an example of fuse-links in an array logic for a CPLD.

Solution

A macrocell is at an output stage of a GAL. A macrocell in the exemplary array logic GAL20V8 has the additional fuse/links. These are programmable like the fuse/links in AND-OR arrays or PAL or PLA. The macrocell(s) of GAL are programmable by an external programming unit, which fuses an undesired link to obtain the desired logic and/or sequential functions at the output stage(s). Example 14.6 describe interconnect link-method A programming unit does the programming of the all the fuse/links present in the macrocells at all the output stages as well as at the AND-OR arrays. The unit is like the one for an E²PROM. The macrocell in GAL20V8 (Figure 14.1(c)) has the following fuse/links, which are to be programmed as per the application circuit:

Link 1: A fuse/link for complementing an output. It controls the availability of an active ‘0’ output as in a PAL 16L8 or the availability of active ‘1’ output. The observed behaviour depends upon whether this fuse/link is linked or fused during programming by the external programming unit.

Link 2: There is another fuse/link to control the output as a sequential circuit output as in a registered PAL or as a combination circuit output as in a simple PAL, for example, in 16L8.

Links 3 and 4: There are two other fuse/links to decide one of the possibilities: (i) enable a permanent tristate at an output; (ii) no tristate at an output; (iii) the tristate NOT gate enable from an external out enable pin; (iv) tristate enabled by an 8th product term (output of the 8th AND array).

Links 5 and 6: There are two other fuse links to select one of the four possible outputs, (i) permanently ‘0’ state and its complement ‘1’ as the two feedbacks to an AND array; (ii) feedback and its complement feedback from the another output stage; (iii) feedback and its complement feedback from the Q output (registered output) of the D -FF as in 16R-8 or 20R8; (iv) feedback and its complement feedback from an output of a tristate NOT.

Example 14.2 Give a CMOS based UV-erasable PLD of fast 0.01 ns access time.

Solution

Intel UV-erasable CMOS PLD 85C224 is from 1990. Refer row 4 of Table 14.1. It has the two speed modes. One mode is a faster speed mode. It has 0.01ms access times (which is fast compared to 0.015 ms in the GALs). In fast speed mode, it consumes 35 mA when in operation at 15 MHz inputs frequency, and 25 μ A in its standby mode. In its slower speed mode, the access time is 0.02 μ s. Its also has an

automatic standby mode, which helps in obtaining a greatly reduced power dissipation when programmed like that.

One of the two speed modes is programmable by a fuse/link. This Intel PLD has another fuse/link to program the one of two possibilities: (i) I/O pin as an input for feedback and its complement feedback. (ii) Q output of $D\text{-}FF$ as input for a feedback and its complement feedback in place of the two fuse links programming for four possibilities in a GAL macrocell shown in Figure 14.1(c). This Intel 85C224 PLD has a permanent additional external OE (output enable) dedicated line. It has all the 8 product terms connected to an OR input but not connected to another tristate control multiplexer (MUX) as in Figure 14.1(c). It uses EPROM cell which is at 1 when UV light erased (nonconducting) and at 0 programming the link between interconnects (Example 14.6). In other words, MUX-3 of Figure 14.1(c) is not present. MUX-1 has two channels only in place of four, and MUX-2 is available as per Figure 14.1(c). This PLD also emulates number of PLDs – for examples, 20V8, 20L8, 20L8, 20R8, 20R6, 20R4, 20P8, 20RP8, 20RP6 and 20RP4.

Example 14.3 Why do we need additional SOP terms when programming an array logic cell?

Solution

While programming a GAL, one uses additional SOP terms for removing static hazards (Section 10.3). The static glitches occur due different propagation periods in different AND arrays. A dynamic glitch at an output can also be there. [Refer static and dynamic hazard described in Chapter 10.] This means that the output makes several transitions before a correct output is observed. The source of dynamic glitch is internal race problem, which should be eliminated by a careful planning by a user of the GAL.

Example 14.4 Give a CLB (configurable logic locu structure).

Solution

A CLB has look-up-table LUT and programmable FF -pair.

Example 14.5 What can be the possible structures of the CLBs?

Solution

The following can be possible structures of the CLBs (Figure 14.2).

- (i) Matrix with single length interconnects and long interconnect.
- (ii) Row-based Architecture with single length interconnects shown in Figure 14.2.
- (iii) Hierarchical structures.

Example 14.6 What are possible ways of implementing programming links at the single length interconnects and long interconnects in the arrays?

Solution

Figure 14.3 shows the programmable switch designs using an SRAM cell or an antifuse structure. The SRAM provides volatile links. The SRAM may

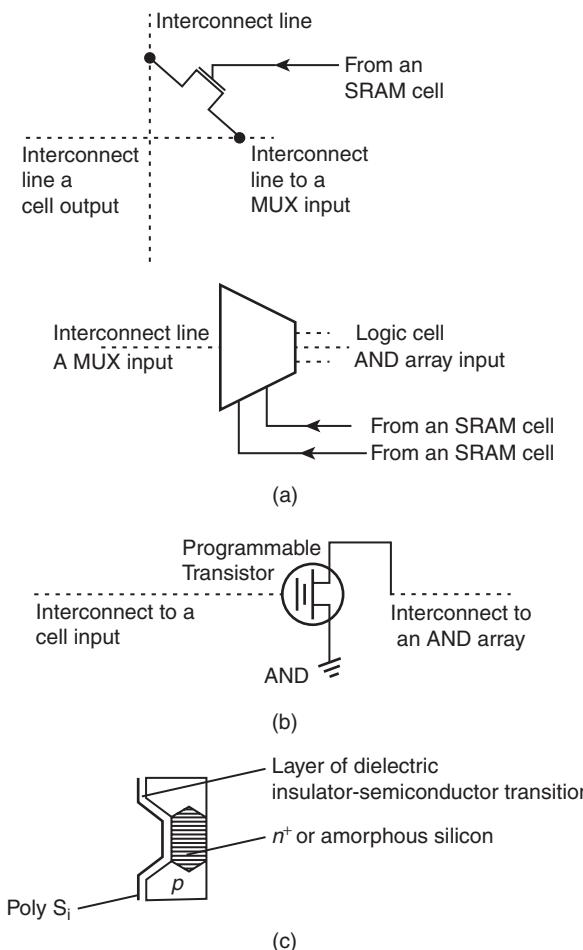


FIGURE 14.3 (a) SRAM cells to program the switching between two ends of an interconnect-line (b) E²PROM or EPROM cell (a transistor) to program the interconnect switch (on or off) interconnecting the input with AND array (c) An antifuse structure as an interconnect switch (An Actel Inc. technology) by insulator to semi conducting transition at a layer during programming.

connect a PROM (EPROM or E²PROM) in the FPGA chip to make it non-volatile.

Example 14.7 How do we use LUTs in the CLB or logic cell designs?

Solution

An LUT (look-up table) means there are m inputs (for a key or code or address) and for each input there is one output. Suppose a look up table has eight rows. In each row, it can have output 1 or 0.

An LUT in an FPGA logic cell is $2^m \times 1$ bit memory unit $m = 3$ for m -input LUT. The LUT output is given to a MUX or MUXs at a cell (CLB) using the

14.10 Digital Systems: Principles and Design

Inputs	Output
000	0 or 1
001	0 or 1
010	0 or 1
.	.
.	.
.	.
111	0 or 1

interconnect switch(es). An LUT output can also be an input to another LUT. An LUT may get input(s) from a selector bit(s) or one (or more) LUT output(s). The interconnect switches are programmed to connect the cell inputs through the LUTs or directly to the MUXs and other gates of the cell.

Example 14.8 How do the IO blocks connect in the FPGA using the interconnects the CLBs and IO blocks?

Solution

The IO blocks (Figure 14.2) connect the other IO blocks by long interconnects and connects the CLBs through single interconnects and also by long interconnects.

Example 14.9 Can we have on a single chip the CLBs as well as processors?

Solution

Yes, for example Micro Blaze for Xilinx has a 32-bit RISC soft processor core, memory and logic CLBs and peripherals on a single chip. XC2 VP125 has over 125 thousand logic cells plus four processors.

Example 14.10 List the characteristics that describe a CPLD or FPGA?

Solution

CPLD

- (i) Constitution of the CPLB. Whether using PLA or PAL or registered PAL or array logic block.
- (ii) Number of logic cells or macrocells used
- (iii) Technology for programming the interconnects between inputs and output stages and between the CLBS.
- (iv) Long or single or both interconnects programmable.
- (v) Structure in the array row, matrix or heirarchical structure.
- (vi) Delay time of interconnects formed by
- (vii) Clock frequency limit-fused links.

FPGA

- (i) Logic design of CLBs
- (ii) Use of LUTs

- (iii) Use of single interconnects
- (iv) Use of long interconnects
- (v) Use of SRAM interconnects.
- (vi) Use of EPROM or E²PROM cells with the SRAM interconnects.
- (vii) Number of logic cells
- (viii) Number of IO blocks horizontal and vertical
- (ix) Delay between interconnects
- (x) Clock frequency limit

Example 14.11

A Xilinx XC 4000 CLB has 2 D-FFs each with the R and S inputs (preset and set) and an enable input. How many CLBs shall be needed to design a (i) 32-bit PIPO and (ii) 32-bit ripple counter.

Solution

Sixteen CLBs the 32-bit-PIPO and counter will be needed.

■ EXERCISES

1. Draw Xilinx XC 4000 logic block (CLB design).
2. Draw flash logic device from Altera which uses a 24 V 10 PAL.
3. Describe Xilinx XC 7000 and XC 9500 CPLDs.
4. Describe XC 4000 FPGA. CLB. Compare the XC 4000 and XC 801.00 features.
5. List the links to be programmed for a ripple counter of 4-bit using CLBs of XC 4000.
6. Show a design each of 32-bit ripple counter, 32-bit synchronous counter, 32-bit \times 32 bit multiplier and a 8 \times 8 bit multiply and then all 16-bit unit (called MAC unit). Use XC 4000 FFGA CLBs.

■ QUESTIONS

1. Define PAL? Define registered PAL? Define GAL?
2. Define a Macrocell? How will you employ a GAL to implement a complicated combination circuit as well as a complicated sequential circuit?
3. Define an FPGA? What do you mean by a logic cell in a FPGA?
4. When do we use a FPGA, when a GAL, when a registered PAL, when a PAL, when an EPROM, and when an individual logic gates and FFs based logic design? Explain thoroughly.
5. Now FPGA have over 100000 logic cells and dissipates very low power. GALs and FPGAs have made many digital ICs a thing of past. How and why is it so?
6. Search the Web and find Xilinx Spartan 3 and XC2VP125 features.

This page is intentionally left blank.

CHAPTER 15

VHDL—RTL Design, Combinational Logic, Data Types, and Operators

OBJECTIVE

We learnt the design of simple combinational and sequential circuits in earlier Chapters. The design of large circuits needs a language to describe the circuits and components. The language facilitates automation of design process. A very-high-speed integrated circuits (VHSIC) Hardware Description Language (VHDL) for description, for design synthesis, and for automation of system design. In this chapter, we shall learn the concept of VHDL and RTL model. We will learn by the examples of combinational logic circuits. The data types and operators which are used in VHDL will also be studied.

15.1 VHDL

A *language* is a system of communication that consists of a set of sounds and written symbols, which are used by the people of a particular country or region for talking and writing. (Collins COBUILD English Language Dictionary)

A *computer language* is a system of symbols, data, words, and operations which are used by a programmer for writing and execution of programs on a computer.

A *description language* means a language for description using a system of symbols, data types, operators, data, words, and schematics. Description may be of a house, an electronic circuit, or a very large scale integrated circuit (VLSI).

A description language is required to describe hardware of an integrated circuit or *very-high-speed integrated circuits* (VHSIC). *VHDL* is the VHSIC Hardware

15.2 Digital Systems: Principles and Design

Description Language for VHSIC hardware description using a system of symbols, data types, operators, data, words, and schematics. VHDL and Verilog are two widely used languages.

VHDL is used to describe:

1. Hardware of very-high speed integrated circuits
2. Hardware of electronic digital systems
3. Hardware of the electronic digital as well as mixed analog-digital systems
4. Hardware of FPGA (field-programmable gate array)
5. Hardware description of microwave circuit design using specific VHDL extensions

VHDL enables:

1. Modeling of the entities in the system by system description
2. Modeling of the architecture of the system by system description
3. Modeling of the behaviour of the system by system description
4. Modeling of the structure of the system by system description [structure of an entity is described by set of interconnected components, circuits or sections of the circuits. For example, Adder may have two bits plus carry adder (full adder) as the component. A component may have adder plus other components.]

Synthesis means finding the gate-level interconnections for a circuit or component or system. VHDL describes design synthesizes in three steps: system description (modeling) as first step, verification (simulation) using testbench as second step, and then synthesis by using tools as third step.

The synthesis of the design in terms of real circuit consisting of the gates and wires after the modeling and verification, and the synthesis enables automation of the hardware design. Figure 15.1 shows three steps for synthesis of a gate-level design and then the program to make the VHSIC.

15.1.1 VHDL Standard IEEE 1076

There are several versions of IEEE 1076. Table 15.1 gives versions of VHDL.

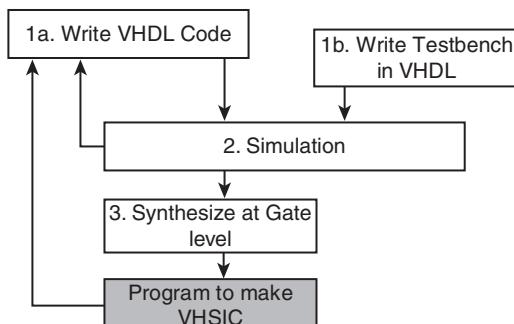


FIGURE 15.1 Two inputs and one output considered as four registers of 1 bit each.

TABLE 15.1 IEEE standard 1076 versions

Year	Version/revision	Important change
1983	Initial version	
1987	Initial version revision	Addition of XNOR operation and new data types ¹
1993	Revision	
2000	Minor revision	Addition of Protected Type
2002	Minor revision	Relaxation of Buffer port mapping rules
2008	Major revision	Introduction of external signals, Revision based on VHDL 4.0 draft (improved version of VHDL 3.0 draft)

¹ String (array of characters), bit vector (array of bits), and also the range of logical, numerical, time, and character data types.

The VHDL IEEE 1076 has extensions as follows:

1. IEEE 1076.1 (VHDL-AMS is extension for Analog and Mixed-Signal (digital + analog) circuit design.
2. IEEE 1076.2 is extension to provide the improved working along with handling of real and complex data types.
3. IEEE 1076.3 provides signed and unsigned types for arithmetic operations.
4. IEEE 1076.2 and 1076.3 are based on 2006 VHDL 3.0 draft. It incorporates VHPI (interface to C/C++ languages). It enables use of a subset of PSL (Property Specification Language) and extended set of operators. It provides for a flexible syntax of case and flexible generated statements.

15.1.2 VHDL Standard IEEE 1164

VHDL IEEE 1164 is a standard based on VHDL 3.0 draft of 2006. VHDL IEEE 1164 is based on the following:

1. The 9-valued logic in place of 2 (logic states 0 and 1) or 3 valued logic states (0, 1, and Z, where Z is tri-state) [Tri-state means a high impedance state, when there is negligible driving current (negligible also called zero drive current) from the input or output in the circuit. 0 and 1 are the strong drive current states.]
2. Scalar std_ulogic
3. Vector std_ulogic_vector
4. Package design unit

Logic means resolved logic. Ulogic means unresolved logic. The std_ulogic has unresolved states. The type std_ulogic is declared in package std_logic_1164. It has multiple drivers at a common input. A resolution function resolves the logic state in case of multiple drivers.

Consider a signal at input X from two wired connections. Example of a resolved state is when one of the input driving signals is in tri-state and other input driving signal is not in tri-state. When both the inputs are not in tri-state, then logic state at X depends on resolution function. When resolution function does not lead to a result then state at X is unresolved.

15.1.3 VHDL Libraries

Library modules are imported during the design. Following example shows how.

Example

Assume that 1164 standard is used. What will be the declarations in VHDL codes for using the library codes?

Solution

```
library IEEE;
use IEEE.std_logic_1164.all;
```

15.1.4 VHDL Identifiers, Keywords, and Comments

Each language has identifiers and rules for using a sequence of characters in the identifiers. Consider English language identifiers for the names of the persons. A name Raj Kamal is an identifier. Raj Kamal identifies the author of this book by a sequence of 9 characters. The names 1RajKamal, Raj#Kamal, and Raj×Kamal are not permitted as the name.

VHDL has:

1. Basic identifiers [a sequence of characters among *a*..., *z*, *A*, ..., *Z*, 0, ..., 9 and _ with the condition that:
 - (i) First character cannot be a digit or an underscore (_).
 - (ii) Last character cannot be an underscore (_).
 - (iii) An underscore (_) is permitted in between a sequence of characters once only.
 - (iv) Lower case and upper case characters are considered the same.

The basic identifiers have case insensitivity.

2. Extended identifiers (extension by using backslashes) [a sequence of any allowed characters among *a*..., *z*, *A*, ..., *Z*, 0, ..., 9, #, \$, ...@ and _ with the condition that:

- (i) First character can also be digit or underscore,
- (ii) Lower and upper case characters are considered different.

The extended identifiers between the backslashes have case sensitivity.

Certain identifiers are called keywords and cannot be used except as per VHDL language rules. Port is a keyword. Coding cannot be used as follows:

Process1 process. STD_LOGIC *A*, *B*: in; out *Y* STD_LOGIC.

[Underlined words are the keywords used improperly here.]

Comments are prefixed by a sequence of two hyphens (--). Specific text or description corresponds to a comment or comments. The comments are ignored when the codes get executed. They are very important in the codes because they enable in understanding them easily by the programmer or reader later on.

Example

1. Give examples of permitted basic identifiers.
2. Give examples of not permitted basic identifiers.

3. Give examples of identical basic identifiers.
4. Give examples of extended identifiers.
5. Give examples of distinct extended identifiers.
6. Give examples of keywords.
7. Give examples of distinct extended identifier and keyword.
8. Give examples of how to describe comments in the codes.

Solution

1. Examples of permitted basic identifiers in VHDL are `Z`, `A`, `Y1`, `muxSignal`, `mux2logic`, `LogicCircuit_C`.
2. Examples of not permitted identifiers in VHDL are `_Z`, `1Y`, `mux3_`, `Logic_Circuit_C`.
3. `muxsignal`, `MUXSIGNAL`, `muxSignAl`, and `MuxsignaL`, are considered as identical identifiers. `LogicCircuit_c`, `logicCircuit_c`, `LOGicCircuit_c`, and `logIccircuit_C` are considered as identical identifiers in the VHDL codes.
4. `\2716ROM\`, `\logiccircuitC\`, `_Counter\`, `\logiccircuitc\`, `_8086\`
5. `\logiccircuitc\`, and `\logiccircuitC\` are distinct identifier and are not identical.
6. If, when, constant, variable, process, signal, port, architecture, and entity are the examples of keywords.
7. `process` and `\process\` are distinct identifiers. The `process` is a keyword, but `\process\` is not a keyword.
8. (i) `A`, `B`: in `STD_LOGIC`; -- Inputs are *A* and *B* and have standard logic values 0 or 1
(ii) architecture behaviour for `counter_C` is -- Synchronous counter signals and process

15.1.5 VHDL Data Objects

Each data object can hold a value of specific type or class.

Four classes (**Constant**, **Variable**, **Signal** and **File**) of data objects in VHDL are as follows:

1. **Constant** is used for declaring a constant value or values which do not change during simulation of program for a system.
2. **Variable** is used for declaring a value or values which may change during simulation of program for a system.
3. **Signal** is considered as a wire in a circuit or modeled as flip-flop that gives output to an input in the RTL model.
4. **File** is used to save a sequence of values that can be read by a `read` procedure, written by `write` procedure, and appended with additional values. VHDL file can save text (a sequence of strings), bits in `bit_file`, or `std_logic` values in bits in the `STD_LOGIC_file`.

Example

1. Give examples of constants. How are they declared?
2. Give examples of variables. How are they declared?
3. Give examples of signals. How are they declared?
4. Give an example of a file. How it is declared?

Solution

1. Constant gate delay: Time = 20 ns; Constant data bus width, address bus width: Integer = 16;
2. Variable A, B: in STD_LOGIC; Y: out STD_LOGIC; Variable CY, Is_Correct: Boolean;
3. Signal clock_input: Bit; Signal ADDRESSBUS : Bit_Vector (0–15);
4. file counter_architecture: Text open Write_Mode;

15.2 RTL DESIGN

RTL means Register Transfer Level model. RTL design can be in the hardware description languages (HDL) such as VHDL and Verilog.

RTL is a high-level language. It generates gate-level design automatically from an RTL model of a circuit. The meaning of the terms register, register transfer, and RTL in RTL are as follows:

1. *Register*: RTL model considers register as a 1-bit storage unit. Register can be considered as one *D* flip-flop (*FF*) or a transparent latch. An input or output in a circuit is considered as state at a register of 1-bit.
 - (i) Suppose an input *A* logic state is 1 then it is assumed that a register is having input = 1 though an interconnection (data path) and 1 is saved in the register *A*.
 - (ii) Suppose an output *Y* logic state is 0 then it is assumed that the register *Y* of 1-bit saves 0 in it and is thus having output state 1 through another interconnection.
 - (iii) Suppose an input *B* logic state is 0 then it is assumed that a register *B* is having input = 0 which is saved in it.
 - (iv) Suppose output *Y* logic state is 1 then it is assumed that the register *Y* of 1 bit stores 1 and is thus having output state 1 to the interconnection.

Figure 15.2 shows two inputs and one output considered in three registers of 1-bit each.

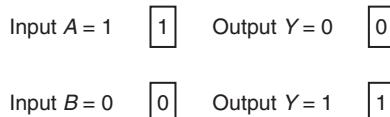


FIGURE 15.2 Two inputs and one output considered as four registers of 1 bit each.

2. *Register transfer:* Consider register transfer by a logic circuit C . The transfer of logic is from A and B to Y after a logic operation as per LogicCircuit_C.

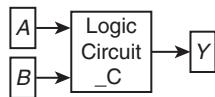


FIGURE 15.3 Two registers A and B transferring logic state to register Y after the logic operation at a circuit. Arrow shows an interconnection (signal)

LogicCircuit_C is considered as an entity, which transfers the logic states of two registers to another register at the Y . The C is a circuit to be designed using logic gates.

3. *Register Transfer Language (RTL):* It models the logic circuit C as follows: It describes that A and B registers transfer the state such that $Y = 1$ when both A and B has logic state 1, the logic operation is as that of an ‘AND’ and ‘AND’ is a transfer function for the LogicCircuit_C.

15.2.1 Data Flow Model

A data flow model uses concurrent assignment(s) using the signal(s) for generating another signal or set of signals. This model uses the concurrent signal assignment from input to output. Concurrency is necessary for modeling the digital circuit which are having concurrent in nature.

Example

Show assignment(s) in data flow model of LogicCircuit_C in RTL. Show the assignment for the signals from the A and B for generating a signal to the Y .

Solution

$Y \leftarrow A \text{ and } B \text{ after } 20 \text{ ns}; \text{-- } Y \text{ gets signal after } 20 \text{ ns from the signals reaching } A \text{ and } B$

Point to Remember

Data flow model gives new assignment to a signal after the operations on the signals. A circuit or component is considered as a system for flow of data along the data paths (interconnections).

15.2.2 Port

Input port means a register of one or more bits which gets input though an interconnection(s). Output port means a register of one or more bits which gives an output though an interconnection(s).

VHDL models ports as signals (input or output) through which an entity or circuit, or component or system communicates with other which gets input(s) from a register(s) through interconnection(s).

RTL design of VHDL considers port as one which gets signals (inputs) from the registers (each of 1-bit register) through interconnection (s) and which gives signals

(outputs) to a register(s) through interconnection(s). Port can be considered as having one or more D flip-flops (FF).

Figure 15.4 shows two input ports A and B and an output port Y . Each input port receives a signal through interconnections and output port sends a signal.

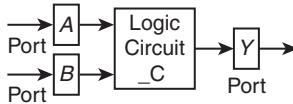


FIGURE 15.4 Two input ports A and B and an output port Y . Each input port receives a signal through interconnections and output port sends a signal to the interconnection.

15.2.3 Finite State Machine (FSM)

The register transfer functions describe the data flow through interconnections (data paths). The flow is between two set of registers in RTL design. A register saves a finite logic state 1 or 0. A set of logic states generate another set of logic states sequentially. The entity behaves as an FSM. It has interconnections (data paths) between registers.

Figure 15.5 shows an FSM model LogicCircuit_M generating states using a state transfer function. Two states A and B using state transfer function are generating state Y .

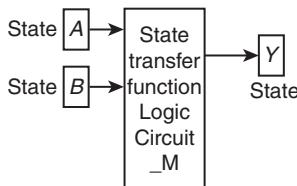


FIGURE 15.5 State transfer function LogicCircuit_M of a finite state machine producing one state Y from two states A and B at the input ports A and B and output port Y .

15.2.4 Entity in RTL Model

15.2.4.1 Entity

A design consists of the entities. An entity is defined first and it can be reused. That is, a library of entities can be defined, which can be used in another design. A VHDL declaration `entity` is used as described in Example 15.6.

Example

Declare entity LogicCircuit_C in Figure 15.3 using RTL. A and B are inputs to port and Y is output to the port. Inputs are standard logic states 1 or 0 and output is standard logic state.

```

library IEEE; -- Include IEEE library
entity LogicCircuit_C is
    port (A,B: in STD_LOGIC; Y : out STD_LOGIC);
end LogicCircuit_C;-- end of assignments for the entity LogicCircuit_C
    
```

VHDL entity means a section of circuit or a circuit or a component or a system. [A system consists of one or more components. A component consists of one or more circuits. A circuit may consist of several interconnected sections.]

A VHDL entity (design) has one or more input and one or more output or ports for the input(s) and output(s).

The ports are connected (wired) to the systems. An entity may consist of interconnected entities, processes and components.

All interconnections in the entity operate concurrently (i.e., at the same time—does not mean simultaneously).

Entity description in VHDL can be considered as consisting of entities, which are external or internal and visible or hidden. Each of them associates with a behaviour (described by the architecture and process) independently.

15.2.4.2 Begin and End

A set of statements in a process or architecture or a set of concurrent assignments is between the begin and end .

Point to Remember

Entities are circuits, components, and systems. VHDL entity (design) has one or more input and one or more output or ports for the input(s) and output(s).

15.3 BEHAVIOUR MODEL FOR PROCESS IN AN RTL DESIGN

The explanation of behaviour model and the process in an RTL design are as follows:

1. *Behaviour model*: It uses assignments for defining the functionality of an entity. It uses a set of sequential assignments in a process, which are executed sequentially. It generates a signal or a set of signals.
2. *Process*: It means an action or sequence of actions or concurrent actions which use the inputs and generate an output(s). For example, sugar-cane is input and the sequence of actions and concurrent actions in a process produce sugar. In other words, *a process is a declaration in VHDL which is defined by specifying a set of concurrent or sequential assignments*.

Figure 15.6 shows a process generating states using a register transfer function. Signals from two register *A* and *B* generate signal *Y*.

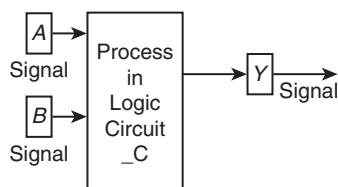


FIGURE 15.6 Two signals from *A* and *B* processed in *LogicCircuit_C* and generate signal from *Y* using concurrent assignments for the process.

Point to Remember

Behaviour model describes the functionality of an entity (circuit or component or system).

15.3.1 RTL Model Architecture

Each entity has a particular architecture. Architecture consists of constructs. A construct may have arithmetic statements, signal assignments, and statements to instantiate component.

Architecture is declared by using a set of concurrent assignments. Architecture may also use a set of sequential assignment statements. Architecture can also define a set of interconnected components.

Architecture may define a process or set of processes in behaviour model. Following example shows **architecture** declaration and **architecture body**. It uses a set of concurrent assignments for the process which is taking place at LogicCircuit_C.

Example

Declare architecture of entity LogicCircuit_C in Figure 15.3 using RTL in behaviour model.

```
architecture RTL for LogicCircuit_C is
begin
    process (A, B) -- Define the assignments with A and B as inputs to
    generate Y
        begin
            if (A = '1') and (B = '1') then
                Y <= '1'; else Y <= '0';
            end if;
        end process;
    end RTL;
```

Entity description in VHDL can be considered as consisting of entities which are external or internal and visible or hidden. Each of them associates with architecture and process independently.

Therefore, when the external interface to an entity is described, the interface description can be used by other circuit or component entities during the design. This concept of internal and external views is central to a VHDL view of system design.

An entity can be defined, relative to other entities, by its connections and behaviour. This enables alternate architecture processes independently without describing remaining codes once again.

Library or libraries can be developed for many designs or for a family of designs. Each library has a pair of entity and architecture declarations.

Points to Remember

1. Architecture is declared by using a set of concurrent assignments. It may also use a set of sequential assignment statements. It can also define a set of interconnected components.
2. Architecture may define a process or set of processes in behaviour model.
3. Description in VHDL can be considered as consisting of entities which are external or internal and visible or hidden. Each of them associates with architecture and process independently.

15.4 RTL DESIGN FOR COMBINATION LOGIC

A combinational circuit has following functionalities:

1. Logic state, at any of the outputs in the combinational circuits, depends only on the inputs at any given instant (not considering always present propagation delay period) and is not correlated at all with any of its previous output or outputs.
2. A truth table of a combinational circuit gives the values of all the m outputs for each possible combination of the n inputs and has 2^n rows and $(n + m)$ columns.
3. A combination circuit can be designed for obtaining the m outputs using the m Boolean expressions (switching functions).

Example

Consider a combinational logic for a two input multiplexer (MUX). It has an address select input D . Address select input is also called channel select input. What is the truth table for the multiplexer?

Solution

The output Y can be represented by truth table as shown in Table 15.2.

TABLE 15.2 Truth table of a MUX $Y = A$ if $D = 0$ else $Y = B$ if $D = 1$

Input A	Input B	Address select D	Output Y
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

15.12 Digital Systems: Principles and Design

The Boolean expression can be written as sum of the products and can be simplified.

$$\begin{aligned}Y &= A \cdot \bar{B} \cdot \bar{D} + A \cdot B \cdot \bar{D} + \bar{A} \cdot B \cdot D + A \cdot B \cdot D \\&= A \cdot \bar{D} + B \cdot D\end{aligned}$$

Complex combination circuit cannot be easily synthesized using truth table or Boolean expressions or Karnaugh map simplifications. Therefore the RTL synthesis can be used as a method for synthesis to get the gate-level circuit.

Example

How will you declare entity LogicCircuit_M for two signals from A and B and generate signal Y such that it either A or B as per the address or channel select input D in the multiplexer (Mux).

Solution

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity LogicCircuit_M is
    port (A, B, D: in STD_LOGIC; Y : out STD_LOGIC);
end;
```

Figure 15.7 shows a combination circuit for a two input MUX with one address select input D and a transfer function generating the output Y .

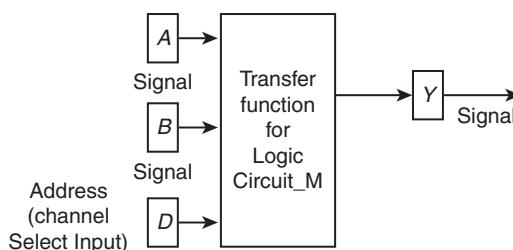


FIGURE 15.7 Two signals from A and B to LogicCircuit_M and generation of signal from Y using concurrent assignments in the transfer function for the multiplexer (Mux).

Example

Show assignment(s) in data flow model in RTL of a combinational circuit for a two input MUX LogicCircuit_M. Show the assignment for the signals from the A and B for generating a signal to the Y .

Solution

```
Y <= (A and (not D)) or (B and D) after 40 ns; -- Y gets signal after  
20 ns from the signals reaching A and B
```

Example

Show dataflow model: concurrent signal assignments for LogicCircuit_M.

Solution

```
D1 <= not D;  
Y1 <= not ((A and D1) or (B and D));  
Y <= not Y1;
```

Figure 15.8 shows a process generating states using a register transfer function. Signals from three registers A , B , and C generate signal Y .

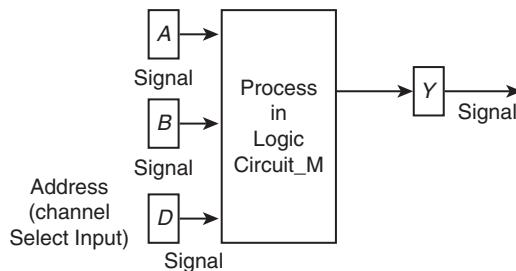


FIGURE 15.8 Three signals from A , B , and C processed in LogicCircuit_M and generate signal from Y using concurrent assignments for the process.

Example

Following is an example how to declare architecture of multiplexer shown in Figure 15.8.

How will you make declaration for architecture of entity LogicCircuit_M in Figure 15.8 using RTL in behaviour model?

Solution

```

architecture RTL for LogicCircuit_M is
begin
    signal A, B, D : std_logic;
    signal Y: std_logic;
    process (A, B, D) -- Define the assignments with A and B as inputs to
                      generate Y
        begin
            if D = '1' then
                Y <= B;
            else
                Y <= A;
            end if;
        end process;
end RTL;

```

Points to remember

1. A library declaration is done before that of the entity.
2. library ieee; use ieee.std_logic_1164.all;
3. A design description must have entity. A combinational logic circuit is an entity declaration. Ports must be declared in the entity.
4. Each entity has at least one architecture declaration.
5. Signals are declared before the process declaration in the architecture.

15.5 DATA TYPES

Consider a data object.

1. Each data object can have a value which is from a set of values.
2. Each data type defines a set of values.
3. Each data type also defines a set of permitted operations.

A type is declared for a data object using type declaration. For example, consider declaration of type for a valid tele_digit permitted for dialing.

`type tele_digit is ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9');` -- An enumerated data type which can discrete values among ten values.

There are four types of data as given in Table 15.3.

TABLE 15.3 Four types of data

Data type	Characteristics	Example
Scalar	Sequential order values	integer values, floating point values, physical values, enumerated values
Composite	Elements of single type together	Array
Access	Elements of different type together	Record
File	Enables access to the object	Pointer
	Enables access to the number of objects	A sequence of values of given type in the objects

There are four scalar types of data as given in Table 15.4.

TABLE 15.4 Four scalar types of data

Data type	Characteristics	Example
Integer	Numeric, discrete	-128; -32768; 65535;
Floating Point	Numeric	3.141; 1.0E-6; 5.25, -12.0; type supplyVoltage is range 1 to 2000 units mV;
Physical	Numeric (integer or floating point)	V = 1000 milliV; kV = 1000 V; end units
Enumeration	Discrete	<code>type tele_digit is ('+', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9');</code>

Special data types are predefined. Following are seven examples of four predefined enumerated types of data as given in Table 15.5.

TABLE 15.5 Seven examples of predefined enumerated types of data

Data type	Characteristics	Values
std_ulogic	Discrete 9-values in a sequential order from 'U' to '_'	('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '_');
Character	Discrete 127 values	a, ..., z, A, ..., Z, '0', ..., '9', '?', ' ', ',', '.', '#', '\$'....
Boolean	Discrete two values	True, False
Bit	Discrete two values	'0', '1'
File_open_kind	Three discrete values	Write_Mode, Read_Mode, Append_Mode
File_open_status	Four discrete values	open_ok, status_error, mode_error, name_error
Severity_level	Four discrete values	error, failure, warning, note

Data type *std_ulogic* (standard unresolved logic) designer requires tools for simulation and debugging. The 9-valued logic provides a very powerful simulation and debugging tool. 2-valued (1 and 0) logic in VHDL has limitations.

It consists of nine character literals in sequence ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '_') as given in Table 15.6.

TABLE 15.6 Values of type std_ulogic

Characters	Values
'U'	Uninitialized (describes default value for all objects during design simulation),
'X'	Unknown logic value either 0 or 1 but strong drive
'0'	Logic 0 strong drive,
'1'	Logic 1 strong drive
'Z'	High impedance
'W'	Unknown logic value either 0 or 1 but weak drive
'L'	Logic 0 but weak drive, wired AND or wired OR connections allowed
H	Logic 1 but weak drive,
'_'	Don't care

Example

1. Give example of enumerated data type for logic states in a decoder circuit 0 or 1 or Z (tri-state).
2. Give example of physical data type—declare type for voltage of a supply source.
3. Give example of data type declaration of signals *A*, *B*, and *D* in a multiplexer. *D* is channel select input.
4. Give example of predefined enumerated std_ulogic in std_logic_1164 and subtype declare using std_ulogic type.

Solution

1. type logicStateDecoder is ('0', '1', 'Z');
2. (i) type supplyVoltage is range 1 to 2000
units
mV;
 $V = 1000$ milliV;
 $kV = 1000$ V;
end units
(ii) type CMOS_voltage is range -12.0 to 12.0;
3. signal A, B, D : std_logic; -- The signals A, B and D are of the std_logic type of data.
4. (i) type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '_');
(ii) subtype xorlogic is std_ulogic range '0' to '1';

Points to Remember

1. Each data object can have a value which is from a set of values.
2. Each data type defines a set of values.
3. Each data type also defines a set of permitted operations.

15.5.1 Subtypes

A subtype of values can be declared of a data type. For example, we can declare characters for phone dialing as follows:

```
subtype tele_digit is Integer range 0 to 9;
subtype logicStateNAND is std_ulogic range '0' to '1';
subtype Midtele_digit is tele_digit range '4' to '6';
```

15.5.2 Array

Array is a composite data type. It consists of a sequentially placed number of values, which can be accessed by an index number. Index number means sequential position in the array. Example is as follows:

```
type Byte is array (0 to 7) of bit;
```

Special array data types are predefined. The three examples of array types of data is given in Table 15.7.

Example

1. Give examples of arrays for instruction words in ROM and data in RAM.
2. Give examples of data objects declared as array type.

Solution

1. type Byte is array (0 to 7) of bit;
type Instruction_word is array (0 to 3) of Byte; -- declare instruction word data type of 4 bytes each
type ROM_Address is array (0 to 16383) of Instruction_word;
-- declare 16384 instruction words at ROM addresses

TABLE 15.7 Three examples of predefined array data types

Array data types	Characteristics	Values
String	Array consisting of number of sequentially arranged characters.	String having characters a, ..., z, A, ..., Z, '0', ...'9', '?', ' ', '.', '#', '\$'.... Examples: "Error Found";"Messages Pending 48"
Bit_Vector	Array consisting of number of sequentially arranged bit values.	signal LogicCircuit_M: Bit_Vector(0 to 4);
std_ulogic_vector	Unconstrained array type	type std_ulogic_vector is array (Natural range <>) of std_ulogic

```

type RAM_Data is array (0 to 3) of Byte; -- declare RAM data type of
4 bytes each
type RAM_Address is array (0 to 65535) of RAM_Data; -- declare
655236 address for data words at RAM addresses
subtype stack_address is array (49152 to 65535) of RAM_
Address; -- declare subtype of data stack address value between 49152
to 65535 RAM_address

```

2. A data object can be declared using array type as follows:

```

signal A_0-A_15: RAM_Address;
signal D_0-D_31: RAM_DATA;

```

15.5.3 Type Checking

Suppose a signal A_0-A_{15} is defined as 0–15, an error results when it is assigned value 16. Whenever there is improper use of data types the error should be reported. Type checking enables errors detection when error results from data type incorrects assignment in the descriptions. A mismatch in inputs or outputs is detected during compilation. A design is generated only after type checking.

A component that expects a 4-bit-wide signal type cannot be connected to a 3- or 5-bit-wide signal; this mismatch causes an error when the HDL description is compiled.

15.6 OPERATORS

Consider a data flow assignment.

$Y \leq A \text{ and } B \text{ after } 20 \text{ ns};$ -- Y gets signal after 20 ns from the signals reaching A and B

The **and** is a logical operator that is executing the logic operation between A and B . There are six types of predefined operators in order of precedence as given in Table 15.8.

15.18 Digital Systems: Principles and Design

TABLE 15.8 Six types of operators

Operations	Characteristics	Operators	Example(s)
Logical	Seven logic operations on Bit or Boolean data objects	and, nand, or, nor, not, xor, xnor,	<p>A and B $Y \leftarrow A \text{ and } B$; -- a transfer to Y after the logic operation</p> <p>$Y \leftarrow A \text{ nand } (B \text{ nand } D)$; -- a transfer to Y after the logic operation first B nand D and then between A nand the result of previous nand operation. Use of bracket is must here.</p>
Six relational operators	<ol style="list-style-type: none"> Four relations $<$, $>$, \leq, and \geq between the data objects, the result of operation is Boolean. Four operations on any of four scalar type (integer, floating point, enumerated, physical) Four operations on discrete array type 	$<$, $>$, \leq , \geq [Comparison from left to right.]	<p>(input > 0); -- greater than. Result is true or false.</p> <p>(input ≥ 0); -- greater than or equal to Result is true or false</p> <p>"Welcome to Garden" \leq "Welcome to Home"; -- result is true</p> <p>$('0', '1') \geq ('1', '0')$; -- result is false</p>
Shift and rotate operators	<ol style="list-style-type: none"> Two relations, $=$, \neq the result of operation is Boolean. $=$, \neq, can be operated on any type except the file type. 	$=$, \neq ,	<p>if (supplyCurrent = 5 mA); -- equal</p> <p>When (supplyVoltage \neq 5 V); -- not equal</p>

	3. Shift right arithmetic		"10000011" sra 1 is "11000001"; -- Put 1 at left most significant bit and shift right 1-bit "10000011" sra 4 is "11111000"; -- Put 1111 at four left significant bits and shift right four bits "00000011" sra 1 is "00000001"; -- Put 0 at left most significant bit and shift right 1-bit "00000011" sra 4 is "00000000"; -- Put 0000 at four left significant bits and shift right four bits
	4. Shift left arithmetic		"10000011" sla 1 is "00000110"; -- same as sll 1 "10000011" sla 4 is "00110000"; -- same as sll 4
	5. Rotate right		"10000011" ror 1 is "11000001"; -- put right most significant bit as leftmost significant bit and srl 1 step "10000011" ror 4 is "10001000"; -- put right four significant bits as left significant bits and srl 4 step
	6. Rotate left		"10000011" rol 1 is "00000111"; -- put left most significant bit as right most significant bit and sll 1 step "10000011" rol 4 is "00111000"; -- put left four significant bits as right significant bits and sll 4 step
Addition	1. Addition 2. Subtraction 3. Concatenation	&, +, -	"Welcome" & " " & "to come any time"; -- results is a concatenated array of characters (string) to "Welcome to come any time" 5 + 11; -- result is 16 5 - 11; -- result is -6
Multiply	1. Multiply 2. Division 3. Modulus result is $x - y \times N$ where N is some number which after multiplication brings $y \times N$ closest to x but less than x and resulting sign of result is sign of x .	* , /, mod, rem	5 * 11; -- result is 55 15 / -3; -- result is -5 11 mod 5; -- result is 1 11 mod 5; -- result is 1 11 mod -5; -- result is -1 15 rem -4; -- result is 3

	4. Remainder $x - (x/y) \times y$ $(x \text{ rem } y)$ results in division and find remainder and resulting sign of result is sign of x)		
Miscellaneous	1. Exponentiation 2. Absolute	**, abs	5.0 **-2; -- result is 0.05 abs (12); -- result is 12 abs (-2); -- result is -2

■ EXAMPLES

Example 15.1

What are the meanings of VHDL keywords: entity, component, system, model, text model, Boolean operations, strongly typed language, case sensitivity, construct, array, file, library, testbench, synthesis, architecture, process, structure, behaviour, and multipurpose modeling?

Solution

Table 15.9 gives the meaning of VHDL keywords.

TABLE 15.9 VHDL keywords

VHDL	Meaning
Entity	A section of circuit or a circuit or a component or a system. [A system consists of one or more components. A component consists of one or more circuits. A circuit may consist of several interconnected sections.]
Component	A component consists of one or more circuits. A set of components form a system.
System	A set of components, circuits, and sections of the circuits are assembled together which behave according a defined program, rules and sequence of actions.
Model	Set of codes which describes, can be translated into logic gates and wires, map to an electronic design such as Counter, Programmable Read-Only Memory (PROM) or Programmable Logic Device (PLD) or Field Programmable Gate Array (FPGA) or Application Specific Integrated Circuit (ASIC).
Text Model	A model written as text, which describes a circuit or logic.
Boolean-Operations	Extended operations using xnor, nand, and nor.
Strongly typed language	Types of the operands and result after operation must be matched. ¹
Case insensitivity	No case sensitivity in VHDL (means MuX, mux, muX are considered as same in the language).
Construct	A description of concurrent assignments that can be used to handle parallel hardware in the design. A construct may have arithmetic statements, signal assignments, and statements to instantiate component.

Array	A set of elements with each element accessible by an index. VHDL permits ascending direction as well as descending direction of the indices in an array.
File	A set of records with input and output capabilities.
Library	A module which is imported in a model.
Testbench	A set of simulation models which verify a design.
Synthesis	A stage after simulation or use of testbench that gives a real device.
Architecture	Each entity has a particular architecture. Architecture consists of constructs.
Process	A declaration in VHDL which is defined by specifying a set of concurrent or sequential assignments.
Structure	Structure of an entity is described by set of interconnected components, circuits or sections of the circuits. For example, Adder may have two bits plus carry adder (full_adder) as a component. It may have adder plus other components that gives a structure.
Structure model	A description of an entity in terms of set of interconnected components.
Behaviour model	A model which describes behaviour of the required system to be described (modeled) and verified (simulated).
Multipurpose modeling	A set of parameters, blocks, and interconnection structure which are usable in multiple projects after tuning (minor changes and modification to model a new project).
Probability	Once created, a computing device project can be ported on another (CMOS or HCMOS or n-MOS or other changed technologies can be used with same project files or designed project.

¹Does not allow literals and objects of different types to be mixed. For example, $3.141 + 2$ not allowed as it mixes real number type data with integer type.

Example 15.2 What are the meanings of VHDL keywords: register, port, RTL design, and RTL synthesis?

Solution

Meanings of register, port, RTL design, and RTL synthesis are as listed in Table 15.10.

TABLE 15.10 VHDL meanings of the register, port, RTL design and RTL synthesis

VHDL	Meaning
Register	A 1-bit storage unit in RTL model. It can be considered as one <i>D</i> flip-flop (<i>FF</i>) or a transparent latch. An input or output in a circuit is considered as state at a register of 1-bit. It can be a set of multiple <i>D</i> flip-flops (<i>FF</i>) or a transparent latches. A multi-bit inputs and outputs are the states at a multi-bit register.
Port	<i>RTL design of VHDL</i> considers <i>port</i> as one which gets signals (inputs) from the registers (each of 1-bit register) through interconnection(s) and which gives signal (outputs) to a register(s) through interconnection(s). It can be considered as having one or more <i>D</i> flip-flops (<i>FF</i>).

15.22 Digital Systems: Principles and Design

RTL design	A design of VHDL based on ports which get signals (inputs) from the registers (each of 1-bit register) through interconnection(s) and which gives signal(output) to a register(s) through interconnection(s).
RTL synthesis	Synthesis means finding the gate-level interconnections for a circuit or component or system. Synthesis of the design in terms of real circuit consisting of the gates and wires after the modeling and verification. It enables automation of the hardware design.

Example 15.3

1. Give four examples of permitted basic identifiers.
2. Give three examples of not permitted basic identifiers.
3. Give examples of identical basic identifiers.
4. Give five examples of extended identifiers.
5. Give an example of distinct extended identifiers.
6. Give five examples of keywords.
7. Give examples of distinct extended identifier and keyword.
8. Give examples of how to describe comments in the codes.

Solution

1. Examples of permitted basic identifiers in VHDL are `Q`, `data_bus`, `string4`, `muxCircuit`.
2. Examples of not permitted identifiers in VHDL are `data_bus_signal`, `_Q`, `4thInput`, `error_`, `Logic_Circuit_C`.
3. `nand`, `NAND`, `NaNd`, `nAnd`, are considered as identical identifiers. `Circuit_M`, `circuit_m`, `CirCuit_M`, `cIrCuiT_m` are considered as identical identifiers in the VHDL codes.
4. `_RAM2\`, `\addciruit\`, `_adder1\`, `\muktiplierN_`, `\8_Cmos\` are extended identifiers.
5. `\circircuitc\` and `\CiruitC\` are distinct extended identifiers and are not identical.
6. `case`, `sll`, `mod`, `or`, `and`, `library` are the examples of keywords.
7. `architecture` and `\architecture\` are district identifiers. Where, `architecture` is a keyword but `\ architecture \` is not a keyword.
8. (a) `constant A, B: in STD_LOGIC; -- Inputs are A and B and have standard logic values 0 or 1`
(b) `architecture behaviour for Counter_C is -- Synchronous counter signals and process`

Example 15.4

Describe data objects, constant, variable, signal, and file used in the VHDL.

Solution

1. Constant is used for declaring a constant value or values which do not change in a program for a system, for example, gate delay, address bus width, data bus width, and inputs in an half adder logic block.

2. Variable is used for declaring a value or values which may change in a program for a system, for example, signals A, B, D and Y .
3. Signal is considered as a wire in a circuit or modeled as flip-flop which is an output for another input in RTL model.
4. File is used to save a sequence of values. For example, file Counter1: std_logic_file; file component1_architecture: text open write_mode.

Example 15.5

1. Give examples of constants. How are they declared?
2. Give examples of variables. How are they declared?
3. Give examples of signals. How are they declared?
4. Give examples of a file. How are they declared?

Solution

1. constant pulseperiod: time = 200 ns; constant RAM_Memory: integer = 65536;
2. VARIABLE A, B, D: in STD_LOGIC; Y_1 : out STD_LOGIC; variable msb, lsb: Boolean;
3. Signal channel_output: Bit; Signal out_Bus: Bit_Vector (0 to 7);
4. file counter_entityandarchitecture: Text open Write_Mode;

Example 15.6

Write for an entity fulladder for a full adder circuit of two operand bits and carry in which gives the result F and carry out.

Solution

```
library IEEE; -- VHDL using IEEE standard libraries
use IEEE.std_logic_1164.all; -- import std_logic from the IEEE
library
entity fulladder is
    port (A, B, CY_In: in STD_LOGIC; F, CY_Out : out STD_LOGIC);
end fulladder;
```

Example 15.7

Write RTL design codes for an entity decoder3. It is for a three address input decoder with three gate inputs for the output control and eight chip select outputs. When gate control signals select the decoder3 then one of the output is 0 and remaining seven outputs are in tri-state.

Solution

```
library IEEE; -- VHDL using IEEE standard libraries
use IEEE.std_logic_1164.all; -- import std_logic from the IEEE
library
entity decoder3 is
    port (A0, A1, A3, G1, G2, G3 : in std_logic; Y0, Y1, Y2,
          Y3, Y4, Y5, Y6, Y7: out std_logic);
end decoder3;
```

15.24 Digital Systems: Principles and Design

Example 15.8 Declare for architecture of an entity XNOR_L. Use RTL design and data flow model.

Solution

```
architecture RTL of XNOR_L is
begin
    Yxor <= A and not B) or (not A and B); -- xor operation
    intermediate signal
    Y = not Yxor; -- xnor operation result
end RTL;
```

Example 15.9 Write for an entity CounterN. It is an n-bit counter with signals inputs for the counting and start. The counters give N outputs.

Solution

```
library IEEE; -- VHDL using IEEE standard libraries
use IEEE.std_logic_1164.all; -- import std_logic from the IEEE
library
constant N: INTEGER := 8;
entity CounterN;
port(CountIn, CountStart: in BIT_VECTOR(0 to N-1);
CountsValue: out BIT_VECTOR(0 to N-1)); -- declare port with
clock input and counter start input and N-bit output.
end CounterN;
```

Example 15.10 Write RTL design codes for an entity Nbit_AdderComponent. Nbit_Adder is an n-bit adder circuit. It consists of n full adders. Each full adder has two inputs a and b. Nbit_Adder generate a carry.

Solution

```
library IEEE; -- VHDL using IEEE standard libraries
use IEEE.std_logic_1164.all; -- import std_logic from the IEEE
library
constant n: INTEGER := 8;
entity NAND_L is
port(A: in Std_Logic; B: in Std_Logic; Y: out
Std_Logic);
end NAND_L;
entity Nbit_AdderComponent
port(a, b: in BIT_VECTOR(n - 1 downto 0); -- declare two
inputs for each full adder circuit
c: out BIT_VECTOR(N-1 downto 0); -- declare output for each
full adder circuit
cy: out BIT); -- declare carry output after n-bit addition
end Nbit_AdderComponent;
```

Example 15.11

Write architecture of a full adder. (Write RTL architecture of an i-th full_adder_i in an n-bit adder in data flow model.) Each full adder has two bit for addition. Each full adder has input carry and results in a carry out.

Solution

```
architecture RTL of full_adder_i is
begin
    S <= (a xor b) xor cyIn; -- combinational logic expression with xor
    operations
    cyOut <= (a and b) or (a and c) or (b and c);
end RTL;
```

Example 15.12

Write RTL design codes for an architecture and process in fullAdder1 of Example 15.11 behaviour model.

Solution

```
library IEEE; -- VHDL using IEEE standard libraries
use IEEE.std_logic_1164.all; -- import std_logic from the IEEE
library
entity fullAdder1 is
    port (a, b, cyIn: in std_logic; S, cyOut: out std_logic);
end fullAdder1;
architecture for fullAdder1 is
begin
    process (a, b, cyIn) -- Define the assignments with a, b and
    cyIn as inputs to generate S and cyOut
        begin
            If (a = '0') and (b = '0') and (cyIn = '0') then (S = '0'; cyOut ='0');
            -- sum of input is 0
            If (a = '0') and (b = '0') and (cyIn = '1') then (S = '1'; cyOut ='0');
            -- sum of input is 1
            If (a = '0') and (b = '1') and (cyIn = '0') then (S = '1'; cyOut ='0');
            -- sum of input is 1
            If (a = '0') and (b = '1') and (cyIn = '1') then (S = '0'; cyOut ='1');
            -- sum of input is 0
            If (a = '1') and (b = '0') and (cyIn = '0') then (S = '1'; cyOut ='0');
            -- sum of input is 1
            If (a = '1') and (b = '0') and (cyIn = '1') then (S = '0'; cyOut ='1');
            -- sum of input is 0
            If (a = '1') and (b = '1') and (cyIn = '0') then (S = '0'; cyOut ='1');
            -- sum of input is 0
            If (a = '1') and (b = '1') and (cyIn = '1') then (S = '1'; cyOut ='1');
            -- sum of input is 1
        end process;
end fullAdder1;
```

Example 15.13

There are 10 inputs InNum0 to InNum9 to a circuit, Circuit_ASCII_NUM. It generates ASCII codes between 48 to 57 (hexadecimals 30H–39H) Declare entity, architecture, and process for Circuit_ASCII_NUM. Use behaviour model.

Solution

The ASCII numbers 48–57 correspond to hexadecimal 30H–39H. It means outputs are between bit vectors 00110000 to 00111001 when inputs are between bit vectors 011111111 to 111111110.

```
library IEEE; -- VHDL using IEEE standard libraries
use IEEE.std_logic_1164.all; -- import std_logic from the IEEE
library
constant m: INTEGER:= 10; -- m = 10 for the inputs
constant n: INTEGER:= 8; -- n = 8 for the outputs
entity Circuit_ASCII_NUM;
port (InNUM: in STD_LOGIC_VECTOR(0 to m - 1);
      ASCIIINum: out STD_LOGIC_VECTOR(0 to n - 1));-- declare port
      with m-bit In_NUM and n bit output.
end Circuit_ASCII_NUM;
architecture for Circuit_ASCII_NUM is
begin
process (InNum, ASCIIINum) -- Define the assignments with
      ASCIIINum to generate ASCIIINum
begin
  if (InNum = "111111110" then ASCIIINum = "00110000"; --
when InNum0 = 0 then output is ASCII code 48.
  if (InNum = "111111101" then ASCIIINum = "00110001"; --
when InNum0 = 0 then output is ASCII code 49 .
  :
  if (InNum = "101111111" then ASCIIINum = "00111000"; --
when InNum0 = 0 then output is ASCII code 56.
  if (InNum = "011111111" then ASCIIINum = "00111001"; --
when InNum0 = 0 then output is ASCII code 56.
  end process;
end Circuit_ASCII_NUM;
```

Example 15.14

1. Which of the type declaration correct? (a) type tele_digit is ('x', '1', '2', '3', '4', '5', '6', '7', '8', '9')
2. What are the highest and lowest integers?

Solution

1. Correct. It is enumerated data type. tele_digit is a valid identifier.
2. $(2^{31} - 1)$ highest and lowest integer and highest and $-(2^{31} - 1)$ lowest integer in VHDL

Example 15.15

1. Declare a data type for three states ‘0’, ‘1’, and ‘z’.
2. Give example of array for logic states Q and Q_1 in a flip-flop.
3. Give example of arrays of 50 values of currents in mA at inputs and outputs.
4. Give example of arrays of 20 values of voltages in mV and V.

Solution

```

1. type ThreeStates is ('0', '1', 'Z');
2. signal QQ1: Bit_Vector(0 to 1);
3. variable i : Integer
   i:= 50
   type CValues is array integer (0 to i-1)
   units
   mA;
   end units
   variable inputCurrentValues, Output CurrentValues: CValues
4. variable j: Integer
   j:= 20;
   type VValues is array integer (0 to j-1)
   units
   mV;
   V = 1000 mV;
   end units
   variable InVoltageValues, OutVoltageValues: VValues

```

Example 15.16

1. Give example of access type of data in memory buffer of 1024 bytes.
2. Give example of file type.

Solution

1. type memorybuffer is array (0 to 1023, 0 to 7) of Bit;
 type memorybuffer_PTR is access memorybuffer;
2. type ReceivedBits is file of STD_LOGIC_VECTOR (0 to 127)

Example 15.17

The operations are given in Table 15.11 column 1. Write the result in column 2. Write the operation whether it logical or relational or shift or addition or multiply in column 3.

Solution

In Table 15.11 column 2 gives the result.

15.28 Digital Systems: Principles and Design

TABLE 15.11 Six types of operators

Operations	Result	Operation
$A := '1'; B := '1'$	'0'	Logical
$A \text{ xor } B$		
$Y <= A \text{ nand } B; \text{ -- a transfer to } Y$	'0'	Logical
after the logic operation		
$Y <= A \text{ nor } (B \text{ nor } D)$	nor operation of B with D and then result obtained performs nor operation with A .	Logical
$('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '_) >=$	True because vector length at left is more than the right	Relational
$('U', 'X', '0', '1', 'Z', 'W', 'L', 'H');$		
$('U', '0', '1', 'Z', 'W', 'L', 'H', '_) <= ('U', 'X', '1', 'Z', 'W', 'L', 'H');$	False because '0' is at higher index than 'X' in the std_logic position	Relational
"Welcome to Garden" <= "Welcome to Home";	True because G has position less than H in the array of characters	Relational
$('0', '1') >= ('U', '0');$	True because U has lower position in std_logic_vector	Relational

Example 15.18 Find the result of operations slr 2, sra 3 and rol 2 on “10000111”.

Solution

-- operations on BIT_VECTOR. Result of
 “10000111” slr 2 is “00011100”
 “10000111” sra 3 is “11110000”
 “10000111” rol 2 is “00011110”

Example 15.19 Find the result of the following operations.

1. $1.3 ** 2$,
2. $-15 \bmod 5$,
3. $5 \bmod 2$;
4. $10/3$
5. $2.0 * 3.0$.

Solution

1. $1.3 ** 2$ is 1.69.
2. $-15 \bmod 5$ is 2.
3. $5 \bmod 2$ is 1.

4. $10/3$ is 3
5. $2.0 * 3.0$ is 6.0.

Example 15.20

Find the Boolean expression for the following operations:

```
variable t1, t2, t3 : Bit;
t1:= A and B;
t2:= t1 nor C;
t3:= t1 or t2;
Z = not t3;
```

Solution

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

■ EXERCISES

1. Show by block diagram steps to synthesize Boolean function $F = A \cdot \overline{B} \cdot C + \overline{A} \cdot C$
2. Show registers (1-bit ports) for synthesizing logic circuit of $A \cdot \overline{B} \cdot C \cdot D + \overline{A} \cdot \overline{C} \cdot D$ using RTL VHDL.
3. Give VHDL codes for the data flow for A xor B .
4. Give VHDL codes for the entity for $A \cdot \overline{B} \cdot C + \overline{A} \cdot \overline{C} \cdot D$
5. Give codes for the architecture for A xor $\overline{B} \cdot C + \overline{A} \cdot \overline{C}$
6. Give VHDL codes for the process in a decoder, which has three (address) inputs and eight outputs. Output bit is 1 at selected address as per the inputs and other seven bits are 0.
7. Give codes for the entity and architecture for a multiplexer, which has three (address) inputs and eight logic inputs and one output as per the address at input [Term address and channel are same in this case].
8. Give codes for the architecture for $A \cdot \overline{B} \cdot C + \overline{A} \cdot \overline{C}$
9. Give examples for data types in Table 15.12.

TABLE 15.12 Three examples of predefined array data types

Array data type	Characteristics	Values
String	Array consisting of number of sequentially arranged characters	
Bit_Vector	Array consisting of number of sequentially arranged bit values	
std_ulogic_vector	Unconstrained array type	

15.30 Digital Systems: Principles and Design

10. Fill in the blanks in Table 15.13.

TABLE 15.13 Six types of operators, characteristics, operators and examples

Operations	Characteristics	Operators	Example(s)
Logical	Logic Operations of and, nand, or, nor, not, xor, xnor	and, nand, or, nor, not, xor, xnor,	<p>A and B $Y \leq A \text{ and } B$;-- a transfer to Y after the logic operation</p> <p>$Y \leq A \text{ nand } (B \text{ nand } D)$;-- a transfer to Y after the logic operation first n B nand D and then between A and the result of previous nand operation. Use of bracket is must here.</p>
Six Relational operators	<p>1. Four relations <, >, <=, <=between the data objects, the result of operation is Boolean,</p> <p>2. Four operations on any of four scalar type (integer, floating point, enumerated, physical)</p> <p>3. Four operations on discrete array type</p> <p>1. Two relations, =, /= the result of operation is Boolean.</p> <p>2. =, /=, can be operated on any type except the file type.</p>	<, >, <=, >= [Comparison from left to right.] =, /=,	if (supplyCurrent = 5 mA);-- equal When (supplyVoltage /= 5 V);--not equal)
shift and rotate operations	Shift left logical Shift right arithmetic Shift left arithmetic	srl, sll, sra, sla, ror, rol	--operations on BIT_VECTOR. Result of "10000011" srl 1 is "01000001";-- 0 at maximum significant bit and shift right one bit "10000011" srl 4 is "00001000" ;-- 0000 at 4 maximum significant bits and shift right one bit "10000011" ror 1 is "11000001";--put right most significant bit as leftmost significant bit and srl 1 step "10000011" ror 4 is "00111000" --put right four significant bits as left significant bits and srl 4 step "10000011" rol 1 is "00000111";--put left most significant bit as right most significant bit and sll 1 step "10000011" rol 4 is "00111000" --put left four significant bits as right significant bits and sll 4 step

Addition	$\&, +, -$	
		0 * 11; -- results is
1.	Multiply	
2.	Division	
3.	Modulus result is $x - y \times N$ where N is some number which after multiplication brings $y \times N$ closest to x but less than x and resulting sign of result is sign of x .	$*, /, \text{mod}, \text{rem}$
		12 / -3; -- results is
		16 mod 5; -- result is
Multiply		19 mod 5; -- result is
4.	Remainder $x - (x/y) \times y$ ($x \text{ rem } y$ results in division and find remainder and resulting sign of result is sign of x)	13 mod -5; -- result is
		16 rem -4; -- results is
Miscellaneous	1. Exponentiation 2. Absolute	$^{**}, \text{abs}$

■ QUESTIONS

1. Write features of 1076 IEEE standard VHDL versions.
2. Write features of 1164 IEEE standard VHDL.
3. Explain 9-valued logic. What do you mean by tri-state and don't care condition?
4. Describe std_logic.
 - (a) What do you mean by unresolved logic?
 - (b) What do you mean by resolved logic?
5. What is a port in VHDL?
6. How is the RTL coding done?
7. Explain entity, architecture, and process by an example of a half adder.
8. Explain entity, architecture, and process by an example of a full adder.
9. Explain entity, architecture, and process by an example of a multiplexer.
10. Explain entity, architecture, and process by an example of a decoder.
11. What do you mean by finite state machine?
12. What are the predefined data objects?
13. What is an array type data? How does it differ from scalar type data?
14. What do you mean by enumerated data type?
15. List all the relational operators and give one example for each.
16. How are the nand, $\&$, exponentiation, and mod operators used?

This page is intentionally left blank.

CHAPTER 16

VHDL—Packages, Sub Programs, and Sequential Circuits

OBJECTIVE

In this chapter, we will learn about the concept of package. A package is a collection of declarations. It has set of subprograms whose behaviours are described in the package body. Its declarations and subprograms commonly used by many design units.

We will learn how the subprograms described by the procedures and functions are written and how they are called during computations. How digital design libraries enable to use the design units in other designs and the VHDL coding for sequential circuits will also be studied.

16.1 PACKAGE

We learnt in Chapter 15 that a digital component or circuit or system consists of a design *entity* in VHDL. An entity may be external or internal and visible or hidden. Entity description in VHDL can include other entities, called components of the entity.

When an entity *A* is used in another entity *B*, then other entity *B* is said to use a *component A*.

Each entity associates with a *behaviour*. It is described by a description consisting of a set of assignments, algorithms, and/or process(es).

Behaviour of each entity is independently described by architecture. The *architecture* contains the description of the processes in the entity, contains the description of interconnected entities, and components.

16.2 Digital Systems: Principles and Design

A set of declarations in a *configuration* is used for description of a configuration for an entity.

Design unit is an entity or configuration or package declarations or package body. A package is a design unit for a digital system. The package can be used in several design files.

A *design file* uses the design units. Figure 16.1(a) shows four types of design units for a system design file in VHDL. Figure 16.1(b) shows design files and each design file containing design units. A package is one of the design units.

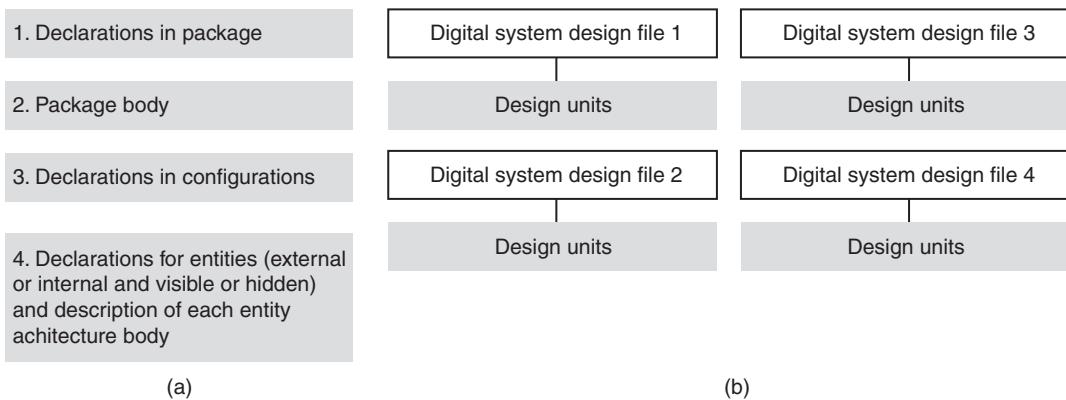


FIGURE 16.1 (a) Four types of design units for a system design file in VHDL (b) Design files containing a collection of design units.

Let us first learn what a package is and how to write a new package for common designs for the systems.

A *package* is a collection of the constants, declarations, and behaviour(s) of subprograms that can be commonly used by many design units. It is a design unit, which can be used by other design units. It consists of *declarations* and *body*. (A program means a set of instructions which are executed to get a set of results. A subprogram means a subset of instructions which are executed to get a subset of results of the program. VHDL subprogram consists of an algorithm. It does a set of computations.)

16.1.1 Package Declarations

Each *package* has declarations. The package has:

1. Set of declarations for subprograms (Subprograms use procedure and function clauses.).
2. Set of declarations for another package sourced into the package using use clause.
3. Set of declarations for constants, variables, signals, types, subtypes, files, components, attributes, and disconnection specifications.

Assume a package named package1. Its declarations are described as follows:

```
package package1 is
    ; -- declarations of design units into the package using use clause
    ; -- declarations for the subprograms in package1 using procdeure or function
        clause
    ; -- declarations of constants, variables, signals, types, subtypes, files, compo-
        nents, attributes
    ; -- declarations of attribute specifications
    ; -- declarations of disconnection specifications
```

A package can have multiple declarations and also include data types, subtypes, and subprogram declarations. Consider an example of `Text.io` which is as package in standard library named STD.

Example	Describe <code>text.io</code> package.
----------------	--

Solution

`Text.io` enables the read of inputs from a file and write of outputs to the file. It defines type for the input lines and output lines. It defines type for text inputs/outputs from or to the file. It also defines two data types—`side` and `width`. It has declarations for the subprograms for `writeline`, `readline`, `endline`, `read`, and `write`.

1. `Textio.vhdl` uses library `std`
2. It declares the following:

```
type line is access string; -- pointer to a string
type text is file of string; -- variable length records in ASCII
type side is (right, left); -- for right or left justifying of output fields
subtype width is natural; -- width of output fields
file input: TEXT open READ_MODE is "STD_INPUT";
file OUTPUT: TEXT open WRITE_MODE is "STD_OUTPUT";
```

`Textio` includes data types for the `String`, `Character`, `Bit`, `Bit_Vector`, `Integer`, `Natural`, `Positive`, `Real`, `Time` and `Delay` and `length`.

3. `Textio` declares procedures for the read of file. The procedures included for the line are as follows:

```
procedure readline (file F:text; L: inout line);
procedure read (L: inout line; value: out String; )
procedure read (L: inout line; value: out Bit; good: out
String)
```

Similar `readline` and `read` procedures are declared for the `Bit`, `Bit_Vector`, `character`, `Integer`, `Real`, `Boolean`, and `Time`.

4. `Textio` declares procedures for the write to the file that are as follows:

```
procedure writeline (file F:text; L: inout line);
procedure write (L: inout line; Value: in String; Justified:
in; Side:=RIGHT; Field: in; Width:=0);
```

Similar `writeline` and `write` procedures are declared for the `Bit`, `Bit_Vector`, `character`, `Integer`, `Real`, `Boolean`, and `Time`.

5. `textio` declares function for finding the line position. The function declaration is as follows:

```
function ENDLINE (L: in LINE) return BOOLEAN;
```

Point to remember

A package is a collection of the constants, declarations, and behaviour of subprograms such that the collection is commonly usable in many design systems. A package is a design unit of a digital system that can be used by other design systems. A package consists of the *constants*, *declarations*, and *body*.

16.1.2 Package Body

A *package body* consists of behaviour of subprgrams (procedure and functions). It also has declarations for the files, subprograms, types, subtypes, constants, and use clauses *that are not declared in declarations for the package*. A package body is not described in case the subprorams (procedure and functions) are not needed and all declarations for the constants, files, types, subtypes and use clauses that *are* present in declarations for that package.

Consider package1 body for which the declarations were given in Section 16.1.1. The package body describes behaviours of the subprograms.

```
package body package1 is
    ; -- declarations for subprograms which are not declared in package1
    declarations
    ; -- subprograms body
    ; -- declare using use
    ; -- declare constants, variables, signals, types, subtypes, files, components, and
       attributes which are not declared in declarations
    ; -- declare attribute and disconnection specifications which are not declared
       in declarations
```

Following is an example of `textio` package body:

Example Describe `textio` package body used in STD.

Solution

```
Package body textio is
    ; -- body for procedures declared in textio package declarations
    procedure readline(file F:text; L: inout line) is
        ; -- local declarations
        begin
            -- behaviour of readline
        end readline;
    procedure read(L: inout line; value: out String;) is
        ; -- local declarations
```

```

begin
    -- behaviour of read
end read;
:
; -- procedure bodies are for readln and read procedures are for Bit, Bit_
Vector, character, Integer, Real, Boolean, and Time
procedure writeline (file F:text; L: inout line) is
    ; -- local declarations
    begin
        -- behaviour of writeline
    end writeline;
procedure write(L: inout line; Value: inString; Justified:
in; Side:=RIGHT; Field: in; Width:=0) is
    ; -- local declarations
    begin
        -- behaviour of write
    end write;
:
; -- procedure bodies for writeln and write procedures are declared for
Bit, Bit_Vector, character, Integer, Real, Boolean, and Time.
; -- body for functions declared in textio package declarations
function endline (L: in LINE) return BOOLEAN is
    ; -- Behaviour of endline
end endline

```

Point to Remember

A package body consists of behaviour of subprograms (procedures and functions). The body has declarations for the logic vectors, signals, variables, files, subprograms, types, subtypes, constants, and use clauses *which are not declared in declarations for the package*. A package body is optional.

16.2 SUBPROGRAMS

VHDL program computations can be divided into subprograms. A subprogram is a subset of sequential statements. When it runs the assignments and statements execute to provide a subset of results. It can be either a procedure or a function.

VHDL subprogram consists of an algorithm. The algorithm is for executing a set of computations. It has sequential statements, declaration, and behaviour.

Sequential statements are in a sequence. The statements can be for declarations of constants, variables, signals, different data types, and subtypes. The statements can be for *Assign*, *Wait*, and *If* statements, *Case* statements, *Range* of loop, and *for* loop statements.

A subprogram has a body which includes the following:

1. It consists of declarations that are not declared elsewhere and are local. For example, the local declarations for the logic vectors, signals, files, strings, time, physical values, types, subtypes, constants, and use clauses.

2. It consists of behaviour. The behaviour is described by sequential statements between begin and end. A statement uses the logic vectors, signals, files, strings, time, physical values or constants data types for the assignment(s).

Consider subprog1 body. The body describes declarations for the arguments used in subprogram, for example, variables and the sequential statements. Subprogram behaviour is an algorithm. An algorithm is a set of sequential assignments or statements.

```
subprogram subprog1 is
    ; -- declarations for the arguments used in subprogram, for example, for
        variables
    begin
        ; -- sequential statements
    end subprog1
```

Point to Remember

A subprogram has a body which includes the declarations that are not declared earlier. Subprogram does a subset of computations in VHDL design units. A subprogram may be coded either as a procedure or function.

16.2.1 Procedure

Procedure is a subprogram, which is called for executing a behaviour described by a subset of statements and assignments.

1. When using a procedure there is no value returned to a parameter. The parameter may be a variable or signal or string or file or logic vector or signal or time or physical. However, the procedure can affect the parameter(s) in the list of parameters of the procedure after execution of algorithm (between begin and end of a set of statements).
2. A procedure body is given a name. There is a list of parameters. It is called arguments of a procedure. The arguments are specified within the left and right brackets. The list of parameters also includes the data type declaration for each parameter.
3. A procedure body is always described to enable the computations and assignments in the procedure. A procedure body description starts after the **is** clause.

The body description starts with the declarations for the logic vectors, signals, files, strings, types, subtypes, constants, and use clauses *that are not declared in declarations for the package* and are local. The set of statements and assignments are given between begin and end clauses. List of parameters is called arguments of a procedure. The arguments are specified within the left and right parenthesis.

Consider procedure1 body. The procedure1 body describes behaviour.

```
procedure proc_1 (list of parameters) is
    ; -- declarations for the arguments used in subprogram, for example, of variables
begin
    ; -- sequential statements
```

```
end proc_1
or
end procedure proc_1; -- procedure keyword is optional
```

Following is an example of using a procedure call to use the parameters for computations and give the computation result(s).

Example

Procedure find_xor4 () has Std_Logic in the arguments (list of parameters). These are M, N, O, P as inputs and Y_i as output. M, N, O, P are input std logic in the arguments. Y_i is out std logic in the argument. Find_xor4 () is for giving the result of four variable xor when find_xor procedure is called during a computation.

1. Show how are the declaration(s) of data types of the argument(s) made.
2. Show the procedure body of find_xor4 () .
3. Show how the procedure find_xor4 () called to perform four variable XOR operations on A, B, C and D which are inputs logic signals. Show how the procedure behaviour assigns the result to Q after the xor series of operations for the result.

Solution

1. find_xor4 declaration(s) for the argument(s) are made within the brackets for find_xor and are as follows:

```
procedure find_xor4 (Signal M, N, O, P: in Std_Logic; Signal Yi: out Std_Logic)
```

2. find_xor body is as follows:

```
procedure find_xor4 (Signal M, N, O, P: in Std_Logic; Signal Yi: out Std_Logic) is
```

;-- declarations of logic signals M, N, O, P and Y_i are already given in the arguments.

begin

$Y_i := M \text{ xor } N \text{ xor } O \text{ xor } P;$ -- find the four variable XOR of $M, N, O, P.$

end procedure find_xor4; -- Remember that the keyword procedure is optional

3. Signal A, B, C, D : in Std_Logic_Vector;

Signal Q : out Std_Logic;

find_xor (A, B, C, D, Q); -- call to procedure find_xor

Points to Remember

1. Procedure does a subset of computations in the VHDL design unit programs.
2. A procedure has a body.
3. A procedure has a list of the parameters (arguments). The data types for parameters (variable or signal or string or file or logic vector or signal or time or physical) are given within parenthesis after the procedure name.
4. The body describes *behaviour* between begin and end clauses.

16.2.2 Function

Function is a subprogram which is called for executing a behaviour described by a subset of statements and assignments. Function returns a value.

1. When using a function then there is a value returned to a parameter. (The parameter may be a variable or signal or string or file or logic vector or signal or time or physical. A parameter is assigned the returned value.)
2. The function after execution of algorithm (a set of statements between begin and end) may or may not affect the parameter(s) in the list of parameters of the function.
3. A function body is given a name. There is a list of parameters. List of parameters is called arguments of a function. The arguments are specified within the left and right brackets. List of parameters also include the data type declaration for each parameter.
4. A function body is always described to enable the computations and assignments in the function.

A function body consists of behaviour of the function after **is** clause. The body starts with declarations for the logic vectors, signals, files, strings, types, subtypes, constants and use clauses *which are not declared in declarations for the package* and are local. The set of statements and assignments are given between begin and end clauses.

Consider `func_1` body. The `func_1` body describes behaviour.

```
function func_1 (list of parameters) return parameter-type is
    ; -- declarations for the arguments used in subprogram, for example, of
       variables
begin
    ; -- sequential statements
    ; -- return statement
end funct_1
or
end function func_1; -- function keyword is optional
```

Following is an example of using a function to return a value of a parameter.

Example

Function `get_xor4()` has `Std_Logic` in the arguments (list of parameters) M, N, O , and P .

1. Show how are the declaration(s) for the argument(s) made.
2. Show the function body of `get_xor4()`.
3. Show how the function `get_xor4()` called to perform four variable XOR operations on A, B, C , and D which are inputs and are of std logic type. Show how the function behaviour assigns the result to Q .

Solution

1. The declaration(s) for the argument(s) are made within the brackets as follows:
`function get_xor4 (Signal M, N, O, P: in Std_Logic) return out Std_Logic`

2. The function body is as follows:

```
function get_xor4 (Signal M, N, O, P: in Std_Logic) return Std_Logic is
    Signal temp: out Std_Logic; -- declare a local signal temp.
    ; -- declarations of logic signals M, N, O, and P are already given in the
      arguments.
    begin
        temp := M xor N xor O xor P; -- find the four variable XOR of M, N, O,
        and P.
        return temp;
    end get_xor4;
```

3. Signal A, B, C, D : in Std_Logic_Vector;

Signal Q : out Std_Logic;

$Q := \text{get_xor4}(A, B, C, D)$; -- function get_xor returning the four variable xor into Q .

Points to Remember

1. Function does a subset of computations in VHDL design unit programs.
2. A function returns a parameter value.
3. A function has a body.
4. There is a list of the parameters (arguments) required in computations. The data types for parameters (variable or signal or string or file or logic vector or signal or time or physical) are given in parenthesis.
5. The body describes the behaviour between begin and end clauses.

16.2.3 Attribute

A constant or value or signal or type or function can be associated with an attribute. For example, an array has an attribute length (i.e., number of elements in an array). Attributes are of two kinds—pre-defined attributes and user-defined attributes.

16.2.3.1 Pre-defined Attribute

Five types of attributes in VHDL are pre-defined. Name of a variable or type or signal is prefixed to pre-defined attribute. An attribute consists of a quote mark (') followed by the name of the attribute. It is used to return a type of information about a constant, signal, variable or type or function.

Scalar data type attributes

A scalar data type can have a value in a range of values. It has six attributes: 'left, 'right, high, 'low, 'ascending, and 'value (String1):

1. leftmost,
2. rightmost,
3. highest,
4. lowest in the range of values from which a value of a scalar can take,

16.10 Digital Systems: Principles and Design

5. ascending attribute returns Boolean value = 1 if scalar values are in ascending order in the range,
6. value (String1) attribute returns a string (for example). value (1) returns “1”.

Signal attributes

Now consider attributes of a Signal.

Example

How does an attribute used for Signal SeqClk? Show it by an example of a clock. The clock is used in a sequential circuit.

Solution

Assume SeqClk is the sequencing clock input in a sequential circuit. SeqClk gives clock inputs. The clock input is of a pre-defined attribute or user-defined attribute. Table 16.1 gives the ten attributes of SeqClk signal.

TABLE 16.1 Ten attributes of SeqClk signal

Attribute	Returned parameter
SeqClk'event	Boolean value true or false when an event on the signal SeqClk takes place or does not take place.
SeqClk'last_event	The time interval which has elapsed since the last event on the signal SeqClk took place.
SeqClk'stable (T)	A Boolean value, true, if no event has occurred on the signal SeqClk during the interval T , otherwise returns a false. [T is optional, default $T = 0$].
SeqClk'transaction	A signal of Bit type which becomes 1 if 0 and 0 if 1 when there is a transaction on the Signal SeqClk takes place. A transaction means an assignment.
SeqClk'quiet (T)	A Boolean value, true, if no transaction took place on signal SeqClk during the interval T else false [Default $T = 0$].
SeqClk'active	A signal of Bit type which becomes 1 when there is a transaction on the signal SeqClk takes place.
SeqClk'last_value	The value of the signal SeqClk before the last event occurred on the signal.
SeqClk'last_active	The time interval which has elapsed since the last transaction on the signal.
SeqClk'delayed (T)	A signal that is the delayed version for the interval T after the original one. [Default T takes the value 0. It means no delay.]
SeqClk'delayed (0)	A signal that is without delayed version of the original one. [$T = 0$]

Following example shows how can event attribute be used.

Example

An attribute can associate a Sequential circuit SeqClk. [Clock means a series of sequences of 1 and 0 generating a series of instances (events).] How does a Signal SeqClk associate an attribute?

Solution

SeqClk'event associates SeqClk with an event. SeqClk'event returns a Boolean. It means associates SeqClk'event become true when the event takes place. Event is

state 0 to 1 transition or 1 to 0 transition. A function to a subprogram or process can use it as follows:

```

if (SeqClk'event) then
    ; -- Statements to be executed or assignments are to be made or functions to
      be called if
    ; -- SeqClk'event is +ve edge or -ve edge. SeqClk'event returns Boolean
      value 1 on event.
if (SeqClk'event or SeqClk = '0') then
    ; -- Statements to be executed or assignments are to be made or functions to
      be called if
    ; -- SeqClk'event is -ve edge. SeqClk'event returns Boolean value 1.
if (SeqClk'event or SeqClk = '1') then
    ; -- Statements to be executed or assignments are to be made or functions to
      be called if
    ; -- SeqClk'event is +ve edge. SeqClk'event returns Boolean value 1.

```

Point to Remember

A signal can have nine attributes, event, last_event, stable, quiet, transaction, active, last_active and delayed (T). [delayed () is a special case for T = 0 means wired connection.] An attribute of a signal is used as the name of signal and then attribute name.

Array attributes

An array data type consists of number of elements, each identified by an index. If an array has m elements; then m is $m - 1$ down to 0. It can also be written as 0 to $m - 1$ (Elements positions put in reverse order). We can also define array i^{th} elements by index i and then i changes from 0 to $i - 1$ and use i in loop statements.

A one dimensional array of integers, `ArrayN_Integer` can be described as follows:

```

type ArrayN_Integer is array(n-1 down to 0) of integer; -- Alternatively
type ArrayN_Integer is array(0 to n-1) of integer;
type ArrayN_Bit is array(n-1 down to 0) of Bit; -- Alternatively
type ArrayN_Bit is array(0 to n-1) of Bit;

```

A two dimensional array, `ArrayMxI` can be described as follows:

```
type ArrayMxI is array(0 to m-1, 0 to i-1) of integer;
```

`ArrayMxI` number of elements = $M \times I$.

The array has eight attributes: 'left, 'right, 'high, 'low, 'ascending, 'length, 'range and 'reverse range:

1. leftmost index,
2. rightmost index,
3. highest bound,
4. lowest bound,
5. 'ascending attribute returns Boolean value = 1 if array elements are in ascending order in the range,

6. 'length attribute returns number of elements,
7. 'range returns the range of the array element and
8. 'reverse_range returns the range of the array elements in reverse order.

Point to Remember

An array can have eight attributes: length, range, reverse range, ascending, lowest, highest, rightmost, and leftmost.

Function attributes

A function is called to return a value or element. Enumerated data type attributes are 'Pos (Value) and 'Val (Pos).

1. Attribute 'Pos (Val)

Example

Give an example of attribute 'Pos (Value).

Solution

- (i) `n: Integer;`
`X: Std_Ulogic_Vector ; -- array elements are 'U', 'X', '0', '1',`
`'Z', 'W', 'L', 'H', '_')`
`n = Std_Ulogic_Vector'Pos ('0');` -- returns the position number *n* of
'0', which is 2.
- (ii) **Character A** is enumerated data type from 127 discrete values
(*a*, ..., *z*, *A*, *..*, *Z*, '0', ...'9', '?', ',', '.', '#', '\$', ...)
`n: Integer;`
`n = Character'Pos ('C');` -- returns the position number of 'C', which
is 99.

2. Attribute 'Val (Pos)

Example

Give an example of attribute 'Val (Pos).

Solution

- (i) Assume presentmode is the value of File_open_kind type.
`presentmode: File_open_kind; -- three modes—Write_Mode,`
`Read_mode, Append_mode`
`presentmode := File_open_kind'Val (0);` -- returns the value of present
mode = Write_Mode
- (ii) **Character** is enumerated data type from 127 discrete values (*a*, ..., *z*,
A, *..*, *Z*, '0', ...'9', '?', ',', '.', '#', '\$', ...)
`p: Character;`
`p = Character'Val (99);` -- returns the character at position number of 99,
which is 'c'

3. Attribute 'Succ (Value1), 'Pred (Value1), 'RightOf (Value1), and 'LeftOf (Value1)

Four attributes are as follows:

- 'Succ (Value1) returns the value at one step more position than that of the value1.
- 'Pred (Value1) returns the value at one step less position than that of the value1.
- 'RightOf (Value1) returns the value at one step more than that of the value1.
- 'LeftOf (Value1) returns the value at one step less than that of the value1.

Point to Remember

A function can have attributes. The six attributes are Pos(Val), Val(Pos), Succ (Val), Pred(Val), RightOf (Val), and leftOf (Val).

16.2.3.2 User-defined Attribute

A user can define an attribute such that it is constant(s) of any type other than access or file type.

Example

Give user-defined attribute to current.

Solution

```
type current is range 200 to 500
units
    microampere;
end units
attribute microampere: current; -- current value is assigned an attrib-
    ute microampere.
```

Microampere is an attribute of current, defined by a user.

16.3 DESIGN LIBRARY

A library is stored and compiled design file for the system designs. Consider a design library named DigitalDesigns. DigitalDesigns declarations are as follows:

Library DigitalDesigns

; -- declarations of packages using use clause

The examples of standard packages used in the design library are given below. Following is an example of standard package. It has library and inherits several packages.

Example

- How does library STD use a package?
- Describe how IEEE standard library for the designs used. What are the packages used in IEEE?

Solution

1. library STD; -- standard library library std;
use std.standard.all; -- needed for bootstrap mode *all* means all packages in that.
use STD.textio; -- text in-out operations and functions
2. IEEE library is a design library. It must be installed for VHDL compilation and simulation. The packages used in library are as follows: The packages other than the standard must then be specifically accessed in VHDL source files. Standard are already compiled libraries. The source files are as follows:

```
library IEEE;
  use IEEE.std_logic_1164.all; -- std_logic_1164 package
  use IEEE.std_logic_textio.all;-- std_logic_textio package
  use IEEE.std_logic_arith.all;-- arithmetic operations package
  use IEEE.numeric_bit.all;-- numeric bit operations package
  use IEEE.numeric_std.all;-- numeric standard operators and functions package
  use IEEE.std_logic_signed.all; -- standard logic package for signed operations and functions
  use IEEE.std_logic_unsigned.all; -- standard logic unsigned operations and functions
  use IEEE.math_real.all; -- standard floating point operations and functions
  use IEEE.math_complex.all; -- standard complex variable operations and functions
```

16.4 SEQUENTIAL CIRCUITS

A general sequential circuit network has memory section and combination circuits at the memory section inputs and outputs. Memory section consists of sequential circuit based on flip-flops (*FFs*).

Figure 16.2 shows general sequential circuit consisting of Logic_StatesCircuit_SeqC and Logic_CombinationalCircuit_C. These components (circuits) can also be abbreviated as SeqC and Circuit_C. A general sequential circuit consists of SeqC. The output states \underline{Q} of SeqC are given as the inputs \underline{Y}_1 and after a Logic_CombinationalCircuit_C generates inputs \underline{Y} . The inputs \underline{Y} are given again to SeqC.

Assume the followings structure of a general sequential circuit. (Figure 16.2) Consider logic state vectors \underline{Y} , \underline{X} , and \underline{Q} . Logic vectors means an array of logic states:

1. Outputs of a sequential circuit flip-flops (in memory section) Q_0, Q_1, Q_2, \dots are represented by a logic vector \underline{Q} . The indices of the array are 0, 1, 2, ..., respectively.
2. Inputs to a sequential circuit X_0, X_1, X_2, \dots are represented by a logic vector \underline{X} . The indices of the array are 0, 1, 2, ..., respectively. Inputs to a sequential circuit also given from the outputs of combinational circuit section of

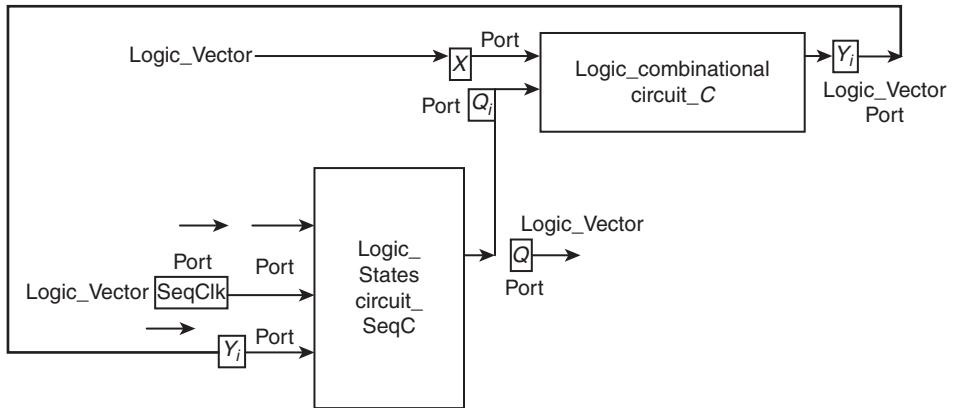


FIGURE 16.2 A general sequential circuit consisting of Logic_StatesCircuit_SeqC and the Output states \underline{Q} are given as inputs \underline{Y} after a Logic_CombinationalCircuit_C generates output \underline{Y} .

the sequential circuit. It means Y_0, Y_1, Y_2, \dots are also the inputs to memory section of the sequential circuit.

3. Inputs of a combinational circuit section of a sequential circuit are same as Q_0, Q_1, Q_2, \dots are represented by a logic vector \underline{Q} . The indices of the array are 0, 1, 2, ..., respectively.

Outputs from combinational circuit section of a sequential circuit Y_0, Y_1, Y_2, \dots are represented by a logic vector \underline{Y} . The indices of the array are 0, 1, 2, ..., respectively.

A general sequential circuit consists of Logic_StatesCircuit_SeqC component. The Output states are given as inputs \underline{Y} after a Logic_CombinationalCircuit_C component generates output \underline{Y} . These components (circuits) can also be abbreviated as SeqC and Circuit_C. The sequential circuit entity SeqCircuit has two components with following inputs and outputs:

1. One port SeqClk at SeqC (memory section).
2. One input port \underline{Y}_1 is also at SeqC.
3. \underline{Y}_1 also interconnects to output port \underline{Y} .
4. $\underline{Y}_1 = \underline{Y}$. \underline{Y} is output of Circuit_C.
5. SeqC (memory section) output is at \underline{Q} .
6. \underline{Q}_1 also interconnects to output port \underline{Q} . $\underline{Q}_1 = \underline{Q}$.
7. Circuit_C inputs are at port \underline{Q}_1 and \underline{X} .

16.4.1 General Sequential Circuit—Entity, Components, Architecture, and Processes

Following is an example of RTL model of a general sequential circuit. Sequential entity components are as per Figure 16.3. Figure 16.3 shows the design units needed in the sequential circuit described by two circuit components in Figure 16.2.

16.16 Digital Systems: Principles and Design

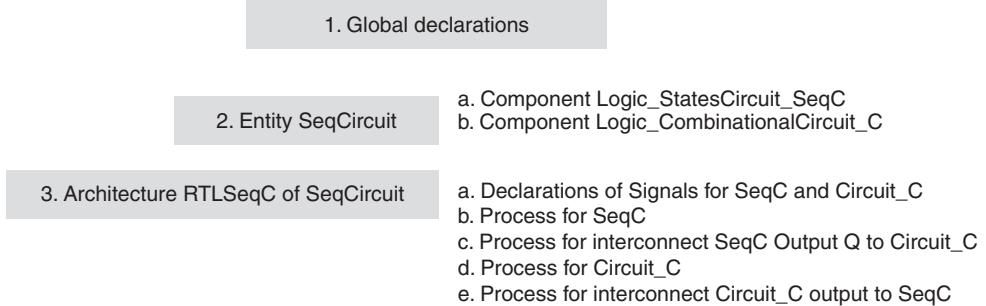


FIGURE 16.3 Design units for a general sequential circuit.

Example Describe the RTL entity, SeqCircuit and its components Logic_StatesCircuit_SeqC and Logic_CombinationalCircuit_C.

Solution

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_textio.all;
use std.textio.all;
entity SeqCircuit is
    n, m, i, j: Positive;
    ; -- Sequential circuit entity has two components (memory section and
       combinational circuit).
    component Logic_StatesCircuit_SeqC
        Port (
            SeqClk: in Std_Logic_Vector (m - 1 downto 0)::  

            Yi: in Std_Logic_Vector (j - 1 downto 0);
            Q: out Std_Logic_Vector (i - 1 downto 0));
    end component
    component Logic_CombinationalCircuit_C is
        Port (X: in Std_Logic_Vector (n - 1 downto 0);
              Qi: in Std_Logic_Vector (i - 1 downto 0);
              Y: out Std_Logic_Vector (j - 1 downto 0));
    end component
end entity;
```

Example Describe the architecture RTLSeqC of a general sequential circuit SeqCircuit.

Solution

```
architecture RTLSeqC of SeqCircuit is
    Tclk, Tdelay: Time;
    Tclk:= 50 ns; -- Interval T of Clock inputs
    Tdelay:= 4 ns;
```

```

tSeqC, tSeqplusComCircuit, tComCircuit: Time
tSeqC: = 20 ns; -- Delay in sequential circuit memory section
tComCircuit: = 10 ns; -- Delay in combinational logic section in sequential
circuit
tSeqplusComCircuit: = tSeqC + tComCircuit; -- Delay of Sequential Circuit
both sections;
process (SeqClk, Yi, Q); -- X, SeqClk, Yi are inputs and Q is output
    Signal SeqClk: in Std_Logic_Vector (m - 1 downto 0) after tclk ns;
    Signal Yi: in Std_Logic_Vector (j - 1 downto 0) after
tSeqCplusComCircuit;
    Signal Q: out Std_Logic_Vector (i - 1 downto 0) after tSeqC
begin
    ; -- process body for Logc_StatesCircuit_SeqC
end process
process (Q, Qi) -- Define the assignments with A and B as inputs to
generate Y
begin
    Qi <= Q' Delayed (0); -- Interconnect Qi input to Q output
process (Qi, X, Y) -- Define the assignments with A and B as inputs to
generate Y
    Signal X: in Std_Logic_Vector (n - 1 downto 0);
    Signal Qi: in Std_Logic_Vector (i - 1 downto 0);
    Signal Y: out Std_Logic_Vector (j - 1 downto 0) after tComCircuit;
begin
    ; -- process body for Logic_CombinationalCircuit_C
end process;
process (Yi, Y) -- Define the assignments with A and B as inputs to gener-
ate Y
begin
    Yi := Y' Delayed (0); -- Interconnect Y output to Yi input
end process;
end RTLSeqC;

```

16.4.2 Synchronous Sequential Circuit

Synchronous sequential circuit is a circuit in which the output \underline{Y} depends on present state \underline{Q} and present inputs \underline{X} at the clocked instances only. The memory section logic vector \underline{Q} activates at a clock event (instance of change from 0 to 1 or 1 to 0). \underline{Q} can also activate in a circuit of latches during an interval of clocking (state 1 of clock or state 0 of clock). The clock inputs are therefore given to the synchronous sequential circuit. SR , D , JK , and T are the examples of synchronous sequential circuit. Other examples of sequential circuits are *counters* and *registers*.

Synchronous sequential circuit outputs are \underline{Q} and inputs are \underline{X} , \underline{Y} and $\underline{\text{SeqClk}}$. Synchronous sequential circuit means as follows: The logic states of \underline{Q} can undergo transitions (0 to 1 or 1 to 0) at an instance (a clock event) in case of flip-flops or during an interval of clocking in case of latches.

Multiple sequences of \underline{Q} are generated when the input SeqClk are given to the memory section of *FFs*. SeqClk clock inputs are successive sequences of 1 and 0. Clock may have a sequence of 1s and 0s at the regular intervals or clock may have a sequence of 0s and 1s at the regular intervals from the reset or start.

Clock pulse means 1 for interval t_1 and 0 for interval t_0 . SeqClk 1 state is for certain duration t_1 or SeqClk 0 state is for certain duration t_0 .

Clock +ve edge means 1 for very small interval δt_1 .

Clock -ve edge means 0 for very small interval δt_0 compared to $t_0 + t_1 = T_{\text{clk}}$ = Clock sequences generation period = sum of interval of 1, t_1 and interval of 0, t_0 .

The clock repeat rate = $f_{\text{clk}} = (1/T_{\text{clk}})$. f_{clk} is called repetition rate of the sequencing clock, or clock frequency.

16.4.3 Sequencing Clock Circuit—Entity, Architecture and Processes

Following is an example of RTL model of a clock circuit for sequential circuits.

Example

A sequential circuit clock outputs are as follows: (Clock outputs will be used as inputs in synchronous sequential circuits). SeqClk0 is a series of sequences of 1 and 0 with sequence repeating each 50 ns. How do you entity SeqClk design unit? How do you architecture RTLSeqClk design unit of SeqClk? What process is executed at RTLSeqClk?

Solution

```
Tclk, tSeqC, TQuiet: Time;
Tclk := 50 ns; -- Interval T of Clock inputs
tSeqC := 25 ns; -- Delay in a not gate
TQuiet := 5 ns; -- Quiet for 5 ns
entity RTL SeqClk0 is
    Port (SeqClk0: in Std_Logic; SeqClk: out Std_Logic);
end entity;
architecture RTLSeqClk of RTL SeqClk0 is
type Y0: Boolean;
Signal SeqClk: out Std_Logic;
Signal SeqClk0: in Std_Logic; -- SeqClk is output as well as input after a duration.
process (SeqClk0, SeqClk, Y0)
    SeqClk0 <= SeqClk and not SeqClk; -- Initial value of SeqClk = 0;
    SeqClk <= not SeqClk after 4 ns; -- Initial value of SeqClk0 = 1;
    while (SeqClk'quiet (TQuiet)) loop ;-- Asynchronous loop
        SeqClk <= not SeqClk after tSeqC; -- Loop will execute and a sequence of
        ;-- 1 for tSeqC and 0 for tSeqC and clock0 has no stop.
    end loop;
end process;
end architecture RTLSeqClk;
```

16.4.4 Clock inputs for Flip-flop and Latch Synchronous Sequential Circuits

Synchronous sequential circuit memory section may consist of a flip-flop or latch. The clock inputs are as follows in the flip-flops and latches.

1. Clock inputs at a flip-flop

SeqClk clocking interval δt can be a defined duration of a logic 0 or 1 state (δt_0 or δt_1). A clocking instance (event) can be defined by clock input.

- (i) +ve edge (0 to 1 transition) δt_1 for very small interval $\ll T_{\text{clk}}$ or
- (ii) -ve edge (1 to 0 transition) $\delta t_0 \ll T_{\text{clk}}$ or
- (iii) Falling (1 to 0 transition) edge of a pulse $\delta t_0 \ll t_1$ or
- (iv) Rising (0 to 1 transition) edge of a pulse $\delta t_1 \ll t_0$.

2. Clock inputs at a latch

When latch is used in place of flip-flop, a clocking interval (event interval) can be defined by:

- (i) clocking interval t_1
- (ii) clocking interval t_0

16.4.5 Multiple Clock Signals from Main Clock

Main system clock is a clock that is used to generate other clock inputs (signals) also for the other circuits (entities). The other clock signals to other entities then have the series of sequences of 1s or 0s of clock inputs after division of clock frequency by 2, 4, and so on of the main clock.

Sometimes, the clock frequency which is required can be generated for $\frac{1}{2}f_{\text{clk}}$ or $\frac{1}{4}f_{\text{clk}}$, ..., $(1/10f_{\text{clk}})$, $(1/12f_{\text{clk}})$, ..., and so on. Then a main clock is used to generate other clock inputs for the other circuits (entities) requiring clock inputs at longer intervals. The other clock input to other entity then has a series of sequences of 1s or 0s of clock inputs after division of clock frequency by 2, 4, 10, 12, ..., and so on of the main clock.

Main clock frequency is rate of repetitions per second. The other clock frequencies (rate of repetition of sequences of 1 and 0) can be generated and are $\frac{1}{2}f_{\text{clk}}$ or $\frac{1}{4}f_{\text{clk}}$, ..., by division of frequency of the main clock. Main clock frequency f_{clk} divides by a clock divider circuit. Main clock period T_{clk} is time interval between a sequence of 1 and 0. The other clock periods can be generated twice ($2T_{\text{clk}}$), four times ($4T_{\text{clk}}$), and so on of the main by the *clock divider circuit*.

■ EXAMPLES

Example 16.1

Describe package `logicfunctions4` declarations for four logic functions and, nand, or, xor with `gate_delay = 4 ns`.

Solution

```
; -- Package declarations
library ieee;
use ieee.std_logic_1164.all;
package logicfunctions4 is
    ; -- Logic1AND declaration
    Entity logicgates4;
    Entity logicgates4 is
        component Logic1AND
            generic (Delay: Time:= 4 ns);
            port (A, B: in Std_Logic; X0: out std_logic);
        end component;
        component Logic1NAND
            generic (Delay: Time:= 4 ns);
            port (C, D: in std_logic; X1: out std_logic);
        end component;
        component Logic1OR
            generic (Delay: Time:= 4 ns);
            port (E, F: in Std_Logic; Y0: out std_logic);
        end component;
        component Logic1XOR
            generic (Delay: Time:= 4 ns);
            port (G, H: in std_logic; Y1: out std_logic);
        end component;
    end Entity logicgates4
end package logicfunctions4;
```

Example 16.2 Describe package logicfunctions4 body for four logic functions and, nand, or, xor with gate delay = 4 ns.

Solution

```
; -- Package body declarations
library ieee;
use ieee.std_logic_1164.all;
package body logicfunctions4 is
    ; -- Logic1AND body
architecture df_Lfunc4 of logicgates4 is
begin
    X0 <= A and B after DELAY;
    X1 <= C nand D after DELAY;
    Y0 <= E or F after DELAY;
    Y1 <= G xor H after DELAY;
end;
end package body df_Lfunc4;
```

Example 16.3

1. What is the use of package text.io.
2. What are declarations for the types and files in text.io.

Solution

1. The package text.io enables the user input/output. It defines input/output types for the line, text, side, and width. It has functions for writeline, readline, endline, read, and write.
2. Declarations for the types and files are as follows:

textio.vhdl uses library std.

```
type line is access string; -- pointer to a string
type text is file of string; -- variable length records in ASCII
type side is (right, left); -- for right or left justifying of output fields
subtype width is natural; -- width of output fields
file input: TEXT open READ_MODE is "STD_INPUT";
file output: TEXT open WRITE_MODE is "STD_OUTPUT";
```

Types are also included for String, Character, Bit, Bit_Vector, Integer, Natural, Positive, Real, Time and Delay and length.

Example 16.4

What are the procedures and functions in text.io package.

Solution

Procedures for read are as follows:

```
readline(file F:text; L: inout line);
read(L: inout line; value: out String); read(L: inout line; value: out String);
read(L: inout line; value: out Bit; good: out String)
```

Similar read procedures are for Bit, Bit_Vector, character, Integer, Real, Boolean and Time

Procedures for write are as follows:

```
writeline(file F:text; L: inout line);
write(L: inout line; Value: in String; Justified: in;
Side:=RIGHT; Field: in; Width:=0);
```

Similar read procedures are for Bit, Bit_Vector, character, Integer, Real, Boolean and Time.

Function is for finding the line position and is as follows:

```
ENDLINE (L: in LINE) return BOOLEAN;
```

Example 16.5

Describe how a function endline is used in VHDL program. The endline takes a line Line1 in the arguments, finds whether it is end of line character and if yes, then it returns a Boolean variable true i.e, IsEndLine as true else false. Show the use of function endline () of the package textio in standard library STD.

Solution

; -- function endline () of the package textio in standard library STD has function body as follows:

```
function endline (Line1:in Line) return BOOLEAN) is
```

```
; -- Behaviour of endline  
end endline
```

Consider two variables IsEndLine and line1. The following is the use of the function to find whether endline character exists in the line:

```
variable IsEndLine: Boolean;  
variable line1: in Line;  
isEndLine:= endline(line1);-- return true if endline character found else false
```

Example 16.6 Describe standard function To_01 () for conversion.

Solution

To_01 () converts ‘L’ into ‘0’ and ‘H’ into ‘1’. [‘L’ is logic 0 but weak drive, wired AND or wired OR connections allowed. ‘H’ is logic 1 but weak drive. If any value in the vector is ‘U’, ‘X’, ‘W’, or ‘Z’ or ‘-’ then it converts to either 1 or 0 as per the additional optional mapping argument xmap in the function To_01 (). A warning is also given during compilation.]

Example 16.7

1. Describe standard function To_Integer () for conversion.
2. Describe standard function To_Unsigned () for conversion.
3. Describe standard function To_Signed () for conversion

Solution

1. To_Integer () converts a data-type to integer.
2. To_Unsigned () converts a data-type to unsigned.
3. To_Signed () converts a data-type to signed.

Example 16.8 What are the requirements in numeric computations?

Solution

Following operators and functions are required in numeric computations for the vectors:

1. Logic operators
2. Relational operators
3. Shift and rotate operators
4. Arithmetic operators
5. Sign changing operators (abs and -)
6. Match function
7. Conversion operators
8. Resize function

Example 16.9 Describe numeric_std package.

Solution

Numeric_std package is used for the vectors. It is used in place of the ieee.std_logic_arith. It overrides the std_logic_vector. This package is a library package for VHDL design codes for the digital systems. The ieee.std_logic_arith does not enable mixing of operations on signed and unsigned functions.

This package includes definitions for the following:

1. Logic operators
2. Relational operators ($<$, $>$, \leq , \geq , $=$, \neq)
3. Shift and rotate operators (srl, sll, sra, sla, ror, rol)
4. Arithmetic operators ($+$, $-$, $*$, $/$, mod, and rem)
5. Sign changing operators (abs and $-$),
6. Match function [$\text{std_Match}(V_1, V_2)$]
7. Conversion operators [$\text{To_Integer}()$, $\text{To_Unsigned}()$ and $\text{To_Signed}()$]
8. Special translation function [$\text{To_01}()$]
9. Resize function $\text{resize}(V, n)$. [If vector V is signed then left most bits are filled with the sign bit at the msb. If vector V is unsigned then new left most bits are filled with the 0s.]

Std_Match function matches each element of two vectors. Any bit value ‘-’ (means don’t care) when compared with is ‘U’, ‘X’, ‘W’, or ‘Z’ then result obtained is false and if compared with ‘0’, ‘1’, ‘L’, and ‘H’ then result obtained is true.

$\text{To_Integer}()$ is the operator to convert a data-type to integer. $\text{To_Integer}()$, $\text{To_Unsigned}()$ is the operator to convert a data-type to unsigned. $\text{To_Signed}()$ is the operator to convert a data-type to signed.

$\text{To_01}()$ converts ‘L’ into ‘0’ and ‘H’ into ‘1’. [‘L’ is logic 0 but weak drive, wired AND or wired OR connections allowed. ‘H’ is logic 1 but weak drive. If any value in the vector is ‘U’, ‘X’, ‘W’, or ‘Z’ or ‘-’ then it converts to either 1 or 0 as per the additional optional mapping argument xmap in the function $\text{To_01}()$. A warning is also given during compilation.]

Resize function can be understood as follows: Assume a signed vector $SV8 = 0111\ 1110$. When resize function is used as follows: $SV16 = \text{resize}(SV8, 16)$ then $SV16$ becomes $0000\ 0000\ 0111\ 1110$ by extending sign bit 0 to left most eight positions. Assume a vector is $SV8 = 11111110$. When resize function is used as follows: $SV16 = \text{resize}(SV8, 16)$ then $V16$ becomes $1111\ 1111\ 1111\ 1110$ extending sign bit 1 to left most eight positions.

Example 16.10 Procedure find_notxor has Std_Logic_Vector A, B, C, D, \dots in the arguments. Procedure has Q out logic in the argument for the result of XOR and NOT. Show how the declarations for the arguments are made.

Solution

The declarations for the arguments are made as follows:

Signal $ABCD$: in Std_Logic_Vector (0 to $i - 1$); -- declare signal A, B, C, D standard logic values in vector indices 0, 1, 2 and 3.

Signal Q : out Std_Logic;

Example 16.11

1. Show how procedure body of `find_notxor()` is written.
2. Show how the procedure `find_notxor()` called to perform XOR operations on A, B, C , and D which are inputs and are elements of logic vector. The procedure behaviour assigns the result to Q after the not operation on the result.

Solution

1. Procedure body of `find_notxor()` is written as follows:

```
procedure find_notxor (Signal Yi; in Std_Logic_Vector (0 to  $i - 1$ );  
Variable i: Integer; Signal X: Std_Logic) is  
signal temp, M, N, O, P: Std_Logic; -- declare a local signal temp. These  
are assigned  
; -- intermediate result before not operation to find X  
begin  
; -- behaviour described by sequence of statements to assign values to  
M, N, O, P using the logic vector positions 0, 1, 2 and 3 of  $Y_i$   
temp := (M xor N) xor (O xor P); -- find the four variable XOR of  
M, N, O, P.  
X := not temp;  
end find_notxor
```

2. Procedure `find_notxor()` is called as follows:

```
Signal ABCD: in Std_Logic_Vector;  
variable i: Integer;  
Signal Q: out Std_Logic;  
i := 4;  
find_notxor (ABCD, i, Q); -- call to procedure find_notxor
```

Example 16.12

How do scalar six attributes used? Give examples.

Solution

1. When 'right attribute is after the name of a scalar-type value then returns the first or leftmost value of scalar-type in its defined range of the scalar_type values.
2. When 'left attribute is after the name of a scalar-type value then returns the last or rightmost value of scalar-type in its defined range of the scalar_type values.
3. When 'high attribute is after the name of a scalar-type value then returns the greatest value of scalar-type in its defined range of the scalar_type values.

4. When 'low attribute is after the name of a scalar-type value then returns the lowest value of scalar-type in its defined range of the scalar_type values.
5. When 'value (String1) attribute is after the name of a scalar-type value then returns the value of String1.
6. When 'ascending attribute is after the name of a scalar-type value then returns Boolean true or false depending on the existence of ascending range of the scalar_type values.

Example 16.13 How does a library used?

Solution

Suppose library of CMOS_logic is being used by an entity CMOS_Multiplier then it will be used as follows:

```
library CMOS_logic;
use work.CMOS_logic.package_gates.all;
entity CMOS_Multiplier is
```

Example 16.14 How is Numeric_std package used?

Solution

Numeric_std package is used as follows:

```
library ieee;
use ieee.std_logic_1164.all; -- standard IEEE standard 1164 pack-
age for unresolved logic
use ieee.numeric_std.all;-- Computations requiring signed, unsigned,
and arithmetic operators
```

Example 16.15 How does a library attribute is declared in a library?

Solution

```
library library_1;
use library_1.attributes.all;
```

Example 16.16 How does an attribute used? Show it by example of Signal *Q*: Std_Logic from memory section.

Solution

Table 16.2 gives the nine attributes of signal *Q* and a special case of T = 0.

16.26 Digital Systems: Principles and Design

TABLE 16.2 Nine attributes of signal Q and a special case of T undefined or 0

Attribute	Returned parameter
Q'event	Boolean value true or false when an event on the Signal Q takes place or does not take place.
Q'last_event	The time interval which has elapsed since the last event on the signal Q took place.
Q'stable (T)	A Boolean value, true, if no event has occurred on the signal Q during the interval T , otherwise returns a false. [T is optional, default $T = 0$]
Q'transaction	A signal of Bit type which becomes 1 if 0 and 0 if 1 when there takes place a transaction on the Signal Q. A transaction means an assignment.
Q'quiet (T)	A Boolean value, true, if no transaction took place on signal Q during the interval T else false [Default $T = 0$].
Q'active	A signal of Bit type which becomes 1 when there takes place a transaction on the signal Q.
Q'last_value	The value of the signal Q before the last event occurred on the signal
Q'last_active	The time interval which has elapsed since the last transaction on the signal.
Q'delayed (T)	A signal that is the delayed version for the interval T after the original one. [Default T takes the value 0. It means no delay.]
Q'delayed ()	A signal that is without delayed version of the original one. [T is = 0]

Example 16.17 How do the +ve clock to an *FF* and –ve clock edge used in generating outputs Q?

Solution

A positive clock edge and the action in flip-flop is described by statement:

```
if (SeqClk'event and SeqClk = '1') then;
    -- Sequential statements
```

A negative clock edge and the action in flip-flop is described by statement

```
if (SeqClk'event and SeqClk = '0') then;
    -- Sequential statements
```

Example 16.18 How do the +ve clock to a latch transparent and –ve clock to a transparent latch used in generating outputs Q?

Solution

State 1 clock input and the statements in latch is described by statement:

```
if (SeqClk = '1') then;
    -- Sequential statements
```

State 0 clock input and the statements in latch is described by statement:

```
if (SeqClk = '0') then;
    -- Sequential statements
```

Example 16.19 A sequential circuit clock gives outputs as follows: (Clock outputs are used inputs in synchronous sequential circuits)

SeqClk: = 0 if RST_Clk = 1 and SeqClk_Start = 0.

SeqClk: = 1 if SET_Clk = 1 and SeqClk_Start = 0

SeqClk is a series of sequences of 1 and 0 and sequence repeats after 50 ns if SeqClk_Start = 1 (RST_Clk can be 0 or 1 or SET_Clk can be 0 or 1.)

SeqClk1 is clock output which is not SeqClk means 180 degree out of phase SeqClk.

Assume clock interval T for one sequence of 1 and 0 is 50 ns for 25 ns each. How do you entity SeqClk design unit? How do you architecture RTLSeqClk design unit of SeqClk? What process is executed at RTLSeqClk?

Solution

```

Tclk, tSeqC: Time;
Tclk:= 50 ns; -- Interval T of Clock inputs
tSeqC:= 25 ns; -- Delay in a not gate
Tquiet:= 5 ns; -- an attribute of a signal that Tquiet period for which a signal
                ;--does not change then return true
entity RTLSeqClk is
    Port (RST_Clk, SET_Clk, SeqClk_Start: in Std_Logic;
          SeqClk, SeqClk1: out Std_Logic);
end entity;
architecture RTLSeqClk of RTLSeqClk is
    tSeqplusComCircuit:= tSeqC + tComCircuit; -- Delay of Sequential
                                                Circuit both sections;
    Signal SeqClk: in Std_Logic;
    Signal RST_Clk: in Std_Logic
    Signal SeqClk_Start: in Std_Logic
    Signal SeqClk1: out Std_Logic; -- In between output
    Signal SeqClk: out Std_Logic; -- SeqClk is output
    process (RST_Clk, SET_Clk, SeqClk_Start, SeqClk, SeqClk1);
        ; -- RST_Clk and SeqClk_Start are inputs. SeqClk1 and SeqClk are outputs
        SeqClk <= SeqClk and not SeqClk1; -- Initial value of SeqClk = 0;
        SeqClk1 <= SeqClk1 or not SeqClk1; -- Initial value of SeqClk1 = 1;
        while (SeqClk_Start and SeqClk'Quiet (Tquiet)) loop
            SeqClk <= not SeqClk after tSeqC; -- Loop will execute and a
            sequence of
            ; -- 1 for tSeqC and 0 for tSeqC. It will repeat till
            ;-- SeqClk_Start = 0 and clock stops.
            SeqClk1 <= not SeqClk;
            end loop
        if (RST_Clk = 1 and SeqClk_Start = 0) then
            SeqClk1 <= RST_Clk and not SeqClk_Start;
            SeqClk <= not SeqClk1;
        end if
        if (SET_Clk = 1 and SeqClk_Start = 0) then
            SeqClk <= SET_Clk and not SeqClk_Start;
            SeqClk1 <= not SeqClk;
        end if
    end process
end architecture RTLSeqClk;
```

Example 16.20

Describe a Moore sequential machine. Describe Moore sequential circuit entity and its components.

Solution

Figure 16.4 shows Moore machine sequential circuit consisting of `Moore_StatesMachine_SeqM` and `Logic_CombinationalCircuit_C` generating states \underline{Q} at sequences of the clock inputs consisting of `Moore_StatesMachine_SeqM` and `Logic_CombinationalCircuit_C`. One input port `SeqClk` interconnects `Moore_StatesMachine_SeqM` (memory section). Another input port \underline{Y}_i is at `Moore_StatesMachine_SeqM`. \underline{Y}_i also interconnects to output port \underline{Y} . $\underline{Y}_i \leq \underline{Y}$. \underline{Y} is output of `Logic_CombinationalCircuit_C`. `Moore_StatesMachine_SeqM` (memory section) output is at \underline{Q} . \underline{Q} interconnects the `Logic_CombinationalCircuit_C` input at port \underline{Q}_i . $\underline{Q}_i = \underline{Q}$.

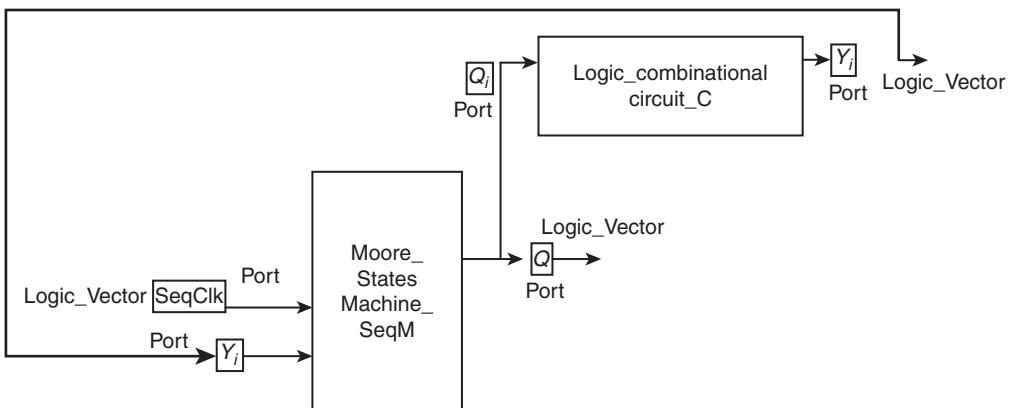


FIGURE 16.4 A Moore machine sequential circuit consisting of `Moore_StatesMachine_SeqM` and `Logic_CombinationalCircuit_C` generating states \underline{Q} at sequences of the clock inputs.

RTL model Moore sequential circuit entity has two components (memory section and combinational circuit). The entity can be described follows:

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_textio.all;
use std.textio.all;
Tclk, Tdelay: Time;
Tclk := 50 ns; -- Interval T of Clock inputs
Tdelay := 4 ns;
entity MooreSeqCircuit is
    n, m, i, j: Positive;
    ; -- Sequential circuit entity has two components (memory section and
       combinational circuit).
    component Moore_StatesMachinet_SeqM
        
```

```

Port (
SeqClk: in Std_Logic_Vector ( $m - 1$  downto 0)::;
 $Y_i$ : in Std_Logic_Vector ( $j - 1$  downto 0);
 $Q$ : out Std_Logic_Vector ( $i - 1$  downto 0));
end component;
component Logic_CombinationalCircuit_C is
Port (
 $Q_i$ : in Std_Logic_Vector ( $i - 1$  downto 0);
 $Y$ : out Std_Logic_Vector ( $j - 1$  downto 0));
end component;
end entity;

```

Example 16.21 Describe architecture RTLMooreSeqM of MooreSeqCircuit.

Solution

```

architecture RTLMooreSeqM of Moore_StatesMachinet_SeqM is
tSeqM, tSeqplusComCircuit, tComCircuit: Time
tSeqM: = 20 ns; -- Delay in sequential circuit memory section
tComCircuit: = 10 ns; -- Delay in sequential circuit combinational logic
section
tSeqplusComCircuit: = tSeqM + tComCircuit; -- Delay of Sequential
Circuit both sections;
process (SeqClk,  $Y_i$ ,  $Q$ ); --  $X$ , SeqClk,  $Y$  are inputs and  $Q$  is output
    Signal SeqClk: in Std_Logic_Vector ( $m - 1$  downto 0) after tclk ns;
    Signal  $Y_i$ : in Std_Logic_Vector ( $j - 1$  downto 0) after
tSeqMplusComCircuit;
    Signal  $Q$ : out Std_Logic_Vector ( $i - 1$  downto 0) after tSeqM
begin
    ; -- process body for Moore_StatesMachinet_SeqM
end process;
process ( $Q$ ,  $Q_i$ ) -- Define the assignments with  $A$  and  $B$  as inputs to
generate  $Y$ 
begin
     $Q_i := Q'$  Delayed (0); -- Interconnect  $Q_i$  input to  $Q$  output
end process;
process ( $Q_i$ ,  $Y$ ) -- Define the assignments with  $A$  and  $B$  as inputs to
generate  $Y$ 
begin
    Signal  $Q_i$ : in Std_Logic_Vector ( $i - 1$  downto 0);
    Signal  $Y$ : out Std_Logic_Vector ( $j - 1$  downto 0) after tComCircuit;
begin
    ; -- process body for Logic_CombinationalCircuit_C
end process;
process ( $Y_i$ ,  $Y$ ) -- Define the assignments with  $A$  and  $B$  as inputs to generate  $Y$ 
begin
     $Y_i := Y'$  Delayed (0); -- Interconnect  $Y$  output to  $Y_i$  input
end process;
end RTLMooreSeqM;

```

Example 16.22 Describe a Mealy sequential circuit.

Solution

Figure 16.5 sequential circuit entity MealySeqCircuit has two components with following inputs and outputs. A general sequential circuit consists of Mealy_StatesCircuit_SeqM component. The Output states are given as inputs \underline{Y} after a Logic_CombinationalCircuit_C component generates output \underline{Y} . These components (circuits) can also be abbreviated as SeqM and Circuit_C. The sequential circuit entity MealySeqCircuit has two components with following inputs and outputs.

1. One port SeqClk at SeqC (memory section).
2. One input port \underline{Y}_i is also at SeqC.
3. \underline{Y}_i also interconnects to output port \underline{Y} .
4. $\underline{Y}_i = \underline{Y}$. \underline{Y} is output of Circuit_C.
5. SeqC (memory section) output is at \underline{Q} .
6. \underline{Q}_i also interconnects to output port \underline{Q} . $\underline{Q}_i = \underline{Q}$.
7. Circuit_C inputs are at ports \underline{Q}_i and \underline{X} .

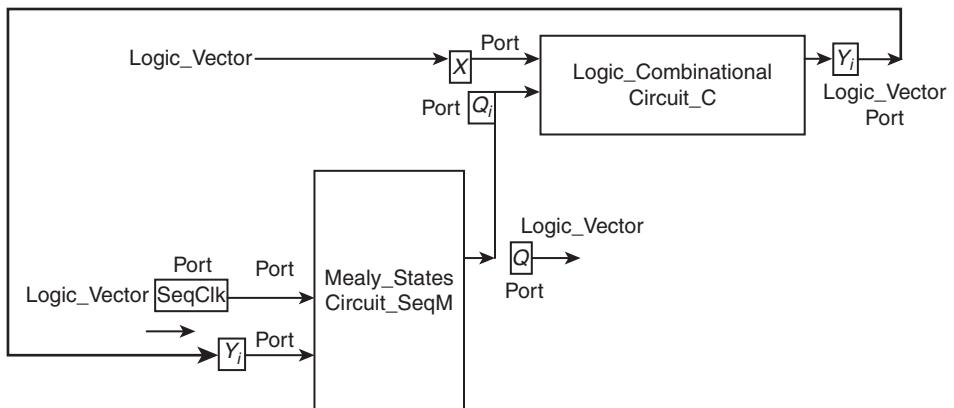


FIGURE 16.5 Mealy sequential circuit consisting of Mealy_StatesCircuit_SeqM and the Output states \underline{Q} are given as inputs \underline{Y} after a Logic_CombinationalCircuit_C generates output \underline{Y} .

Example 16.23 How can you describe MealySeqCircuit entity and components (memory section and combinational circuit).

Solution

The entity can be described follows:

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_textio.all;
use std.textio.all;
  
```

```

Tclk, Tdelay: Time;
Tclk: = 50 ns; -- Interval  $T$  of Clock inputs
Tdelay: = 4 ns;
entity MealySeqCircuit is
  n, m, i, j: Positive;
  ; -- Sequential circuit entity has two components (memory section and
  combinational circuit).
component Mealy_StatesCircuit_SeqM
  Port (
    SeqClk: in Std_Logic_Vector ( $m - 1$  downto 0)::  

    Yi: in Std_Logic_Vector ( $j - 1$  downto 0);  

    Q: out Std_Logic_Vector ( $i - 1$  downto 0));
end component;
component Logic_CombinationalCircuit_C is
  Port (X: in Std_Logic_Vector ( $n - 1$  downto 0);  

    Qi: in Std_Logic_Vector ( $i - 1$  downto 0);  

    Y: out Std_Logic_Vector ( $j - 1$  downto 0));
end component;
end entity;

```

Example 16.24 How can you describe architecture RTLMealySeqC of MealySeqCircuit?

Solution

```

architecture RTLMealySeqM of MealySeqCircuit is
  tSeqC, tSeqplusComCircuit, tComCircuit: Time
  tSeqC: = 20 ns; -- Delay in sequential circuit memory section
  tComCircuit: = 10 ns; -- Delay in combinational logic section in sequential
  circuit
  tSeqplusComCircuit: = tSeqC + tComCircuit; -- Delay of Sequential Circuit
  both sections;
  process (SeqClk, Yi, Q) -- X, SeqClk, Y are inputs and Q is output
    Signal SeqClk: in Std_Logic_Vector ( $m - 1$  downto 0) after tclk ns;
    Signal Yi: in Std_Logic_Vector ( $j - 1$  downto 0) after
    tSeqCplusComCircuit;
    Signal Q: out Std_Logic_Vector ( $i - 1$  downto 0) after tSeqC
  begin
    ; -- process body for Logc_StatesCircuit_SeqC
  end process;
  process (Q, Qi) -- Define the assignments with A and B as inputs to
  generate Y
    begin
      Qi <= Q' Delayed (0); -- Interconnect Qi input to Q output
  process (Qi, X, Y) -- Define the assignments with A and B as inputs to
  generate Y
    Signal X: in Std_Logic_Vector ( $n - 1$  downto 0);
    Signal Qi: in Std_Logic_Vector ( $i - 1$  downto 0);

```

```
Signal Y: out Std_Logic_Vector ( $j - 1$  downto 0) after tComCircuit;
begin
    -- process body for Logic_CombinationalCircuit_C
    end process;
process (Yi, Y) -- Define the assignments with A and B as inputs to generate Y
begin
    Yi := Y' Delayed(0); -- Interconnect Y output to Yi input
end process;
end RTLMealySeqC;
```

■ EXERCISES

1. Describe package declarations and body for seven logic functions using **and**, **nand**, **or**, **xor** four logic gates with delay = 20 ns. Seven logic functions are to find Gray codes for the binary 0000, 0001, 0010, 0011, 0100, 0101, and 0110.
2. Procedure **find_notxor** has Std_Logic_Vector A, B, C, D, \dots has n elements. Procedure has Q and Q_1 out logic in the arguments for the results of (i) XORs of n-logic inputs and (ii) not of XORs. Show how the declarations for the arguments are made.
3. How does a library is made using use functions for using logic as well as numeric standard functions?
4. Open file mode can be write or read or append. How does an attribute used to find the open file mode?
5. A sequential circuit clock gives outputs as follows: (Clock outputs are used inputs in synchronous sequential circuits)

SeqClk: = 0 if RST_Clk = 1 and SeqClk_Start = 0.

SeqClk: = 1 if SET_Clk = 1 and SeqClk_Start = 0

SeqClk is a series of sequences of 1 and 0 and sequence repeats after 50 ns if SeqClk_Start = 1 (RST_Clk can be 0 or 1 or SET_Clk can be 0 or 1.)

Assume clock interval T for one sequence of 1 is 10 ns and 0 for 40 ns each.

SeqClk1 is clock output which is not SeqClk means 180 degree out of phase SeqClk.

- (i) How do you entity SeqClk design unit? How do you architecture RTLSeqClk design unit of SeqClk?
 - (ii) What process is executed at RTLSeqClk?
 - (iii) What will be the waveform of the sequential circuit clock that gives outputs after the process executes?
6. Draw a Moore sequential circuit for the following:
 - (i) **Moore_StatesCircuit_SeqM**
 - (ii) NOT. **Moore_StatesCircuit_M1** (memory section). One input port \underline{Y}_1 is at **Moore_StatesCircuit_M1**. \underline{Y}_1 also interconnects to output port \underline{Y} . $\underline{Y}_1 = \underline{Y}' \text{delayed}(0)$. \underline{Y} is output of not operation with \underline{Q}_1 . **Moore_StatesCircuit_M1** output is at port \underline{Q} . \underline{Q} interconnects the NOT input at port \underline{Q}_i . $\underline{Q}_i = \underline{Q}' \text{delayed}(0)$.

7. Describe Moore sequential circuit entity, its components, and architecture for the above exercise.
8. Describe architecture RTLSeqC of MooreSeqCircuit entity.
9. Draw a Mealy sequential circuit for the following:
 - (i) Mealy_StatesCircuit_SeqM2
 - (ii) Combinational logic. Combinational logic operation is X xor Q_i . Mealy_StatesCircuit_M2 (memory section) one input port is \underline{Y}_1 . \underline{Y}_1 also interconnects to output port \underline{Y} . $\underline{Y}_1 \Leftarrow \underline{Y}'\text{delayed}(0)$. \underline{Y} is output of the combinational logic xor operations of X and \underline{Q}_1 inputs. Mealy_StatesCircuit_M2 output is at port \underline{Q} . \underline{Q} interconnects the combinational logic input at port \underline{Q}_i . $\underline{Q}_i = Q'\text{delayed}(0)$.
10. Describe a MealySeqCircuit entity and components (memory section and xor logic circuit). Describe architecture RTLM MealySeqC of MealySeqCircuit.

■ QUESTIONS

1. What is a package? Give an example of the declarations and body of a package of seven logic functions and, nand, or, xor, not, notxor, and buffer.
2. What do you mean by a package body? Give an example of the body of a package of read and write functions for a file.
3. What are the kinds of subprograms?
4. What is a procedure? How is the procedure declarations and behaviour described in the procedure body?
5. What is a function? How is the function declarations and behaviour described in the function body? How is a return statement described in the function body?
6. How does a procedure differ from process?
7. How does a function differ from process?
8. How is the scalar attributes used?
9. How is the array attributes used?
10. What do you mean by general sequential circuit? Draw a general sequential circuit.
11. How to write RTL model for a sequential clock generating +ve edges each 20 ns?
12. When do you use the term state-machine for a sequential circuit?
13. What do you mean by synchronous sequential circuit?
14. Draw a synchronous sequential circuit in Moore model.
15. Draw a synchronous sequential circuit in Mealy model.
16. What are the differences in the Moore state machine and Mealy state machine? Can a Mealy state machine be converted to a Moore state machine?

This page is intentionally left blank.

CHAPTER 17

VHDL—Test Benches

OBJECTIVE

Previous two Chapters explained how to model entity and architecture for a combinational or sequential circuit. We also learnt how to model a package using library, components, and subprograms, for the circuit. The design units are entities, architectures, packages, and test benches). The units may or may not work as modeled. They need to be tested. We will first learn the statements, vectors, test vectors, data types, and files in the processes and subprograms. We will then study the ways of testing a combinational or sequential circuit. Then we will study how to model the test benches.

Firstly, let us consider why a design should be subjected to tests and why it should be simulated. Consider the following: An object may or may not work as expected. Let us throw a ball up. We do not know how much far will it go up. We may plan that it should not come back, but it returns back. We may plan that it should come back to exactly same place from where it was thrown. This does not happen. This is because we have to consider the effects of gravity of earth on height, wind speed, and direction at the instance of throwing, and deviation from ball exactly vertical throw.

Simulation can be done by visualizing on computer screen, the ball thrown at various speeds under different conditions. We must also visualize by method of comparisons of observed responses with responses stored in data files or tables or lists.

We compare the expectations and observations under various conditions of velocity at which ball was thrown, inclinations from vertical, gravity parameter and its variation with height, and wind conditions to find how far the ball will go up and find how much distance it will return from where it was thrown. Then the ball can actually be thrown and see its actual working.

What will happen if in place of ball a satellite, which is much heavier than a ball, is sent in order to put it in earth's orbit? More complex conditions will be encountered. Simulations under the varying conditions help immensely.

Similar situation occurs when we model a system in VHDL.

17.1 PROCESSES AND SUBPROGRAMS

The processes and subprograms have several statements and expressions. Their uses are described in the following subsections.

17.1.1 Statements

The statements in the processes and subprograms are of two kinds.

1. *Concurrent statements*: These are statements outside the process or subprogram (procedure or function). The statements for the assignment can be in any order. Consider Signal related statements outside a process in architecture.

Signal YA: Std_Logic_vector (n downto 0): = ('0' & A);
; -- YA signal of extended vector of A from $n - 1$ to n elements
; -- extended to n^{th} length. n^{th} element of YA = logic '0'.

Signal YB: Std_Logic_vector (n downto 0): = ('0' & B);
; -- YB signal of extended vector of B from $n - 1$ to n elements
; -- extended to n^{th} length. n^{th} element of YA = logic '0'.

Signal YSum_N: Std_Logic_vector (n downto 0); -- YSum_N signal of extended vector length, n .
; -- Sum_N $n - 1$ to 0 extends to Signal YSum_N

The statements executed are not in any specific order. A set of concurrent statements can execute in any order, that is, YA can be executed next to YB or YSum_N.

A concurrent statement again executes when an event takes place at A or B. YA is again executed if either A or B changes from '1' to '0' or '0' to '1'.

2. *Sequential statements*: These are statements inside a process or subprogram (procedure or function). Behaviour is modeled by the sequential statements in the process or subprogram. Sequential statements are between **begin** and **end**. The statements cannot execute in any order. Statements execute in their sequence. A sequential statement executes in order of its expressions, not on event.

Consider signal Y and following sequential statements in a process body.

Y <= A and B;
Y₁ <= not Y or A; -- then Y₁ executes in sequential order after Y.

-- If event for A takes place, either A becomes ‘0’ from ‘1’, or ‘1’ from ‘0’ then firstly Y executes then the Y_1 . This is because process is sensitive to A .

The process after executing last sequence in statements waits for the next event. The simulation of waveforms is, however, continued and no wait is asserted enforced on the waveforms.

Process or subprogram after executing set of statements in sequence may call a procedure or function. Return statement when executed in the procedure or function then there is return from the called procedure or function also takes place. When return from called function completes, remaining sequential statements of the process or subprogram executes from next sequence.

Point to Remember

A set of *concurrent* statements can execute in any order. A set of *sequential* statements are inside the process or subprogram and cannot be executed in any order. Sequential statements execute in sequence of the statements.

17.1.2 Vectors

The vectors when used in inputs and outputs are called inputs and output vectors. Consider for example, a demultiplexer circuit (DeMux). DeMux has address channel select input vector of m elements, $Addr_0, Addr_1, \dots, Addr_{m-1}$.

The circuit input \underline{A} is a Std_Logic_Vector. It has k elements A_0, A_1, \dots, A_{k-1} . Each $Y(Y_0, Y_1, Y_2, \dots)$ is also a Std_logic_vector of k elements 0 to $k - 1$. Therefore, DeMux has $n = 2^k$ outputs, Y_0, Y_1, \dots, Y_{n-1} , each output is also a Std_logic_vector of k elements 0 to $k - 1$.

All the elements of each \underline{Y} remain unchanged except that of addressed channel. Addressed channel $\underline{Y} \leq \underline{A}$. For example, if address channel input vector is “0000” then $Y_0 = A$, if “0001” then $Y_1 = A$, and if “0010” then $Y_2 = A$, and so on. Following example shows how to model input, outputs and addr vectors of the DeMUX.

Example

How will you model the vectors for the input and output signals of deMux?

Solution

```
Type demux_table is array of (1 to n) of Std_Logic_Vector (1 to k)
Y_OutputVectors: demux_table;
A_InputVector: in Std_Logic_Vector;
Addr_Vector: in Std_logic_Vector;
```

A set of constant vectors can be used to store the constants in a file or table. File may be an American Standard Code for Information Interchange (ASCII) file (file with ASCII characters.) The constants can later be used in the entity or behaviour or structure or subprograms. The constant vectors when used for testing are called test vectors. The test vectors are used for the constant inputs and outputs that are needed for the testing.

Example

How will you declare the `Test_Vectors` for the input and output signals of a multiplexer? The length of each input A_0, A_1, \dots, A_7 is k and n is the length of the A input vector. Assume $k = 2, n = 4$ and `TestInputVectors_0` is a set of i distinct input vectors, which are used for testing. Number of different test input-vectors $i = 8$.

Solution

-- Total number of `Std_Logic` inputs = $k \times n = 16$ at the input of multiplexer. Multiplexer input is a table type data. Input can be considered as a table of n columns (number of inputs) and k rows (number of logic states at each input). Each table cell has one `Std_Logic` input.

```
Type mux_inputTable is array of (0 to  $n - 1$ ) of Std_Logic_Vector (0 to  $k - 1$ )
```

`A_InputVectors`: `mux_input`; -- total number of `Std_Logic` inputs = $k \times n = 8$.

```
Type TestInputVectors_Table is array of (0 to  $i - 1$ ) of A_InputVectors; --  $i$  = Number of test input vectors
```

```
TestInputVectors_0: TestInputVectors_Table := ("10", "00", "00", "11", "01", "11", "11", "00"), ("00", "10", "11", "10"), ("11", "01", "00", "01"), ("10", "11", "01", "11"), ("01", "00", "10", "11"), ("00", "11", "00", "01"), ("11", "00", "11", "10"); -- total number of Std_Logic inputs modeled as the test vectors =  $k \times n \times i = 64$ .
```

-- Total number of `Std_Logic` states in each `TestInputVectors` = $k \times n = 16$ for the test of multiplexer. The `TestInputVectors` of multiplexer is a set of table type input vectors. Input vectors can be considered as a table of n columns and k rows. Each column has one `Std_Logic` input.

Point to Remember

A set of input vectors and output vectors can be used in testing. A set of constant vectors can be used to store the constants in a file or table and read when needed.

17.1.3 Conversions from a Data Type to another

A data type can be converted from one type to another. For example, a set of logic states can be converted to an arithmetic binary number or integer.

A conversion enables arithmetic operations of sum, subtraction, multiplication, and division.

Another example is conversion of an Integer to Time. Following example shows a use of it.

Example

How will a Signal Y_i input is given after a delay in Sequential circuit from the present state of `Std_Logic_Vector` (the state at last change in \underline{Q})? First convert the integer variables, delay and clk to time. Assume that

```
Tclk, tSeqM, tSeqplusComCircuit, tComCircuit: Time;  
-- Tclk: = 50 ns; -- period of clock
```

```
-- Let for tSeqM: = 20 ns; -- Delay in sequential circuit memory section
-- tComCircuit: = 10 ns; -- Delay in combinational logic section in sequential
circuit,
-- tSeqplusComCircuit: = tSeqM + tComCircuit; -- Delay of Sequential Circuit
both section.
```

Solution

We can model the above declarations and statements using the variables as follows:

```
variable delay: Integer; = 10;
variable clk: Integer; = 50;
Tclk: = clk * ns; -- Clock intervals in ns
tSeqM: = delay * 2 * ns; -- Delay in sequential circuit memory section is 20 ns
tComCircuit: = delay * ns; -- Delay in combinational logic section in sequential
circuit is 10 ns
tSeqplusComCircuit: = tSeqM + tComCircuit; -- Delay of Sequential Circuit
both sections 30 ns.
```

Conversion functions examples are as follows:

1. to_String();
2. to_StdLogicVector();
3. to_BitVector();

A function `to_String(Y)` in `textio` package can be used to convert `Std_Logic_Vector` `Y` into a string. For example, it converts ('1', '0', '1', '1', '0', '1', '0', '1') to a String "10110101".

Point to Remember

A data type conversion enables conversion to string, integer, bit_vector, Boolean and time. Conversion to string enables write into text file and later read from the file.

17.1.4 Now and Wait

`Now` is a function already defined in VHDL. It returns current simulation time.

```
If Now > 120 ns then
Wait;
End If;
```

A process may wait for the signals given in the sensitivity list of signals or for a period or until a Boolean expression is `true`. The following are wait statements used in modeling a process.

1. `Wait for Y;` -- Signal `Y` is in sensitivity list
2. `Wait for 8 ns;` -- wait for 8 ns
3. `Wait for A and B and C;` -- wait for `A, B` and `C` are 1s
4. `Wait for Y until A and B and C for Tclk/2 ns;` -- Wait for event on `Y`, until `A, B` and `C` are 1s and for `Tclk/2` ns
`Tclk` is the time interval defined earlier.

Point to Remember

Predefined function `Now` returns current time at the simulator.

`Wait` statement enables suspension of a process or subprogram for event on signal in sensitivity list or for a time period or for a Boolean expression returning true. There can be combined wait for time and Boolean expression returning true and signal event.

17.1.5 Files

VHDL has file input and output capabilities. VHDL can be used as text processing general-purpose language. The files can be used for testing, stimulation or verification of signals. Variables and signals initial and specific instances can be written into a text file (ASCII file). Text file write procedures are in library `textio` standard package. We can write into a file buffer as follows:

```
Write (File_Buffer, String ("T = "));  
Write (File_Buffer, to_String (Now)); -- Write time now  
Write (File_Buffer, String ("D = "));  
Write (File_Buffer, to_String (D_in)); -- write string after conversion of  
D_in Std_Logic_Vector
```

We can write into a file buffer as follows:

```
Read (File_Buffer, String D_in);  
variable words1: String (1 downto 0); -- declare String words 1 of 2  
characters  
variable words2: String (2 downto 0); -- declare String words 1 of 3  
characters  
Signal D_in: Std:Logic_Vector; -- declare D_in Std:Logic_Vector  
Read (File_Buffer, String ("T = ")); -- read String words 1  
Read (File_Buffer, Now); -- read Time now  
Read (File_Buffer, String ("D = ")); -- read String words 2  
Read (File_Buffer, D_in); -- read Std_Logic_Vector D_in after con-  
version of string D_in
```

Point to Remember

File enables write of the variables and signals at various instances. The write into a text file is done using `textio` package write functions. The read from the text file can be done later for use, for example, for use by a simulation model.

17.1.6 Events and Sensitivity List

Event means change in

1. ‘0’ after ‘1’,
2. ‘1’ after ‘0’,

3. +ve edge, and
4. -ve edge

A *sensitive list* is a list of set of signals for which a process is sensitive. Sensitive means signal can change the results of the process. When an event occurs during a process the signals in sensitive list are executed in a sequence of statements. A process executes statements in a sequence not concurrently.

Point to Remember

A process, which is not a specified signal in sensitivity list and no explicit execution of wait statements, continues execution without wait. A process stops execution when it has specified a signal in sensitivity list or has explicit execution of Wait statement. The process starts again on an event of sensitive list signal. The process starts again when wait is over.

17.1.7 Assertion, Report, and Severity Functions

A function **Assert** may be executed after a set of sequential statements. **Report** generates in the last statement or last but one of the sequence. Report returns a string if the Boolean expression after the **Assert** is false. **Severity** returns severity level failure, error, warning, note (in order of decreasing severity (significance)).

Example

Carry should be generated on addition of binary numbers 11 and 11 and sum should be 0. How do you write the **Assert** statements to report error and severity on execution?

Solution

```
Assert (CY_Out = '1') Report "Error in CY_Out" Severity Error;
Assert (Sum_N = "10") Report "Error in Sum_N" Severity Error;
Assert (Sum_N = "10") Report "Error in Sum_N" Severity Error;
```

Point to Remember

An assertion statement executes sequentially if inside the process and concurrently if outside the process. The **Report** generates after the assertion when the Boolean expression after **Assert** is false. Severity returns **Severity** failure, error, warning, note (in order of decreasing severity (significance)) of returned level.

17.1.8 Instantiation During Structural Modeling

Instantiation means creation of a new instance. For example, create a new object from a given object. An entity instantiation is by specifying the ports inputs, outputs and signals. [Port is a signal using which the entity communicates to other models (architecture)].

Components interconnect and form an entity. A component can instantiate the new components on mapping the ports. Advantage of instantiation is that several components can be processed from just one component by just mapping the port inputs and outputs to other components.

For example, an up counter can be instantiated as given below:

`Up_Counter: CounterCircuit port map (Count_Enable, Reset0, SeqClk, Q0)`

17.2 TESTING OF COMBINATIONAL AND SEQUENTIAL CIRCUITS

There are two kinds of circuits (Combinational and Sequential) that need different approaches for testing.

17.2.1 Testing of a Combinational Circuit

A combination circuit may or may not work as expected. Circuit outputs may or may not be as modeled for a given input test vector. When circuit entity and behaviour are modeled, the behaviour of outputs needs to be simulated (visualized) for all possible test input vectors in case of combinational circuits.

We compare the observed and expected output logic vectors. The comparison is from the table or list or file or screen simulations. A set of test input vectors are taken. All possible vectors of the inputs in the set are used for testing. A test vector corresponds to a row of truth table in case of a combinational circuit.

Figure 17.1 shows Combination circuits—`Logic_Combinational Circuit_M` and `Circuit_C`.

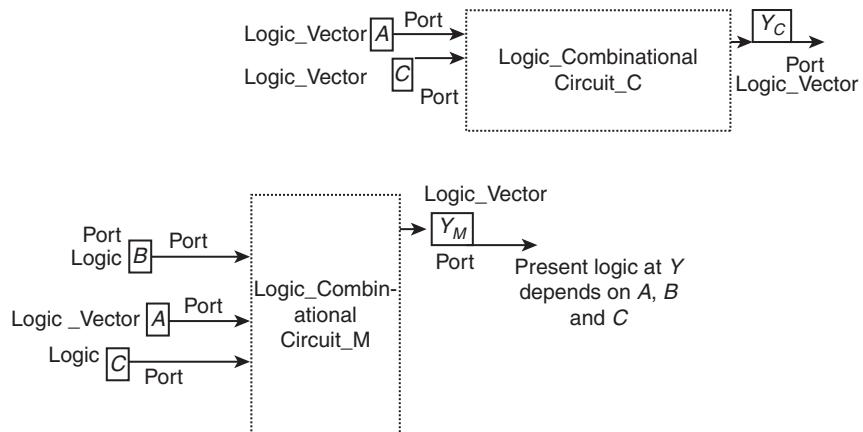


FIGURE 17.1 Two Combination circuits—`Logic_Combinational Circuit_M` and `Circuit_C`.

The followings can be done for the testing of a circuit:

1. Comparing between observed and expected table outputs for all the vectors and logic inputs

2. Comparing between observed and expected outputs in stored file for all the vectors and logic inputs
3. Comparing two lists and then compare the observed and expected output for all the vectors and logic inputs

Consider an example of the entity and behaviour modeled for a XNOR circuit and use of comparisons of the tables during testing

Example

Consider 3-input XNOR circuit. How will you compare with tables for the circuit during the test and find number of errors in output Y for all the input vectors of A , B , C in the circuit?

Solution

Testing of modeled xnor can be done as follows: when there is odd number of 1s then the output is ‘0’, else ‘1’. Table 17.1 gives the observed Y in column 3 for the input logic vectors A , B , C . Table 17.2 gives the expected Y in column 3 for the input logic vectors A , B , C .

One way of testing is that when xnor circuit is tested, the output Y in column 4 is compared with the observed Y at Table 17.1. Table 17.3 compares the Y observed and expected and reports error in the outputs in column 6.

TABLE 17.1 The observed Y in column 3 for the input logic vectors A , B , C

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

TABLE 17.2 Expected Y in column 3 for the input logic vectors A , B , C

A	B	C	$Y = \text{not } A \text{ xor } (B)$ Expected
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

17.10 Digital Systems: Principles and Design

TABLE 17.3 Comparison of Y_{observed} and expected and reports error in outputs

A	B	C	$Y = \text{not } A \text{ xor } (B)$ Expected	Y Observed	Error
0	0	0	1	1	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	1	1	0
1	0	0	0	1	1
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	0	0	0
Counts of Errors				2	

We can write a VHDL model for XNOR testing, which finds that number of times error occurred for all vectors of inputs, Error_Count = 2.

Consider an example of the entity and behaviour modeled for a XNOR circuit and use of comparisons of the tables during testing.

Example

Consider 3-input NAND circuit. How will you test after reading the file for the circuit inputs and outputs during the test and find number of errors in output Y for all the input vectors of A, B, C in the circuit?

Solution

Testing of modeled NAND can be done as follows: when any of the inputs is 0, Y is 1.

A file can save the inputs and outputs as follows:

File for 3 Inputs NAND;

```
NAND Input 3 Output 1;
A '1' B '1' C '1' Y '0';
A '1' B '1' C '0' Y '1';
A '1' B '0' C '1' Y '1';
A '0' B '1' C '1' Y '1';
A '1' B '0' C '0' Y '1';
A '0' B '0' C '1' Y '1';
A '0' B '1' C '0' Y '1';
```

The observed Y on each input vectors is tested with respect to the values read from the file for 3 inputs NAND. The number of errors can be counted.

Consider an example of the entity and behaviour modeled for a XNOR circuit and use of comparisons of the tables during testing.

Example

Consider a XNOR. How will you test using computer screen simulation? Assume that simulator runs for 120 ns. Use test vectors are ("011", "111", and "000"). A test vector changes every 36 ns.

Solution

Figure 17.2 shows the computer screen simulation of inputs A , B and C . ABC test vectors are (“011”, “111”, and “000”) and output Y . Simulator shows that Y is ‘0’ for odd number of 1s in input vector of elements A , B and, $C = 1, 1$, and 1. The entity stands tested and verified for three input vectors.

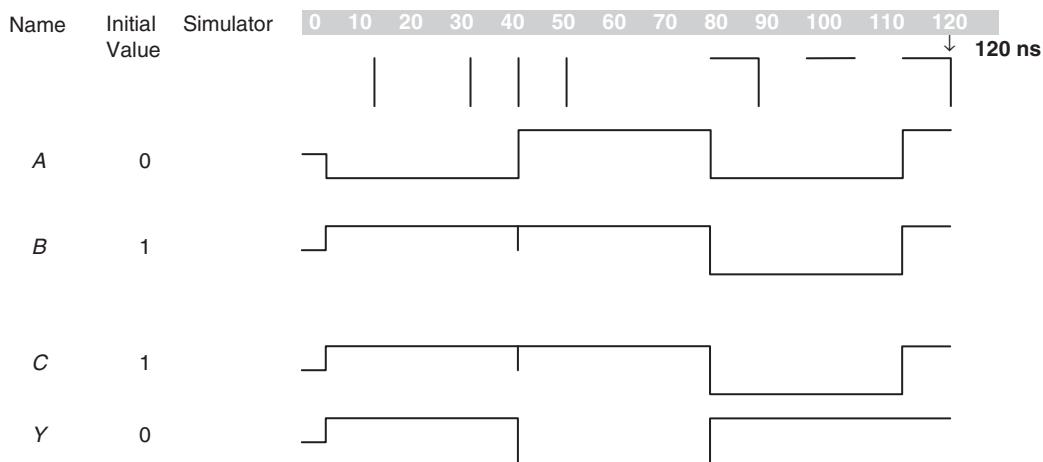


FIGURE 17.2 Computer screen simulation from a simulator for a XNOR circuit.

Point to Remember

Behaviour visualization and counting number of errors can be on comparisons of the responses expected from the available data in files or tables or lists or visualization on computer screen.

17.2.2 Testing of a Sequential Circuit

A sequential circuit may or may not work as expected. Its characteristic is that the present output state is also an input after logic operations and delay. The present state drives the next state and the transition to next state is after a delay. (The transition is after clock event in case of synchronous sequential circuit.)

Figure 17.3 shows synchronous sequential circuit consisting of Logic_Combinational_Circuit_C and memory section Circuit_SeqM. It includes sequencing clock input in case of synchronous sequential circuit. The Circuit_C generates next state after a delay = tSeqM from the SeqClk event (+ve edge or negative edge).

Circuit output states that at the memory section of the circuit may or may not be as modeled. When circuit entity and behaviour of state vectors are modeled, the behaviour should simulate (visualize) the state behaviour after a delay.

Synchronous sequential circuit: All possible input vectors are considered. A vector also includes the inputs based on the present state-sequence at memory

section output. Clock sequences include the input vectors in case of synchronous sequential circuit.

All possible vectors of the inputs are taken into account. The clock levels, clock events ('0' after '1', '1' after '0', +ve edge and -ve edge) and intervals between the clock sequences of 1 and 0 events are taken into account in case of synchronous sequential circuit.

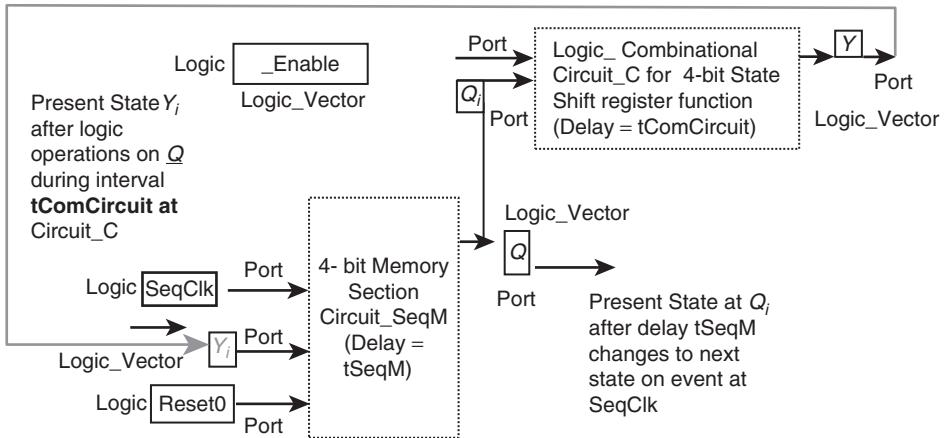


FIGURE 17.3 Sequential circuit consisting of Logic_Combinational Circuit_C and memory section Circuit M plus sequencing clock input in case of synchronous sequential circuit.

The followings can be done for the testing of a sequential circuit:

1. Comparing between observed and expected sequences of states and state transitions from present to next state in state table
 2. Comparing between observed and expected sequences of states and state transitions from present to next state in a file
 3. Comparing two lists for comparison of the observed and expected states
- The comparison is for all possible states, logic input vectors, and clock events.

Example

1. Give an example of functioning of a finite state machine (FSM)—a synchronous sequential circuit.
2. How will you test using computer screen simulation? Assume that simulator runs for 120 ns and -ve edges occur after every 17.5 ns.

Solution

1. Consider an example of a divide-by-two circuit. It is a sequential circuit with two -ve edge triggered JK-FFs in memory section, *Circuit_SeqM*. The section consists of one internal JK-FF0 (flip-flop 0) and another internal JK-FF1 (flip-flop1). J and $K = 1$ s for both. When a clock input -ve edge event takes place at SeqClk, the Q_0 logic vector output toggles. Q_0 is input clock input of other internal JK-FF1 (flip-flop1). The outputs of JK-FF1 are

Q and Q' (not Q). Q and Q' reflect the present_state. The present_state undergoes transition to next state on each successive alternate –ve edges. Circuit_C consists of Enable input, E and Q (present state) generates output $Y = Q$ and E after a delay.

It has tow finite interval states $Q = 1$ from 0 after 2. Tclk and $Q = 0$ from 1 after 2. Tclk. It can be called a finite state machine (FSM).

- Figure 17.4 shows the computer screen simulation of input T and output Q . Simulator shows that there is one transition at Q at $2 \times$ Tclk for every two –ve edges input at the successive events at Tclk. The transition is after a delay from clock even. The entity divide-by-two stands tested and verified for 17.5 ns intervals between the –ve edges at T .

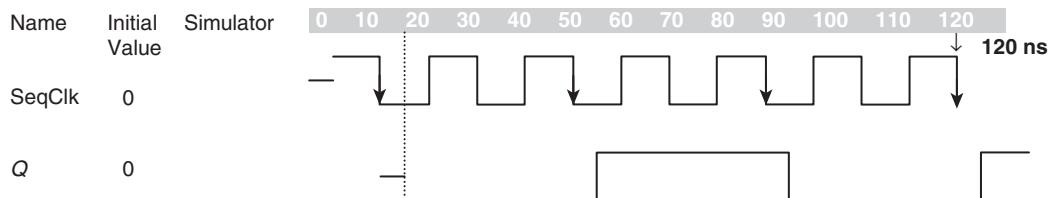


FIGURE 17.4 Computer screen simulation from a simulator for a divide by two circuit, Q changes after 2 Tclk and after a delay from even –ve edge.

17.3 TEST BENCHES

An important question is that how the circuit will function when synthesized. Will the behaviour be same as expected for the circuit?

A test bench is a collection of descriptions called models in VHDL. It verifies the design functioning. It uses files as input. It is used for testing, counting the errors, verification, and simulation. It consists of a set of stimulus and simulator. Set of test benches is a part of system design units. A circuit for the system is synthesized after the successful tests, verifications, and simulations.

Test bench enables the followings for the testing of a circuit:

- Comparing using a table, observed and expected outputs for all the vectors and logic inputs
- Comparing using a stored file, observed and expected for all the vectors and logic inputs
- Comparing using lists and compares the observed and expected for all the vectors and logic inputs
- Simulation of inputs and outputs

Once the testing, verification, and simulation results show the correctness of the design units, gate-level schematics and synthesis of a circuit are generated automatically. A simulator provides for interaction with the user.

A test bench has models (descriptions) to generate test input vectors using stimulus. Figure 17.5 shows the steps for testing using a test bench.

17.14 Digital Systems: Principles and Design

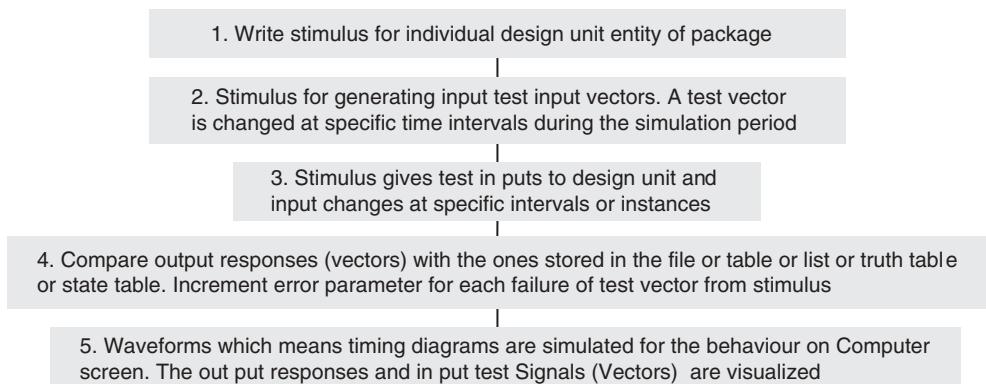


FIGURE 17.5 Steps for testing using a test bench.

The following are the actions during testing:

- Stimulus provides input test input vector, which changes at specific time intervals during the simulation period.
- Stimulus provides test inputs to the entity, which is to be tested.
- The output responses (vectors) are compared with the ones stored in the file or table or lists or truth table or state table.
- Error parameter increments for each failure of test vector inputs using stimulus.
- Stimulus generates test inputs to the entity, which is to be tested at successive intervals to enable visualizing timing diagrams.
- Stimulus generates states as inputs in case of synchronous sequential circuits.
- Stimulus includes the generation of clock events in case of synchronous sequential circuits.

Test bench enables the followings for the testing of a sequential circuit:

- Comparing using a table, the observed State transitions with the expected for all the state vector inputs and logic inputs (Input includes states and events at the sequencing clock in case of synchronous sequential circuit)
- Comparing using a stored file, observed and expected States for all the state vector inputs and logic inputs
- Comparing using lists and compares the observed and expected States for all the state vectors and logic inputs
- Finds the delay in state transition intervals
- Simulation of inputs and outputs

Figure 17.6 shows the steps for modeling the test benches.

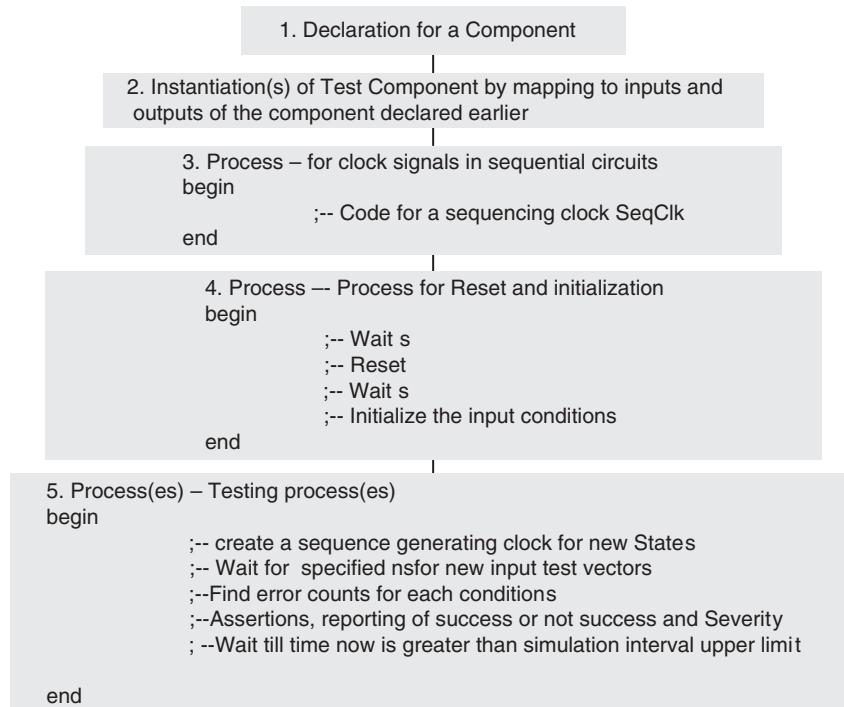


FIGURE 17.6 Steps for modeling a test bench for test component architecture.

Point to Remember

A stimulus is modeled for test entity. The stimulus includes input test vectors, which give Signals to the entity. A test input vector changes at specific time intervals. Each time a new vector is input during the period of simulation. The waveforms (timing diagrams) are simulated for the behaviour of output responses and compare with the inputs. Test benches consist of a set of stimulus and simulator.

■ EXAMPLES

Example 17.1

1. How will you declare concurrent statements for Signals in an adder?
2. When and where are the concurrent statements used?

Solution

1. The concurrent statements of Signals in an adder are as follows:
Signal YA: Std_Logic_vector (*n* downto 0): = ('0' & A);

```

; -- YA signal of extended vector of A from  $n - 1$  to  $n$  elements
; -- extended to  $n^{\text{th}}$  length.  $n^{\text{th}}$  element of YA = logic 0.
Signal YB: Std_Logic_vector ( $n$  downto 0) := ('0' & B);
; -- YB signal of extended vector of B from  $n - 1$  to  $n$  elements
; -- extended to  $n^{\text{th}}$  length.  $n^{\text{th}}$  element of YA = logic 0.
Signal YSum_N: Std_Logic_vector ( $n$  downto 0); -- YSum_N signal of
extended vector length,  $n$ .
; -- Sum_N  $n - 1$  to 0 extends to Signal Y Sum_N
2. The statements are used when they need not execute in any specific order.
 $YA$  can be executed next to  $YB$  or  $YSum_N$ . A concurrent statement again
executes when an event takes place at  $A$  or  $B$ . YA is again executed if either
A or B changes. The concurrent statements are used outside the process of
addition.

```

Example 17.2 How will you declare concurrent statements for Signals in a set of D-FFs?

Solution

The concurrent statements of Signals in an adder are as follows:

```

File Signal_Dump: Text open Write_Mode is "/home/D_FF/test0.
signal_dump";
-- a text file is declared and opened in write mode
generic (n : Natural := 1); -- declare constant n = 1, a natural number
; -- n is the number of D inputs
Signal D_in, SeqClk: in Std_Logic_Vector ( $n - 1$  down to 0); -- One D-FF in an array of D-FFs
Signal Q, Q1: out Std_Logic_Vector ( $n - 1$  down to 0);
SeqClk: in Std_Logic;

```

The concurrent statements are used outside the process for testing the set of *D-FFs*.

Example 17.3 How will a test bench architecture that you use Assert statement in case an entity modeled for the Boolean expression $\underline{Y} = \underline{A}$ and not $\underline{B} + \underline{A}$ and not \underline{B} does not result in $\underline{Y} = 0$ for each element position for even numbers of 1s equal to zero?

Solution

Boolean expression $\underline{Y} = (\underline{A} \text{ and not } \underline{B}) + (\text{not } \underline{A} \text{ and } \underline{B})$ does not result in $\underline{Y} = 0$ for each element position for even numbers of 1s equal to zero.

```

Signal Y: out Std_Logic_Vector ( $n - 1$  downto 0);
Signal A, B: in Std_Logic_Vector ( $n - 1$  downto 0);
B <= not B
Y <= A and not B + (not A and not B);
A <= "00011000";
B <= "0001100"
Wait for 100 ns;
A <= ,

```

```
If( $Y = 0$ ) then error_count = error_count + 1;
end If;
Assert ( $Y = 0$ ) Report “Error XOR operation by (A and not B) + (not A and B)
not successful” Severity Error;
-- End of the Test
```

Example 17.4 How will you use sequential statements for Signals in an encoder for ASCII code generation?

Solution

Process

```
begin
Select_en <= "0"; -- Assign encoder select = '0'
A <= "00000000"; -- Assign A = 0
Wait for 8 ns;
Select_en <= "1"; -- Assign encoder select = '1'
Wait for 1 ns;
If ( $Y \neq "00110000"$ ) then error_count = error_count + 1;
end If;
Assert ( $Y = "00110000"$ ) Report “ $Y$  is not ASCII code of 0 when input
= 0” Severity Error;
-----
Select_en <= "0"; -- Assign encoder select = '0'
A <= "00000001"; -- Assign A = 1
Wait for 8 ns;
Select_en <= "1"; -- Assign encoder select = '1'
Wait for 1 ns;
If ( $Y \neq "00110001"$ ) then error_count = error_count + 1;
end If;
Assert ( $Y = "00110001"$ ) Report “ $Y$  is not ASCII code of 1 when input
= 1” Severity Error;
```

The sequential statements are used inside the process for testing the encoder for ASCII codes.

Example 17.5 How will you use sequential statements of Signals in a *D*-FF? When and where are the sequential statements used?

Solution

Process

```
variable error_count: Integer := 0;
begin
D_in <= 1; -- Wait for 8 ns, wait for period 1 interval = Tclk
If ( $Q \neq '1'$ ) then error_count := error_count + 1; -- error Case 1 If  $Q$ 
differs
```

```
                                ;-- and does not become '1'
end If;
Assert ( $Q = '1'$ ) Report " $Q$  changed and does not show '1' after Tclk"
Severity Error;
; -----
D_in <= 0; Wait 10 ns; -- wait for period 1.25 interval = Tclk
If( $Q = '0'$ ) then error_count := error_count + 1; -- error Case 2 If  $Q$  changed
                                ; -- and does not remain '0'
end If;
Assert ( $Q = '0'$ ) Report " $Q$  changed and not found '0' after 1.25 Tclk"
Severity Error;
; -----
```

The sequential assignments used inside the process. They are used to follow a sequence for processing.

Example 17.6

A demultiplexer circuit has outputs, Y_0, Y_1, \dots, Y_{m-1} and each output Y is a Std_Logic_Vector. The circuit has address_channel select input vector $Addr_0, Addr_1, \dots, Addr_{k-1}$. The circuit has inputs, A_0, A_1, \dots, A_{m-1} . The $m = 2^k$. [$k = 2$ when $m = 4$, $k = 3$ when $m = 8$ and $k = 4$ when $m = 16$.]

1. How will you model the vectors for the input and output signals?
2. How will you model constant text vectors for m to n demultiplexer with each addressed channel having m inputs?
3. How will you model constant text vectors for 1 to 2 demultiplexer with each addressed channel having four inputs? Assume that the three test vectors are used by a test bench.

Solution

1. Modeling of input and output vectors for the Signals are as follows:

Variable m, n, k : Positive

Type demux_Table is array of ($n - 1$ down to 0) of Std_Logic_Vector
($k - 1$ downto 0)

Y_OutputVectors: out demux_Table; -- total number of Std_Logic outputs = $k \times n = 8$.

A_InputVector: in Std_Logic_Vector ($k - 1$ down to 0);

AddrInputVector: in Std_Logic_Vector ($m - 1$ down to 0)

2. Test vector can be made by concatenation of A_InputVector and AddrInputVector

SignalX_InputVector: in Std_Logic_vector := AddrInputVector & A_InputVector;

-- Elements of Addr_InputVector extended to right by postfixing elements of A_InputVector

-- to declare a Testvector

Variable j: Positive;

j := $k + m$;

```

Type TestInputVectors_Table is array of(0 to  $i - 1$ ) of X_InputVector
( $j - 1$  down to 0);
--  $i$  = Number of test input vectors;  $j$  = number of elements in test input
vector
TestInputVectors_0: TestInputVectors_Table;
-- total number of Std_Logic inputs modeled as test vectors =  $j \times i$ .
3. 1 to 2 demux will have test input vectors as follow:
m:= 1;
k:= 4;
n:= 2;
i:= 3;
TestInputVectors_0:= ("00000", "11111"), ("01111", "10000"),
("01010", "10101");
-- Each test Vector has one address input element and four input A elements,
and there are three test vectors
-- total number of Std_Logic inputs modeled as the test vectors =  $(k + m) \times$ 
 $n \times i = 30$ .

```

Example 17.7

A register circuit has inputs, D_0, D_1, \dots, D_{n-1} and inputs Load_In and SeqClk logic. The circuit has outputs, Q_0, Q_1, \dots, Q_{n-1} . How will you model the test vectors of constant input vectors?

Solution

Modeling of input and output vectors for the Signals are as follows:

```

Variable m, n, k:Positive ( $n - 1$  down to 0)
Signal D_in: in Std_Logic_Vector ( $n - 1$  down to 0);
Signal Q, Q1: in Std_Logic_Vector ( $n - 1$  down to 0);
Signal SeqClk: in Std_Logic;
Signal D_in: in Std_Logic;
Signal Input_Vector: in Std_Logic_Vector ( $n$  down to 0): = Load_in &
D_in;
-- Input vector is concatenation of Signals Load_in and D_in
Type DFFs_Table is array of ( $i - 1$  down to 0) of Std_Logic_Vector ( $n$ 
downto 0)
D_TestInVectors: in DFFs_Table; -- total number of Std_Logic
inputs =  $(1 + n) \times i$ .

```

Example 17.8

How do you use read function of textio? Show it by example of read of D and Q input and output of D -FF.

Solution

```

Read (File_Buffer, StringD_in); -- read into file buffer string for D_in
D_in <= to_StdLogicVector (StringD_in); -- D_in Std_Logic_Vector is
assigned
--from the string read for D_in from a text file

```

17.20 Digital Systems: Principles and Design

Read (File_Buffer, StringQ); -- read into file buffer string for D_{in}
Q <= to_StdLogicVector (StringQ); -- Q Std_Logic_Vector is assigned
--from the string read for D_{in} from a text file

Example 17.9

How can the signals of a D flip-flop written into a file so that for each set of D and clock inputs the output Q and Q_1 are also written?

Solution

```
library STD; -- standard library library std;
use std.standard.all; -- needed for bootstrap mode all means all
; --packages in that.
use STD.textio; -- text in-out operations and functions
Write (File_Bufffer, "SeqClk =")
Write (File_Buffer, to_String(SeqClk)); -- write into file buffer string for
Sequential clock state
Write (File_Bufffer, ", D =")
Write (File_Buffer, to_String(D_in)); -- write into file buffer string for  $D_{in}$ 
Write (File_Bufffer, ", Q =")
Write (File_Buffer, to_String(Q)); -- write into file buffer string for  $Q$ 
Write (File_Bufffer, ", Q_1 =")
Write (File_Buffer, to_String(Q1)); -- write into file buffer string for  $Q_1$ 
```

Example 17.10

How are vectors of bits, written into in line0, a vector of integers is written into in line1, a vector of Booleans in line3 when library textio standard package is used? Append two more lines.

Solution

```
library STD; -- standard library library std;
use std.standard.all; -- needed for bootstrap mode all means all
; -- packages in that.
use STD.textio; -- text in-out operations and functions
variable address_bits: Bit_Vector (n - 1 downto 0);
variable Line0: Line; -- Line0 is variable of data type of Line
Write (Line0, address_bits);
variable counts: Integer (m - 1 downto 0);
variable Line1: Line; -- Line1 is variable of data type of Line
Write (Line1, counts);
variable Testexpressions_k: Boolean (k - 1 downto 0);
variable Line2: Line; -- Line3 is variable of data type of Line
Write (Line2, Testexpressions_k);
variable data_bit: Bit;
variable Line3: Line; -- Line3 is variable of data type of Line
Write (Line3, data_bit);
variable clk: Integer;
variable Line4: Line; -- Line4 is variable of data type of Line
Write (Line4, clk);
```

```

variable Testexpression_new: Boolean;
variable Line5: Line; -- Line5 is variable of data type of Line
Write (Line5, Testexpression_new);

```

Example 17.11

How do you model statements in a process and outside the process to write a text file? The file dumps a set of signals into a file for a *D*-FF in a set of *D*-FFs till 120 ns.

Solution

The declarations outside the process can be modeled as follows:

```

File Signal_Dump: Text open Write_Mode is "/home/D_FF/test0.signal_dump"; -- a text file is declared,
                           --opened in write mode and
generic (n:Natural:=1) ; -- declare constant n = 1, a natural number
                           ; -- n is the number of D inputs
Signal D_in, SeqClk: in Std_Logic_Vector(n-1 down to 0); -- One
D-FF in array of D-FFs
Signal Q, Q1: out Std_Logic_Logic_Vector(n-1 down to 0));
SeqClk: in Std_Logic;

```

The write statements in the process can be as follows:

```

variable File_Buffer:Line; -- variable for File buffer before dumping
lines into the file
begin
  Write (File_Buffer, String ("D_FF Dump on +ve Edge Events after
each Tclk"));
  -- write heading line
  If Now < 120 ns then
    Wait on SeqClk = '1' and SeqClk'event;-- Each +ve edge sequencing
    clock
    --event the following sequence repeats
    Write (File_Buffer, String ("T= "));
    Write (File_Buffer, String (Now));
    Write (File_Buffer, String (", D= "));
    Write (File_Buffer, to_String (D_in));-- write string after conver-
    sion of D_in Std_Logic_Vector
    Write (File_Buffer, String (", Q= "));
    Write (File_Buffer, to_String (Q)); -- write string after conversion
    of Q Std_Logic_Vector
    Write (File_Buffer, String (", Q1= "));
    Write (File_Buffer, to_String (Q1)); -- write string after conversion
    of Q Std_Logic_Vector
    Writeline (Signal_Dump, File_Buffer); -- Write lines in File
    buffer to file signal dump.
  Elseif
    Wait;
  End If;

```

17.22 Digital Systems: Principles and Design

Text file read procedures are in library textio standard package. Following is the example of using read functions for D and Q input and output.

Example 17.12

How will Twobit_Add component instantiates TestBTwo_ADDER for a test bench on port mapping?

Solution

```
component Twobit_Add is
    ; -- Remember n = 2 in entity TestBTwo_Adder
    port (A, B: in Std_Logic_Vector (n - 1 downto 0);
          Sum_N: out Std_Logic_Vector (n - 1 downto 0),
          CY_Out: out Std_Logic);
end component Twobit_Add;
Signal A, B: Std_Logic_Vector (n-1 downto 0);
Signal Sum_N: Std_Logic_Vector (n-1 downto 0);
Signal CY_Out: Std_Logic;
begin
    TestBTwo_ADDER: Twobit_Add port map (A, B, CY_Out,
    Sum_N)
```

Example 17.13

How will D_FF01 component instantiates Test_DFF for a test bench on port mapping?

Solution

```
entity TestBD_FF is
end TestBD_FF; -- Declare an entity for the test bench for flip-flop entity
architecture RTLTestBD_FF of TestBD_FF is
Signal D_in, SeqClk: in Std_Logic;
Signal Q, Q1: out Std_Logic;
Component D_FF01 is; -- D_FF component
port (D_in, SeqClk: in Std_Logic;
      Q, Q1: out Std_Logic);
end component D_FF01;
begin
    Test_DFF: port map (D_in, SeqClk, Q, Q1);
```

Example 17.14

How will you use Assert function in case of an adder and report errors for $A = 11$ and $B = 11$ and $A = 00$ and $B = 00$?

Solution

Following are the statements for Assert functions, which are used to report error for $A = 11$ and $B = 11$ and $A = 00$ and $B = 00$.

```
A <= "11"; B <= "11"; -- Test for first case A and B both 11
Wait for 8 ns; -- Assume Adder entity gate delays < 8 ns
If (CY_Out /= '1' or Sum_N /= "10") then
```

```

        error_count:= error_count + 1; -- Increment counts of error
    end If;
Assert (CY_Out = '1') Report "Error in CY_Out" Severity Error;
Assert (Sum_N = "10") Report "Error in Sum_N" Severity Error;
;-----
    A <= "00"; B <= "00"; -- Test for second case A and B both 00
    Wait for 8 ns; -- Assume Adder entity gate delays < 8 ns
    If (CY_Out/= '0' or Sum_N/= "00") then
        error_count:= error_count +1; -- Increment counts of error
    end If;
Assert (CY_Out = '0') Report "Error in CY_Out" Severity Error;
Assert (Sum_N = "00") Report "Error in Sum_N" Severity Error;

```

Example 17.15

How will you use Assert functions? Show it by example in case of *D*-FF.

Solution

```

D_in <= 0; Wait 2 ns; -- wait for period 0.25 interval = Tclk
If (Q/= '1') then error_count:= error_count + 1; -- error Case 4 If Q found
changed
                                                ;even before the next +ve edge
end If;
Assert (Q = '1') Report "Q found changed even before next +ve edge"
Severity Error;
;-----
D_in <= 1; Wait 22 ns;-- wait for period 2.75 intervals of Tclk
If (Q /= '1') then error_count:= error_count + 1; -- error Case 5 If Q changed
                                                ;--and does not remain '1'
end If;
Assert (Q = '1') Report "Q changed and does not show '1' after 2.75.Tclk"
Severity Error;
;-----

```

Example 17.16

How will a four-bit counter to count 0–15 behaviour be modeled in a test bench by instantiation?

Solution

Consider architecture of a Fourbit_counter. The counter component is used to instantiate the other structural component for test bench of four 16-bit counter. A component CounterCircuit has Count_Enable, Reset0, SeqClk at the inputs. It has Q, which is a Std_Logic_Vector in the output.

A counter output is from flip-flop output in states of Q_0, Q_1, \dots, Q_{m-1} . The counter output Q can be modeled as Std_Logic_Vector ($m - 1$ downto 0). CounterCircuit is a component for a general n-bit counter with m outputs for Q . A 4-bit counter has output $Q_0Q_1Q_2Q_3$ as count output. Structure modeling for an NFourbit_Counter is as follows in the architecture:

```
architecture Fourbit_Counter is
variable m: Positive := 4;
Component CounterCircuit is
    port (Count_Enable, Reset0, SeqClk: in Std_Logic;
          Q: out Std_Logic_Vector (m - 1 downto 0), );
    end component CounterCircuit;
Signal Count_Enable, Reset0, SeqClk: Std_Logic;
Signal Q_0, Q_0Q_1, Q_0Q_1Q_2Q_3: Std_Logic_Vector (m - 1 downto 0);
begin; -- architecture body begins
    TestBFourBit_Counter: CounterCircuit port map (Count_Enable,
                                                    Reset0, SeqClk, Q_0Q_1)
Process
    ;
    ;
```

Example 17.17

A test bench generates stimulus for the simulation of the behaviour of an entity and show on screen or otherwise the situations and actions, which will take place in the entity on synthesis from the logic gates. Assume an FSMSL for state transitions between S_0 , S_1 , S_2 and S_4 states. How will you write a test bench for a machine?

Solution

Assume $S_0 = S_0(1 \text{ to } k - 1)$ & '0' operation takes place for each state. Firstly following is tested.

1. When $\text{Reset0} = 1$ then $Q = ('0', '0', '0', '0')$ after 2 Tclk where Tclk is sequencing clock period.
2. When $\text{Reset0} = 0$ then Q remains same as previous state at Q .
3. When $\text{Reset0} = 0$ then $Q = ('0', '0', '0', '0')$ after 2 Tclk, where Tclk is sequencing clock period.
4. Then $\text{Reset0} \leq 0$ and the following sequential actions are tested.
5. When SeqClock +ve edge event occurs then FSMSL memory section of FFs state transfers to Q . Q becomes $Q(1 \text{ to } k - 1) \& '0'$.
6. When SeqClock +ve edge event occurs then FSMSL memory section of FFs state transfers to Q . Q becomes $Q(1 \text{ to } k - 1) \& '0'$.
7. When SeqClock +ve edge event occurs then FSMSL memory section of FFs state transfers to Q . Q becomes $Q(1 \text{ to } k - 1) \& '0'$
8. When SeqClock +ve edge event occurs then FSMSL memory section of FFs state transfers to Q . Q becomes $Q - Q$, which means all $Qs = '0'$.

State machine keep generating next states on each sequencing clock event. The stimulus of a test bench sends the signals for all possible situations and counts the number of errors occurred. If the stimulus report error counts = 0, then the entity behaviour stands tested.

Figure 17.7 shows the functions of test benches for an FSM's architecture.

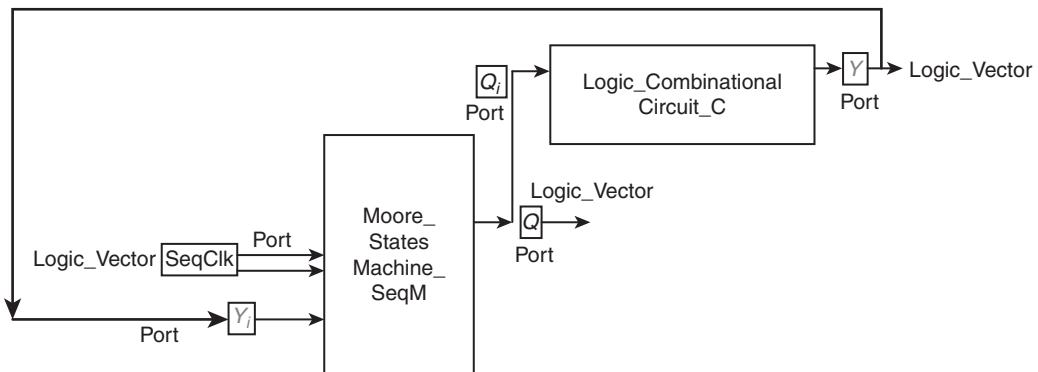


FIGURE 17.7 Functions of test benches for an FSM's architecture.

■ EXERCISES

1. How will you use concurrent assignments of Signals? When and where are the concurrent assignments used?
2. How will you use sequential statements for Signals? When and where are the sequential assignments used?
3. How will you use sensitivity list?
4. What are the statements in Wait function is used?
5. What do you mean by a test bench? Why is it essential to model test benches for a system design?
6. How are the error reported by a test bench? How does severity level reported on Assert?
7. How will you write a test bench for a Shift register as a Moore sequential machine as given in Figure 17.8?

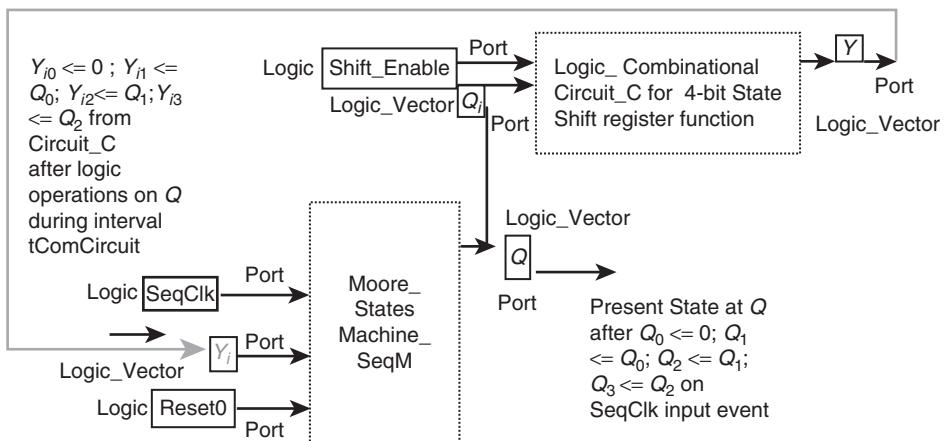


FIGURE 17.8 FSM sequential circuit for shift register consisting of **FSM_StateSLSeqM** and the Output states Q are given as inputs Y_i after a **Logic_SLFunctionlCircuit_S** generates output Y for next sequence of the state.

■ QUESTIONS

1. What do you mean by concurrent statements and sequential statement?
2. What do you mean by test vectors? How can you write sixteen test vectors for all possible combinations in a combinational circuit of four variables?
3. How can you instantiate a component for test bench?
4. What are the functions of test benches? How does a stimulus function?
5. What do you mean by simulation?
6. What are the required timing diagrams for simulation of an FSM with four states?
7. Describes steps for writing test benches.
8. What are the conditions tested in a BCD encoder?
9. How will you write a test bench for multiplier?
10. How will you write a test bench for the circuit given in Figure 17.9?

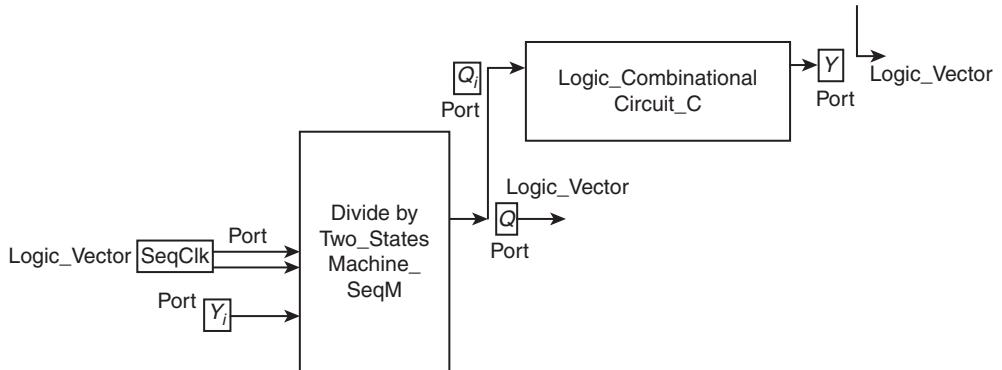


FIGURE 17.9 A Moore machine sequential circuit consisting of Divide by two Moore_StatesMachine_SeqM and Logic_CombinationalCircuit_C generating states Q at sequences of the clock inputs.

CHAPTER 18

VHDL—Examples of Modeling of Adder, Counter, Flip-Flop, Finite State Machine, Multiplexer, and Demultiplexer

OBJECTIVE

We learnt the followings in the earlier chapters: the uses of data types and operators, how to model entity and architecture using Register Transfer Level (RTL) with the help of examples of simple logic gates and combinational circuits.

The design units (entities, architectures, packages, and test benches) are required to be modeled for gate level synthesis of a circuit. We will study the examples of modeling of the adder using arithmetic addition and concatenation operators, and modeling of the counter, flip-flop, finite state machine, multiplexer, and demultiplexer. We shall also learn how to write test benches for these entities and components.

18.1 ADDER

Assume A and B are logic vectors. A is an array with n elements 0 to $n - 1$. Each element is in logic state 0 or 1. B is another array with n elements 0 to $n - 1$. Each element is in 0 or 1 logic state. Sum_N is another array with n elements from $n - 1$ downto 0 and logic of each element is 0 or 1.

A and B are two input Std_Logic_Vectors then Sum_N is output Std_Logic_Vectors after the addition. Carry out is the standard logic output.

18.1.1 Adder Circuit

An addition operator does the arithmetic addition of two bits of each element 0th, 1st, 2ndup to $(n - 1)^{th}$ in the input vectors A and B. A and B are Std_Logic_Vectors.

18.2 Digital Systems: Principles and Design

The addition result is sum, Sum_N, and Carry, CY_Out. A concatenation operator ‘&’ extends the length of the vector, for example, from n to $n + 1$. Elements then are n^{th} downto 0 in place of $(n - 1)^{\text{th}}$ downto 0 earlier.

RTL entity NbitAdder can be described as follows in VHDL. We use arithmetic addition, ‘+’ operator, and concatenation operators. We can describe RTL design codes for an entity Nbit_Adder.

Example

Nbit_Adder is an n -bit adder circuit which adds n -bits with n -bits, and generates a carry in the output. How will you model RTL design codes for the entity Nbit_Adder and architecture? Use behavioural model.

Solution

```
'Library IEEE; -- VHDL using IEEE standard libraries
use IEEE . Std_Logic_1164.all ; -- import Std_Logic from the IEEE Library
use IEEE . Std_Logic_unsigned.all ; -- Std_Logic functions from the IEEE Library
                                ; -- for the unsigned arithmetic operations
use IEEE . Std_Logic_arith.all ; -- Std_Logic from the IEEE Library
                                ; -- Std_Logic_arith functions enable use of
                                ; -- addition operator +, & and other arithmetic
                                operators
generic (n : Natural); -- declare constant n, a natural number
                        ; -- n is the number of bits added by arithmetic adder
entity Nbit_Adder is
port (A, B: in Std_Logic_Vector (n - 1 downto 0);
      Sum_N: out Std_Logic_Vector (n - 1 downto 0),
      CY_Out: out Std_Logic);
end Nbit_Adder;
architecture RTLBehaviourNbit_Adder of Nbit_Adder is
Signal YA: Std_Logic_Vector (n downto 0) := ('0' & A);
                        ; -- YA signal of extended vector of A from
                        n - 1 to n elements
                        ; -- extended to  $n^{\text{th}}$  length.  $n^{\text{th}}$  element of YA
                        = logic 0.
Signal YB: Std_Logic_Vector (n downto 0) := ('0' & B);
                        ; -- YB signal of extended vector of B from
                        n - 1 to n elements
                        ; -- extended to  $n^{\text{th}}$  length.  $n^{\text{th}}$  element of YA
                        = logic 0.
Signal YSum_N: Std_Logic_Vector (n downto 0); -- YSum_N signal of extended
vector length, n.
                        ; -- Sum_N n - 1 to 0 extends to Signal
YSum_N
begin
  YSum_N <= A + B; -- Addition of  $n + 1$  length YA vector and  $n + 1$  length YB
vector.
```

```

Sum_N <= YSum_N (n - 1 downto 0); -- only n elements, n - 1 downto 0 are
                                     ; -- the result of addition, sum_N
CY_Out <= YSum_N (n); -- nth element is the carry result after the sum
end RTLBehaviourNbit_Adder;

```

Use of Concatenation operator, & and addition operator + can be understood as follows:

Concatenation operator, &

Assume Signal A elements from $n - 1$ downto 0 are ‘0’, ‘1’, ‘0’, ‘0’, ‘0’, ‘1’, ‘0’, ‘0’ for the binary number = 01000100 (decimal number 68). $n = 8$. Assume that concatenation operator, & is used for Signal YA as follows:

Signal YA: in Std_Logic_Vector (n downto 0): = (‘0’ & A); -- YA nine elements in the input are ‘0’, ‘0’, ‘1’, ‘0’, ‘0’, ‘0’, ‘1’, ‘0’, ‘0’ in order (n downto 0) after the concatenation operation, (‘0’ & A).

Assume operation (‘0’ & B) results in the YB nine elements after the operation. Assume that the nine elements are ‘0’, ‘0’, ‘1’, ‘0’, ‘0’, ‘0’, ‘1’, ‘1’, ‘1’. Then $n - 1$ downto 0 binary number is 01000111 and decimal number is 71.

Signal YB: in Std_Logic_Vector (n downto 0): = (‘0’ & B);

Addition operator, +

Addition YA + YB gives (‘0’, ‘0’, ‘1’, ‘0’, ‘0’, ‘1’, ‘0’, ‘0’) + (‘0’, ‘0’, ‘1’, ‘0’, ‘0’, ‘1’, ‘1’, ‘1’, ‘1’) = (‘0’, ‘1’, ‘0’, ‘0’, ‘0’, ‘1’, ‘0’, ‘1’, ‘1’). [0 0100 0100 + 0 0100 0111 = 01001011, carry at 9th element is 0 and 8-bit sum is 10001011]. Answer is 139 decimal and CY_Out = 0.

18.1.2 Test Bench for the Adder

Test bench generates stimulus for the simulation of the behaviour of an entity and show on screen or otherwise the situations and actions, which will take place in the entity on synthesis from the logic gates.

When stimulus sets Signals A = 00 and B = 01 to a two-bit adder. If Sum_N shows 01 and CY_Out shows 0, the adder is tested for one set of vectors, 00 and 01. The error count parameter do not increase.

When stimulus sets Signals A = 11 and B = 01 to the two-bit adder. If Sum_N shows 00 and CY_Out shows 1, the adder is tested for one set of vectors, 11 and 01.

The stimulus of a test bench sends the signals for all possible combinations and counts the number of errors occurred. If the stimulus report error counts = 0, then the entity behaviour stands tested.

Example

How can you describe RTL design codes for a *test bench* for entity Nbit_Adder and architecture in behavioural model? Select constant $n = 2$. Test eight cases for the two-bit adder

Solution

Length of A is 2, B is 2, and YA is 3. Length of Sum_N is 2 and CY_Out is logic ‘0’ or ‘1’.

Library IEEE; -- VHDL using IEEE standard libraries

use IEEE . Std_Logic_1164 . all ; -- import Std_Logic from the IEEE Library

```

use IEEE . Std_Logic_unsigned.all ; -- Std_Logic functions from the IEEE Library
                                ; -- for the unsigned arithmetic operations
use IEEE . Std_Logic_arith.all ; -- Std_Logic from the IEEE Library
                                ; -- Std_Logic_arith functions enable use of
                                ; addition operator +, & and other arithmetic
                                ; operators
generic (n:Natural)
entity TestBTwobit_Adder is
generic (n:Natural:=2); -- declare constant n = 2 , a natural number
                        ; -- n is the number of bits added by arithmetic adder
end TestBTwobit_Adder; -- Declare an entity for the test bench for n-bit
                        adder entity
architecture RTLTestBTwobit_Add of TestBTwobit_Adder is
component Twobit_Add is
                        ; -- Remember n = 2 in entity TestBTwobit_Adder
port (A, B: in Std_Logic_Vector (n - 1 downto 0);
      Sum_N: out Std_Logic_Vector (n - 1 downto 0),
      CY_Out: out Std_Logic);
end component Twobit_Add;
Signal A, B: Std_Logic_Vector (n - 1 downto 0);
Signal Sum_N: Std_Logic_Vector (n - 1 downto 0);
Signal CY_Out:Std_Logic;
begin
    TestBTwo_ADDER: Twobit_Add port map (A, B, CY_Out,
                                           Sum_N)
Process
    variable error_count:Integer: =0; -- Initial value of counts of
                                      error is 0
begin
    A <= "11"; B <= "11"; -- Test for first case A and B both 11
    Wait for 8 ns; -- Assume Adder entity gate delays < 8 ns
    If (CY_Out/=‘1’ or Sum_N/ = “10”) then
        error_count: = error_count + 1; -- Increment counts of
                                         error
    end If;
    Assert (CY_Out = ‘1’) Report “Error in CY_Out” Severity Error;
    Assert (Sum_N = “10”) Report “Error in Sum_N” Severity Error;
    -----
    A <= “00”; B <= “00”; -- Test for second case A and B both 00
    Wait for 8 ns; -- Assume Adder entity gate delays < 8 ns
    If (CY_Out/=‘0’ or Sum_N/ = “00”) then
        error_count: = error_count + 1; -- Increment counts of error
    end If;
    Assert (CY_Out = ‘0’) Report “Error in CY_Out” Severity Error;
    Assert (Sum_N = “00”) Report “Error in Sum_N” Severity Error;
    -----

```

```

A <= "10"; B <= "10"; -- Test for third case A and B both 10
Wait for 8 ns; -- Assume Adder entity gate delays < 8 ns
If (CY_Out /= '1' or Sum_N /= "00") then
    error_count := error_count + 1; -- Increment counts of error
end If;
Assert (CY_Out = '1') Report "Error in CY_Out" Severity Error;
Assert (Sum_N = "00") Report "Error in Sum_N" Severity Error;
;-----

A <= "01"; B <= "01"; -- Test for fourth case A and B both 01
Wait for 8 ns; -- Assume Adder entity gate delays < 8 ns
If (CY_Out /= '0' or Sum_N /= "10") then
    error_count := error_count + 1; -- Increment counts of error
end If;
Assert (CY_Out = '0') Report "Error in CY_Out" Severity Error;
Assert (Sum_N = "10") Report "Error in Sum_N" Severity Error;
;-----

A <= "00"; B <= "11"; -- Test for fifth case A and B = 00 and 11
Wait for 8 ns; -- Assume Adder entity gate delays < 8 ns
If (CY_Out /= '0' or Sum_N /= "11") then
    error_count := error_count + 1; -- Increment counts of error
end If;
Assert (CY_Out = '0') Report "Error in CY_Out" Severity Error;
Assert (Sum_N = "11") Report "Error in Sum_N" Severity Error;
;-----

A <= "11"; B <= "00"; -- Test for sixth case A and B = 11 and 00
Wait for 8 ns; -- Assume Adder entity gate delays < 8 ns
If (CY_Out /= '0' or Sum_N /= "11") then
    error_count := error_count + 1; -- Increment counts of error
end If;
Assert (CY_Out = '0') Report "Error in CY_Out" Severity Error;
Assert (Sum_N = "11") Report "Error in Sum_N" Severity Error;
;-----

A <= "01"; B <= "10"; -- Test for seventh case A and B = 01 and 10
Wait for 8 ns; -- Assume Adder entity gate delays < 8 ns
If (CY_Out /= '0' or Sum_N /= "11") then
    error_count := error_count + 1; -- Increment counts of error
end If;
Assert (CY_Out = '0') Report "Error in CY_Out" Severity Error;
Assert (Sum_N = "11") Report "Error in Sum_N" Severity Error;
;-----

A <= "10"; B <= "01"; -- Test for eight case A and B = 10 and 01
Wait for 8 ns; -- Assume Adder entity gate delays < 8 ns
If (CY_Out /= '0' or Sum_N /= "11") then
    error_count := error_count + 1; -- Increment counts of error
end If;
Assert (CY_Out = '0') Report "Error in CY_Out" Severity Error;

```

```
Assert (Sum_N = "11") Report "Error in Sum_N" Severity Error;
-----
Wait; -- Wait for next commands
end Process;
end RTLTestBTwobit_Add;
-----
Configuration Config_RTLTestBTwobit_Add of TestBTwobit_
Adder is
  for RTLTestBTwobit_Add
    end for;
end Config_RTLTestBTwobit_Add;
-----
```

18.2 COUNTER

\underline{Q} is an array with n elements from $n - 1$ downto 0 and logic of each element is 0 or 1. \underline{Q} output standard logic vector after increment on +ve edge when Count_Enable = '1'. An addition operator does the arithmetic addition of 1 and bits of each element 0th, 1st, 2ndup to $(n - 1)$ th of the in vector \underline{Q} .

18.2.1 Counter Circuit

RTL entity Nbit_Counter can be described as follows in VHDL. We use arithmetic addition, '+' operator and subtraction operators. We can describe RTL design codes for an entity Nbit_Counter.

Example

Nbit_Counter is an n -bit up-counter circuit, which on a +ve edge clock increments counts value at \underline{Q} provided an input Count_Enable is active. A reset input resets the counter. When next increment input is applied beyond maximum possible value of Q (all elements of vector Q are 1s) then also the counter resets to all element of $\underline{Q} = 0$ s. How will you model RTL design codes for the entity Nbit_Counter and architecture? Use behavioural model.

Solution

```
Library IEEE; -- VHDL using IEEE standard libraries
use IEEE.Std_Logic_1164.all; -- import Std_Logic from the IEEE Library
use IEEE.Std_Logic_unsigned.all; -- Std_Logic functions from the IEEE Library
                                ; -- for the unsigned arithmetic operations
use IEEE.Std_Logic_arith.all; -- Std_Logic from the IEEE Library
                                ; -- Std_Logic_arith functions enable use of
                                ; addition operator +, & and other arithmetic operators
generic (n : Natural);-- declare constant n, a natural number
                      ; -- n is the number of bits added by arithmetic counter
entity Nbit_Counter is
port (Count_Enable, Reset0, SeqClk: in Std_Logic;
```

```

    Q: out Std_Logic_Vector (n - 1 downto 0), );
end Nbit_Counter;
architecture RTLBehaviourNbit_Counter of Nbit_Counter is
Signal Present_Q: Std_Logic_Vector (n downto 0); -- Present_Q is present
value of Counts Q at output
begin
  Process (Count_Enable, Reset0, SeqClk)
  begin
    If (Reset0 = '1') then Present_Q <= Present_Q - Present_Q;
    -- Present_Q = 0
    elsif (SeqClk = '1' and SeqClk'event) then
      If (Count_Enable = '1') then Present_Q <= Present_Q + 1;
      -- Present_Q increments
      end If;
    end If;
  end Process
  Q <= Present_Q; -- Signal Present_Q concurrently assigns to
end RTLBehaviourNbit_Counter;

```

Use of subtraction operator ‘-’ can be understood as follows:

Subtraction operator, ‘-’

Assume

Signal Present_Q is ('0', '1', '0', '0', '0', '1', '0', '0')

Subtraction Present_Q - Present_Q gives ('0', '1', '0', '0', '0', '1', '0', '0') - ('0', '1', '0', '0', '1', '0', '0') and is 0 decimal. It means Present_Q is ('0', '0', '0', '0', '0', '0', '0', '0')

18.2.2 Test Bench for the Counter

Test bench generates stimulus for the simulation of the behaviour of an entity and show on screen or otherwise the situations and actions which will take place in the entity on synthesis from the logic gates.

1. When stimulus resets two-bit counter on Reset0 = '1', then Signal Present_Q = 00.
2. When Count_Enable = 1 and a positive edge is input when SeqClk = '1' and SeqClk'event is true then Present_Q becomes 01 (decimal 1).
3. When Count_Enable = 1 and next positive edge input occurs when SeqClk = '1' and SeqClk'event is true then Present_Q becomes 10 (decimal 2).
4. When Count_Enable = 1 and next positive edge input occurs when SeqClk = '1' and SeqClk'event is true then Present_Q becomes 11 (decimal 3).
5. When Count_Enable = 1 and next positive edge input occurs when SeqClk = '1' and SeqClk'event is true then Present_Q becomes 00 (decimal 0).

The stimulus of a test bench sends the signals for all possible situations and counts the number of errors occurred. If the stimulus report error counts = 0, then the entity behaviour stands tested.

Example

How can you describe RTL design codes for a *test bench* for entity Nbit_Counter and architecture in behavioural model? Select constant $n = 2$. Test possible cases. Assume up-counter.

Solution

Length of Present_Q is 2.

```
Library IEEE; -- VHDL using IEEE standard libraries
use IEEE.Std_Logic_1164.all; -- import Std_Logic from the IEEE Library
use IEEE.Std_Logic_unsigned.all; -- Std_Logic functions from the IEEE Library
                                ; -- for the unsigned arithmetic operations
use IEEE.Std_Logic_arith.all; -- Std_Logic from the IEEE Library
                                ; -- Std_Logic_arith functions enable use of
                                ; addition operator +, & and other arithmetic operators

generic (n: Natural)
entity TestBTwobit_Counter is
generic (n: Natural := 2); -- declare constant n = 2, a natural number
                            ; -- n is the number of bits added by arithmetic
                            counter
end TestBTwobit_Counter; -- Declare an entity for the test bench for n-bit
                           counter entity
architecture RTLTestBTwobit_Count of TestBTwobit_Counter is
component Twobit_Count is
    ; -- Remember n = 2 in entity TestBTwobit_Counter
    port (Count_Enable, Reset0, SeqClk: in Std_Logic;
          Q: out Std_Logic_Vector(n - 1 downto 0), );
end component Twobit_Count;
Signal Count_Enable, Reset0, SeqClk: Std_Logic;
Signal Present_Q: Std_Logic_Vector(n - 1 downto 0);
begin
    TestBTwobit_COUNTER: Twobit_Count port map (Count_Enable,
                                                Reset0, SeqClk, Present_Q)
    Process
        begin
            SeqClk <= '0'; Wait for 4 ns;
            SeqClk <= '1'; Wait for 4 ns;
        end Process; -- Declare Tclk (clock cycle time) = 8 ns
    Process
        variable error_count: Integer := 0; -- Initial error_count reset to 0
        begin
            Reset0 <= '1'; -- Active reset for initial condition Present_Q = 0
            Count_Enable <= '1'; -- Enable Counting
            Wait for 16 ns; -- wait for interval 2 × Tclk
            Reset0 <= '0'; -- Inactive reset Initial Counts increment enable
            ;-----
            Wait for 8 ns; -- wait for interval Tclk for case 1
            If (Present_Q /= 1) then error_count := error_count + 1;
        end Process;
    end;
```

```

    end If;
Assert (Present_Q = 1) Report "Test for Counter Increment to 1
Unsuccessful" Severity Error;
;-----
Wait for 8 ns; -- wait for interval Tclk for case 2
If (Present_Q /= 2) then error_count:=error_cnt+1;
end If;
Assert (Present_Q = 2) Report "Test for Counter Increment to 2
Unsuccessful" Severity Error;
;-----
Wait for 8 ns; --wait for interval Tclk for case 3
If (Present_Q /=3) then error_count:=error_cnt + 1;
end If;
Assert (Present_Q = 3) Report "Test for Counter Increment to 2
Unsuccessful" Severity Error;
;-----
Wait for 8 ns; -- wait for interval Tclk for case 4
If (Present_Q /=0) then error_count:=error_cnt + 1;
end If;
Assert (Present_Q = 0) Report "Test for Counter 0 after 3 counts
Unsucc.l" Severity Error;
;-----
Wait for 16 ns; -- wait for interval Tclk for case 5
Reset0 <= '1';
Wait 8 ns
If (Present_Q /= 0) then error_count:=error_cnt + 1;
end If;
Assert (Present_Q = 0) Report "Test for Counter Reset on Reset0 = 1
Unsuccessful" Severity Error;
;-----
If(error_count = 0) then Assert false
    Report "Testbench 2-bit Counter completed successfully"
    Severity note; -- Test result at count value
else Assert true
    Report "Something wrong, try again"
    Severity Error; "Testbench 2-bit Counter found unsuccessful"
    end If;
Wait; -- Wait for next commands
end Process;
end RTLTestBTwobit_Count;
-----  

Configuration Config_RTLTestBTwobit_Count of TestBTwobit_
Counter is
    for RTLTestBTwobit_Count
    end for;
end Config_RTLTestBTwobit_Count;

```

18.3 FLIP-FLOP

D-Flip-Flop (*FF*) and J-K Flip-Flop are described in the following subsections:

18.3.1 D-Flip-Flop

Let a *D*-flip-flop (*FF*) input is *D_in*. Assume *Q* and *Q₁* are output logic states such that *Q₁* = not *Q*. The output changes when SeqClk input event +ve edge takes place.

RTL entity *D_FF* can be described as follows in VHDL.

Example

How will you model RTL design entity and its behaviour for a *D-FF* which transfers *D_in* to output *Q* on +ve clock edge? *Q₁* is complementary logic state of *Q*.

Solution

```
Library IEEE; -- VHDL using IEEE standard libraries
use IEEE.Std_Logic_1164.all ; -- import Std_Logic from the IEEE Library
use work.all;
entity D_FF is
    port(D_in, SeqClk: in Std_Logic;
          Q, Q1: out Std_Logic);
end D_FF;
architecture RTLBehaviourD_FF of D_FF is
begin
    Process
        If (SeqClk = '1' and SeqClk'event) then
            Q <= D_in; -- +ve edge event assigns Q state from D_in logic state.
        end Process
        Q1 <= not Q; -- Signal Q1 concurrently assigns to not of Q
    end;
    RTLBehaviourD_FF;
```

18.3.2 Test Bench for the D-Flip-Flop

Test bench generates stimulus for the simulation of the behaviour of an entity and show on screen or otherwise the situations and actions which will take place in the entity on synthesis from the logic gates.

1. When SeqClock –ve edge event occurs then if *D_in* = 0 then *Q* is same as previous *Q* and *Q₁* is same as previous *Q*.
2. When SeqClock –ve edge event occurs then if *D_in* = 1 then *Q* is same as previous *Q* and *Q₁* is same as previous *Q*.
3. When SeqClock = '0' then if *D_in* = 0 then *Q* is same as previous *Q* and *Q₁* is same as previous *Q*.
4. When SeqClock = '0' then if *D_in* = 1 then *Q* is same as previous *Q* and *Q₁* is same as previous *Q*.

5. When SeqClock = '1' then if $D_{in} = 0$ then Q is same as previous Q and Q_1 is same as previous Q .
6. When SeqClock = '1' then if $D_{in} = 1$ then Q is same as previous Q and Q_1 is same as previous Q .
7. When SeqClock +ve edge event occurs then if $D_{in} = 0$ at that instance then after delay $Q = 0$ and $Q_1 = 1$.
8. When SeqClock +ve edge event occurs then if $D_{in} = 1$ at that instance then after delay $Q = 1$ and $Q_1 = 0$.

The above situations can be tested as follows:

Consider a Process by Tclk is set as 8 ns. (Tclk = interval between successive +ve edge transitions, 0.5 times Tclk = interval between successive -ve edge transitions, 0.25 times Tclk= interval between successive +ve edge transitions and middle of period for 1 or interval between successive +ve edge transitions and middle of period for 1).

Case 1: Test after Tclk (= 8 ns)

Assume that the previous value of $D = 0$ and Q is also 0 or $D = 1$ and Q is also 1. Whatever may be the case, if D sets to 1 then a +ve edge must occur or must have occurred during interval, Tclk. If we test Q after Tclk, then error should be reported if it is found 0, not 1.

Case 2: Test after 1.25 Tclk (= 10 ns)

Assume that the previous value of $D = 0$ and Q is also 0 or $D = 1$ and Q is also 1. Whatever may be the D or Q value, if D resets to 0 now then a +ve edge must have occurred during interval, Tclk. If we test Q after 1.25 times Tclk then error should be reported if it is found 1, not 0.

Case 3: Test after 2 times Tclk (= 16 ns)

Assume that the previous value of $D = 0$ and Q is also 0 or $D = 1$ and Q is also 1. Whatever may be D or Q , if D sets to 1 now then a second +ve edge must have occurred and second -ve edge must have occurred during interval = 2.Tclk. Q should remain 1 after 2.Tclk. If we test Q after 2 times Tclk, then error should be reported if it is found 0, not 1.

Case 4: Test after 0.25 Tclk (= 2 ns)

Assume that the previous value of $D = 1$ and Q is also 1. When D resets to 0, then Q should still be 1 because only after next +ve edge. Next positive edge can occur atleast after 0.5 Tclk. If we test Q after 0.25 Tclk too short interval, then error should be reported if it is found 0.

Case 5: Test after 2.75 times Tclk (= 22 ns)

Assume previous value of $D = 0$ and Q is also 0 or $D = 1$ and Q is also 1. Whatever may be D or Q , if D resets to 1 now, then a second +ve edge and two -ve edges must have also occurred during interval = 2.75 times Tclk. Q should remain 1 after 2.75.Tclk also. If we test Q after 2.75 times Tclk, then error should be reported if it is found 1, not 0.

18.12 Digital Systems: Principles and Design

The stimulus of a test bench sends the signals for all possible situations and counts the number of errors occurred. If the stimulus report error counts = 0, then the entity behaviour stands tested.

Example	How can you describe RTL design codes for a <i>test bench</i> for entity <i>D_FF</i> and architecture in behavioural model?. Test possible cases.
----------------	---

Solution

```
entity TestBD_FF is
end TestBD_FF; -- Declare an entity for the test bench for flip-flop entity
architecture RTLTestBD_FF of TestBD_FF is
    Signal D_in, SeqClk: in Std_Logic;
    Signal Q, Q1: out Std_Logic);
    Component D_FF01 is ;-- D_FF component
        port (D_in, SeqClk: in Std_Logic;
              Q, Q1: out Std_Logic);
    end component D_FF01;
begin
    Test_DFF: port map (D_in, SeqClk, Q, Q1);
    Process
        begin
            SeqClk <= '0'; Wait for 4 ms;
            SeqClk <= '1'; Wait for 4 ms;
        end Process; -- clock +ve edge, '1', -ve edge, '0' and +ve edge,
                      periodically
                      ; --occurs after 8 ms
    Process
        variable error_count: Integer := 0;
        begin
            D_in <= 1; Wait 8 ns ;-- wait for period 1 interval = Tclk
            If (Q/=‘1’) then error_count:= error_count + 1 ;-- error Case 1 If Q
                           differs
                           ; -- and does not become ‘1’
        end If;
        Assert (Q = ‘1’) Report “Q changed and does not show ‘1’ after Tclk”
               Severity Error;
        ;-----
        D_in <= 0; Wait 10 ns ; -- wait for period 1.25 interval = Tclk
        If (Q/=‘0’) then error_count:= error_count + 1 ;-- error Case 2 If Q changed
                           ; --and does not remain ‘0’
        end If;
        Assert (Q = ‘0’) Report “Q changed and not found ‘0’ after 1.25 Tclk”
               Severity Error;
        ;-----
        D_in <= 1; Wait 16 ns ; -- wait for period 2 intervals of Tclk
```

```

If ( $Q \neq '1'$ ) then error_count := error_count + 1 ; -- error Case 3 If  $Q$ 
changed
;--and does not remain '1'
end If;
Assert ( $Q = '1'$ ) Report "Q changed and does not show '1' after 2.Tclk"
Severity Error;
;-----
D_in <= 0; Wait 2 ns ; -- wait for period 0.25 interval = Tclk
If ( $Q \neq '1'$ ) then error_count := error_count + 1 ; -- error Case 4 If  $Q$ 
found changed
; even before the next +ve edge
end If;
Assert ( $Q = '1'$ ) Report "Q found changed even before next +ve edge"
Severity Error;
;-----
D_in <= 1; Wait 22 ns ; -- wait for period 2.75 intervals of Tclk
If ( $Q \neq '1'$ ) then error_count := error_count + 1 ; -- error Case 5 If  $Q$ 
changed
;--and does not remain '1'
end If;
Assert ( $Q = '1'$ ) Report "Q changed and does not show '1' after 2.75.
Tclk" Severity Error;
;-----
If (error_count = 0) then Assert false
Report "Testbench 2-bit Counter completed successfully"
Severity note; -- Test results at error count value 0
else Assert true
Report "Errors found in D-FF behaviour"
Severity Error; "Testbench D-FF found unsuccessful"
end If;
Wait; -- Wait for next commands
end Process;
end RTLTestBD_FF;
;-----
Configuration Config_RTLTestBD_FF of TestBD_FF is
for RTLTestBD_FF
end for;
end Config_RTLTestBD_FF;
;-----
```

18.3.3 JK-Flip-Flop

Let *JK*-flip-flop (*FF*) inputs are *J* and *K*. (Section 7.4) Assume Q and Q_1 are output logic states such that $Q_1 = \text{not } Q$. Assume that Q changes when SeqClk input event –ve edge takes place.

RTL entity *JK_FF* can be described as follows in VHDL.

18.14 Digital Systems: Principles and Design

Example

How will you model RTL design entity and its behaviour for a *JK-FF*? State Q and complementary state Q_1 is according to the inputs states at J and K at the instance of –ve clock edge? Q_1 is complementary logic state of Q .

Solution

```
Library IEEE ; -- VHDL using IEEE standard libraries
use IEEE.Std_Logic_1164.all ; -- import Std_Logic from the IEEE Library
use work.all;
entity JK_FF is
    port (J, K, SeqClk, Reset0: in Std_Logic;
          Q, Q1: out Std_Logic);
end JK_FF;
architecture RTLBehaviourJK_FF of JK_FF is
Signal Present_State: Std_Logic;
Signal JK_State: Std_Logic_Vector (1 downto 0);
begin
    JK_State <= J & K ; -- concatenate logic J and K into one logic vector
                      ; -- For example if  $J = '1'$  and  $K = '0'$  then  $JK\_State$  is
                      ('1', '0')
JK_P: Process (SeqClk, Reset0) is
begin (Reset = '1') then Present_State <= 0;
If (SeqClk = '0' and SeqClk'event) then
    Case (JK_State)
        when "00" => null ; -- Present_State No change
        when "01" => Present_State <= '0' ; -- Present_State reset to '0'
        when "10" => Present_State <= '1' ; -- Present_State set to '1'
        when "11" => Present_State <= not (Present_State) ; -- Toggling of
                                                               Present_State
    end Case
end if;
end Process
Q <= Present_State; -- +ve edge event assigns  $Q$  state from Present_State as per
                     J-K state truth table
Q1<= not (Present_State); -- Signal  $Q_1$  concurrently assigns to not of  $Q$ 
end RTLBehaviourD_FF;
```

18.3.4 Test Bench for the JK-Flip-Flop

Test bench generates stimulus for the simulation of the behaviour of an entity and show on screen or otherwise the situations and actions which will take place in the entity on synthesis from the logic gates.

1. When SeqClock –ve edge event occurs then if $JK = 00$ then Q is same as previous Q and Q_1 is same as previous Q .
2. When SeqClock –ve edge event occurs then if $JK_State = 01$ then Q resets to '0'.

3. When SeqClock –ve edge event occurs then if $JK_State = 10$ then Q sets to ‘1’.
4. When SeqClock –ve edge event occurs then if $JK_State = 11$ then Q resets to ‘0’ if ‘1’ and sets to ‘1’ if ‘0’ (means toggling, not operation on previous Q on each –ve edge event).
5. When SeqClock +ve edge event occurs then if $JK_State = 00$ or 01 or 10 or 11 then Q sets to ‘1’.
6. When SeqClock = ‘0’ and SeqCloc’event = false then then Q is same as previous Q and Q_1 is same as previous Q or When SeqClock = ‘1’ and SeqCloc’event = false then if $JK_State = 00$ or 01 or 10 or 11 then Q is same as previous Q and Q_1 is same as previous Q .
7. When $Reset0 = 1$ then Q resets to 0.

The above situations can be tested as follows:

Consider a Process by Tclk is set as 8 ns. (Tclk = interval between successive +ve edge transitions, 0.5 times Tclk = interval between successive –ve edge transitions, 0.25 times Tclk = interval between successive +ve edge transitions and middle of period for 1 or interval between successive +ve edge transitions and middle of period for 1).

First set $Reset0$ to ‘1’. Wait 2.5 Tclk and then reset the $Reset0$ to ‘0’:

1. Assign JK_State and Test after 1.5 Tclk (= 16 ns) after at least one –ve edge event
Assign $JK_State = 00$. Whatever may be the case, if we test Q after 1.5 Tclk, then error should be reported if Q is found changed from ‘0’.
2. Assign JK_State after 0.25 Tclk and Test after 1.25 Tclk (= 10 ns) after at least one –ve edge
Wait 0.25 Tclk, assign $JK_State = 10$. Whatever may be the case, if we test Q after 1.5 Tclk, then error should be reported if Q is not equal to ‘1’.
3. Assign JK_State and Test after 1.5 Tclk (= 16 ns) after at least one –ve edge event
Wait 0.25 Tclk, assign $JK_State = 00$. Whatever may be the case, if we test Q after 1.5 Tclk, then error should be reported if Q is found changed from ‘1’.
4. Assign JK_State after 0.25 Tclk and Test after 1.25 Tclk (= 10 ns) after at least one –ve edge event
Wait 0.25 Tclk and assign $JK_State = 01$. Whatever may be the case, if we test Q after 1.25 Tclk, then error should be reported if Q is not equal to ‘0’.
5. Assign $JK_State = 11$ after 0.25 Tclk and Test after 1.25 Tclk (= 10 ns) after at least one –ve edge
Wait 0.25 Tclk, assign $JK_State = 11$. If we test Q after 1.25 Tclk, then error should be reported if Q is equal to ‘1’ if ‘0’ earlier and equal to ‘0’ if ‘1’ (toggles).
6. Test after 1.25 Tclk (= 10 ns) after at least one –ve edge when $JK_State = 11$
If we test Q after 1.25 Tclk, then error should be reported if Q is equal to ‘1’ if ‘0’ earlier and equal to ‘0’ if ‘1’ (toggles).
7. Test after 1.25 Tclk (= 10 ns) after at least one –ve edge when $JK_State = 11$

18.16 Digital Systems: Principles and Design

Whatever may be the case, if we test Q after 1.25 Tclk, then error should be reported if Q is equal to ‘1’ if ‘0’ earlier and equal to ‘0’ if ‘1’ (toggles).

The stimulus of a test bench sends the signals for all possible situations and counts the number of errors occurred. If the stimulus report error counts = 0, then the entity behaviour stands tested.

Example

How can you describe RTL design codes for a *test bench* for entity *JK_FF* and architecture in behavioural model? Test possible cases.

Solution

```
entity TestBJK_FF is
end TestBJK_FF; -- Declare an entity for the test bench for flip-flop entity
architecture RTLTestBJK_FF of TestBJK_FF is
begin
    Component JK_FF01 is -- JK_FF component
        port (J, K, SeqClk, Reset0: in Std_Logic;
              Q, Q1: out Std_Logic);
    end component JK_FF01;
    begin
        Test_JKFF: port map (J, K, SeqClk, Reset0, Q, Q1);
        Process
            begin
                SeqClk <= '0'; Wait for 4 ms;
                SeqClk <= '1'; Wait for 4 ms;
            end Process; -- clock +ve edge, '1' after 4 ms from -ve edge and -ve edge
            after 4 ms,
            ; -- periodically. Tclk = 8 ms
        Process
            variable error_count: Integer := 0;
            begin
                Reset0 <= '1'; Wait 20 ms;
                Reset0 <= '0'; Wait 8 ms;
                If ( $Q \neq '0'$ ) then error_count := error_count + 1; -- error Case 0
                Reset0 does not reset  $Q$ 
            end If;
            Assert ( $Q = '0'$ ) Report "Reset0 does not reset  $Q$  after 3.5 Tclk also"
            Severity Error;
            ;-----
            J <= '0'; K <= '0'; Wait 10 ns; -- wait for period 1.25 times Tclk
            If ( $Q \neq '0'$ ) then error_count := error_count + 1; -- error If  $Q$  differs
            ; -- and does not become '1'
            end If;
            Assert ( $Q = '0'$ ) Report " $Q$  changed from '0' on  $J = '0'$  and  $K = '0'$  Error
            in J-K FF" Severity Error;
            ;-----
```

```

Wait 2 ns;  $J \leq '1'; K \leq '0'$ ; Wait 10 ns;-- wait for period 1.25 times Tclk
If ( $Q = '1'$ ) then error_count := error_count + 1; -- error If  $Q$  differs
; -- and does not become '1'
end If;
Assert ( $Q = '1'$ ) Report "Q is not '1' on  $J = '1'$  and  $K = '0'$  Error in J-K
FF" Severity Error;
;-----
Wait 2 ns;  $J \leq '0'; K \leq '0'$ ; Wait 10 ns;-- wait for period 1.25 times Tclk
If ( $Q = '1'$ ) then error_count := error_count + 1; -- error If Q differs
;--and does not become '0'
end If;
Assert ( $Q = '1'$ ) "Q changed from '1' on  $J = '0'$  and  $K = '0'$  Error in J-K
FF" Severity Error;
;-----
Wait 2 ns;  $J \leq '0'; K \leq '1'$ ; Wait 10 ns;-- wait for period 1.25 times
Tclk
If ( $Q = '0'$ ) then error_count := error_count + 1; -- error If  $Q$  differs
; -- and does not become '1'
end If;
Assert ( $Q = '0'$ ) "Q is not '0' on  $J = '0'$  and  $K = '1'$  Error in J-K FF"
Severity Error;
;-----
Wait 2 ns;  $J \leq '1'; K \leq '1'$ ; Wait 10 ns; -- wait for period 1.25 times
Tclk
If ( $Q = '1'$ ) then error_count := error_count + 1; -- error If  $Q$  differs
; -- and does not become '1'
end If;
Assert ( $Q = '1'$ ) "Q does not toggle to '1' on  $J = '1'$  and  $K = '1'$  Error in
J-K FF" Severity Error;
;-----
Wait 2 ns;  $J \leq '1'; K \leq '1'$ ; Wait 10 ns; -- wait for period 1.25 times
Tclk
If ( $Q = '0'$ ) then error_count := error_count + 1; -- error If  $Q$  differs
;--and does not become '1'
end If;
Assert ( $Q = '0'$ ) "Q does not toggle to '0' on  $J = '1'$  and  $K = '1'$  Error in
J-K FF" Severity Error;
;-----
If (error_count = 0) then Assert false
Report "Testbench JK-FF completed successfully"
Severity note; -- Test results at error count value 0
else Assert true
Report "Errors found in JK-FF behaviour"
Severity Error; "Testbench 2-bit Counter found unsuccessful"
end If;

```

```
Wait; -- Wait for next commands
end Process;
end RTLTestBJK_FF;
-----
Configuration Config_RTLTestBJK_FF of TestBJK_FF is
    for RTLTestBJK_FF
        end for;
end Config_RTLTestBJK_FF;
-----
```

18.4 FINITE STATE MACHINE

A finite state machine (FSM) is a circuit, which generates next state output depending on its present state. A sequential circuit made up from sequential memory section and combinational circuit can function as an FSM.

Example

Consider a 4-bit counter. How can it be called a FSM?

Solution

It can be considered as a FSM as follows: A 4-bit counter has 16 states. On a count_input, its state undergoes transition from 0000, 0001, 0010, 0011, up to 1111. On next count_input, its state will again be 0000. These states can be labelled as S₀, S₁, S₂, S₃, ... up to S₁₅ and again state of the counter will be 0000. There are 16 state transitions.

18.4.1 Finite State Machine Sequential Circuit

Figure 18.1 shows a sequential circuit entity FSMSeqCircuit. FSM sequential circuit consists of FSM_State transition Circuit_SeqM and Logic_CombinationalCircuit_C. These circuits can also be abbreviated as SeqM and Circuit_C. SeqClk event causes a state transfer to Q in synchronous sequential circuit.

1. There are two Processes. The Processes at SeqM and Circuit_C execute concurrently.
2. Memory section SeqM may consist of flip-flops and latches. A state transfer to Q takes place at SeqM after an interval, tSeqM after an SeqClk event. The Output states Q are given as inputs Q_i to Circuit_C.
3. Circuit_C generates output Y after an interval = tComCircuit. [Minimum interval for generating new $Y_i = tSeqM + tComCircuit$ = Time delay at SeqM plus in Circuit_C = tSeqM + tComCircuit.]
4. A new input Y_i enables a next sequence of the state at Q after interval tSeqM from event at SeqClk.
5. An input port is Reset0. An input event at Reset0 resets the state output Q to an initial state after tSeqM.

Initially circuit SeqM transfers the state at Y_i and generates output Q (a logic vector). It means during a time interval tSeqM the memory section SeqM executes

state transfer function and generates a state. Then, state transition function executes at Circuit_C. Transition is as per the inputs Q_i at an instance when a sequencing clock event occurs. \underline{Y} is now at next_state.

Another Process at Logic_CombinationalCircuit_C, abbreviated as Circuit_C generates output Y as per the inputs from SeqM, which means as per the present state Q . The Circuit_C processes the state transition function.

SeqM transfers to \underline{Q} the next state at \underline{Y}_1 on next sequencing clock $\underline{Y}_1 (= \underline{Y})$.

The sequential circuit FSMSeqCircuit, two concurrent Processes have the following inputs and outputs in RTL design model.

1. One port for SeqClk logic input at SeqM (memory section).
2. One input-output port \underline{Y}_1 is at SeqM from Circuit_C
3. An input port Reset0. This is for resetting the state to an initial state at \underline{Q} .
4. \underline{Y}_1 input of SeqM also interconnects to output port \underline{Y} so that a next state can be generated by SeqM. $\underline{Y}_1 = \underline{Y}$. (\underline{Y} is output of Circuit_C).
5. SeqM (memory section) output is logic vector \underline{Q} .
6. The logic vector \underline{Q}_1 also interconnects to output port logic vector \underline{Q} . $\underline{Q}_1 = \underline{Q}$.
7. Circuit_C inputs are logic vector at port \underline{Q}_1 and logic input at port transition_enable.

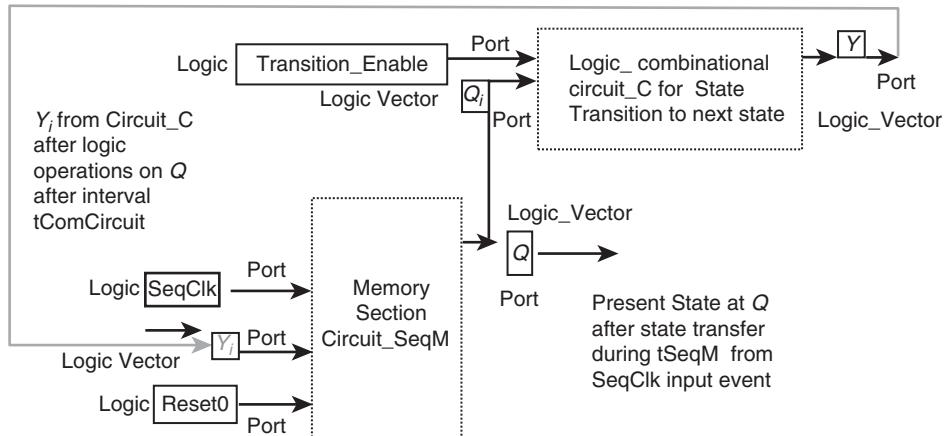


FIGURE 18.1 FSM sequential circuit consisting of FSM_StatesCircuit_SeqM and the Output states \underline{Q} are given as inputs \underline{Y}_1 after a Logic_CombinationalCircuit_C generates output \underline{Y} for next sequence of the state.

Example

How can you describe FSMSeqCircuit entity and components (memory section and combinational circuit)?

Solution

The entity can be described follows:

```
Library ieee;
use ieee.Std_Logic_1164.all;
```

```
Tclk, Tdelay: Time;
Tclk: = 50 ns; -- Interval T of Clock inputs
Tdelay: = 4 ns;
entity FSMSeqCircuit is
    n, m, i, j: Positive;
    ; -- Sequential circuit entity has two components (memory section and com-
        binational circuit).
    Port (Yi: inout Std_Logic_Vector (k - 1 downto 0));
    SeqClk: in Std_Logic;
    Reset0: in Std_Logic;
    Q: Out Std_Logic_Vector (k - 1 downto 0); -- for example, k = 4 for a
        4-bit counter.
    ; -- Logic_CombinationalCircuit_C
    Qi: inout Std_Logic_Vector (k - 1 downto 0);
    transition_enable: in Std_Logic;
    Y: out Std_Logic_Vector (k - 1 downto 0);
end entity FSMSeqCircuit;
```

Example

How can you describe architecture RTLFSMSeqC of FSMSeqCircuit? Assume that the FSM generates six states, S_0, S_1, S_2, S_3, S_4 , and S_5 . There are six finite states. Next state to S_5 is S_0 . $S_0 \rightarrow S_1, S_1 \rightarrow S_2, S_2 \rightarrow S_3, S_3 \rightarrow S_4, S_4 \rightarrow S_5, S_5 \rightarrow S_0, S_0 \rightarrow S_1, \dots$ on successive events at sequencing input clock. Assume +ve edge at SeqClk causing state transfer of Y_i to Q at memory section (flip-flops section).

Solution

```
architecture RTLFSMSeqM of FSMSeqCircuit is
    tSeqM, tSeqplusComCircuit, tComCircuit: Time
    tSeqM: = 20 ns; -- Delay in sequential circuit memory section
    tComCircuit: = 10 ns; -- Delay in combinational logic section in sequential
        circuit
    tSeqplusComCircuit: = tSeqM + tComCircuit; -- Delay of Sequential Circuit
        both sections;
    ; -- Present state at  $Q$  and next state at  $Q$  is in one of the states,
    type Q_Type is (S0, S1, S2, S3, S4, S5);
    Signal S0: Std_Logic_Vector (k - 1 down to 0);
    Signal S1: Std_Logic_Vector (k - 1 down to 0);
    Signal S2: Std_Logic_Vector (k - 1 down to 0);
    Signal S3: Std_Logic_Vector (k - 1 down to 0);
    Signal S4: Std_Logic_Vector (k - 1 down to 0);
    Signal S5: Std_Logic_Vector (k - 1 down to 0);
    Signal Next_State, Present_state: Q_type;
    Signal Present_State: Q_Type:= S0;
    Signal Next_State: Q_Type:= S1;
    Process (Q, Present_State)
        ; -- Define the assignments with Present state input wired connection to Q
```

```

begin
  Q<= PresentState'Delayed (0); -- Interconnect Y output to Yi
  input
end Process;
Process (Q, Qi); -- Define the assignments with Qi as input from connection
to Q
begin
  Qi<= Q'Delayed (0); -- Interconnect Qi input to Q output
end Process;
Process (Y, Next_State)
  ; -- Define the assignments with Present state input Yi wired connection
  to Y
begin
  Y<= NextState'Delayed (0); -- Interconnect Y output to Yi input
end Process;
Process (Yi, Y) -- Define the assignments with Yi as input from connec-
tion to Y
begin
  Yi<= Y'Delayed (0); -- Interconnect Y output to Yi input
end Process;
Process (SeqClk, Yi, Reset0, Present_State, Next_State);
  ; -- SeqClk, Yi, Reset0 are inputs and Present_State is output
begin
  ; -- Process body for SeqM
  If (Reset0='1') then Present_State <= S0 after tSeqM
    elsif (SeqClk'event and SeqClk='1') then
      Present_State <= Next_State; ;-- SeqClk event results in state
      transition at Q to next state.
  end If
end Process;
Process (transition_enable,Next_State,Qi)
  ; -- Process body for Logic_CombinationalCircuit_C
begin
  If (Qi = S0 and transition_enable = 0) then Next_State <= S0
  elsif (Qi = S0 and transition_enable ='1' then Next_State
  <= S1;
  end if;
-----
If (Qi = S1 and transition_enable = 0) then Next_State <= S1
elsif (Qi = S1 and transition_enable ='1' then Next_State
<= S2;
  end if;
-----
If (Qi = S2 and transition_enable =0) then Next_State <= S2
elsif (Qi = S2 and transition_enable = '1' then Next_State
<= S3;

```

```
        end if;
-----
If (Qi = S3 and transition_enable = 0) then Next_State <= S3
elsif (Qi = S3 and transition_enable = '1' then
    Next_State <= S4;
    end if;
-----
If (Qi = S4 and transition_enable = 0) then Next_State <= S4
elsif (Qi = S4 and transition_enable = '1' then
    Next_State <= S5;
    end if;
-----
If (Qi = S5 and transition_enable = 0) then Next_State <= S5
elsif (Qi = S5 and transition_enable = '1' then
    Next_State <= S0;
    end if;
end Process;
end RTLFSMSeqC
```

18.4.2 Test Bench for the Finite State Machine

Test bench generates stimulus for the simulation of the behaviour of an entity and show on screen or otherwise the situations and actions which will take place in the entity on synthesis from the logic gates. Assume an FSM for state transitions between S_0 , S_1 , and S_2 states. Assume $S_0 = ('0', '0', '0', '0')$, $S_1 = ('1', '0', '0', '0')$, $S_2 = ('0', '0', '1', '0')$. Firstly, the following is tested.

1. When Reset0 = 1 then $\underline{Q} = ('0', '0', '0', '0')$ after 2 Tclk where Tclk is sequencing clock period.
2. When Reset0 = 0 then \underline{Q} remains same as previous state at \underline{Q} .
3. When Reset0 = 0 then $\underline{Q} = ('0', '0', '0', '0')$ after 2 Tclk where Tclk is sequencing clock period.

Then Reset0 <= 0 and the following sequential actions are tested.

1. When SeqClock +ve edge event occurs then FSM memory section of FFs state transfers to \underline{Q} . \underline{Q} becomes S_1 .
2. When SeqClock +ve edge event occurs then FSM memory section of FFs state transfers to \underline{Q} . \underline{Q} becomes S_2 .
3. When SeqClock +ve edge event occurs then FSM memory section of FFs state transfers to \underline{Q} . \underline{Q} becomes S_0 .
4. When SeqClock +ve edge event occurs then FSM memory section of FFs state transfers to \underline{Q} . \underline{Q} becomes S_1 .

State machine keeps on generating next states on each sequencing clock event.

The stimulus of a test bench sends the signals for all possible situations and counts the number of errors occurred. If the stimulus report error counts = 0, then the entity behaviour stands tested.

Example

How can you describe RTL design codes for a *test bench* for entity FSM for state transitions to S_0 , S_1 , S_2 , and S_3 ? Describe **architecture** in behavioural for test bench? Test possible cases.

Solution

```

Library ieee;
Use ieee.Std_Logic_1164.all;
entity TestBFSMS0_S201 is
end TestBFSMS0_S2; -- Declare an entity for the test bench for flip-flop
entity
architecture RTLTestBFSMS0_S2 of TestBFSMS0_S201 is
    Tclk, tSeqM, tSeqplusComCircuit, tComCircuit: Time
    Tclk:= 50 ns;
    tSeqM:= 20 ns; -- Delay in sequential circuit memory section
    tComCircuit:= 10 ns; -- Delay in combinational logic section in sequential
    circuit
    tSeqplusComCircuit:= tSeqM + tComCircuit; -- Delay of Sequential Circuit
    both sections;
    generic (n: Natural:= 2); -- declare constant n = 2, a natural number
    Signal S0: Std_Logic_Vector (k - 1 down to 0):= ('0', '0', '0', '0');
    Signal S1: Std_Logic_Vector (k - 1 down to 0):= ('1', '0', '0', '0');
    Signal S2: Std_Logic_Vector (k - 1 down to 0):= ('0', '0', '1', '0');
begin
    Component TestFSMS0_S2 is
        n, m, i, j: Positive;
        ; -- Sequential circuit entity has two components (memory section and com-
        binational circuit).
    Port (
        SeqClk: in Std_Logic;
        Reset0: in Std_Logic;
        Q: Out Std_Logic_Vector (k - 1 downto 0);
        ; -- Logic_CombinationalCircuit_C
        transition_enable: in Std_Logic;
    end Component TestFSMS0_S2;
    begin
        Test_FSMS0S2: port map (Q, SeqClk, Reset0);
        Process
        begin
            SeqClk <= '0'; Wait for 25 ms;
            SeqClk <= '1'; Wait for 25 ms;
        end Process; -- clock +ve edge, '1' after 4 ms from -ve edge and -ve edge
        after 4 ms,
            ; -- periodically. Tclk = 8 ms
        Process
            variable error_count: Integer:= 0;

```

```
begin
    Reset0 <= '1'; Wait 80 ms;
    Reset0 <= '0'; Wait 30 ms; Wait for at least 0.6 Tclk
    If ( $Q \neq S0$ ) then error_count := error_count + 1; -- error Case 0
    Reset0 does not reset  $Q$ 
    end If;
    Assert ( $Q = '0'$ ) Report "Reset0 does not reset state to S0 after 2.5 Tclk
    also" Severity Error;
    ;-----
    Wait 60 ns; Wait for 1.2 Tclk
    If ( $Q \neq S1$ ) then error_count := error_count + 1; -- error Case 1
    end If;
    Assert ( $Q = S1$ ) "S0 to S1 transition unsuccessful in 1.2 Tclk also"
    Severity Error;
    ;-----
    Wait 50 ms; Wait for Tclk
    If ( $Q \neq S2$ ) then error_count := error_count + 1; -- error Case 2
    end If;
    Assert ( $Q = S2$ ) Report "Reset0 does not reset state to S0 after 1.2 Tclk"
    Severity Error;
    ;-----
    Wait 50 ms; Wait for Tclk
    If ( $Q \neq S0$ ) then error_count := error_count + 1; -- error Case 3
    end If;
    Assert ( $Q = S0$ ) Report "S0 to S1 transition unsuccessful in Tclk also"
    Severity Error;
    ;-----
    Wait 50 ns; Wait for Tclk
    If ( $Q \neq S1$ ) then error_count := error_count + 1; -- error Case 4: S0 did
    not change to S1 at  $Q$ 
    end If;
    Assert ( $Q = S1$ ) "S0 to S1 transition unsuccessful in Tclk also" Severity
    Error;
    ;-----
    Wait 50 ms; Wait for Tclk
    If ( $Q \neq S2$ ) then error_count := error_count + 1; -- error Case 05
    end If;
    Assert ( $Q = S2$ ) Report "S1 to S2 transition unsuccessful in Tclk"
    Severity Error;
    ;-----
    Wait 50 ms; Wait for at least Tclk
    If ( $Q \neq S0$ ) then error_count := error_count + 1; -- error Case 6
    end If;
    Assert ( $Q = S0$ ) Report "S2 to S0 transition unsuccessful in Tclk
    "Severity Error;
    ;-----
```

```

If (error_count = 0) then Assert false
Report "Testbench FSMS0_S2 completed successfully"
Severity note; -- Test results at error count value 0
else Assert true
Report "Errors found in JK-FF behaviour"
Severity Error; "Testbench 2-bit Counter found unsuccessful"
end If;
Wait; -- Wait for next commands
end Process;
end RTLTestBFSMS0_S2;
-----
Configuration Config_RTLTestBFSMS0_S2 of TestBFSMS0_S201is
  for RTLTestBFSMS0_S2
  end for;
end Config_RTLTestBFSMS0_S2;
-----

```

18.5 MULTIPLEXER

Assume n to 1 multiplexer. (Section 13.3) Logic vector \underline{Y} is output as per one of the n input logic vectors. Each input may be a logic vector of length 1 or 2 or 4 or ... \underline{Y} output is assigned to one of input vectors of length m . Select_in is a logic vector of length k .

A 2 to 1 multiplexer (MUX21) has $n = 2$ and $k = 1$. 4 to 1 multiplexer (MUX41) has $n = 4$ and $k = 2$. 8 to 1 multiplexer (MUX81) has $n = 8$ and $k = 3$. [$n = 2^k$. $k = 3$ for 8 input logic vectors.]

18.5.1 Multiplexer Circuit

Assume 2 to 1 multiplexer (MUX21). Y is output of MUX21 as per one of the inputs of A or B . Y output is assigned one of A logic states or B logic states as per Select_in input logic vector of length 1. Demultiplexer has $n = 2$, $m = 1$ and $k = 1$.

Example

How will you model RTL design entity and its behaviour for a 2 to 1 multiplexer (MUX21)? State Y is according to the input Select_in for selecting the addressed channel.

Solution

```

Library IEEE; -- VHDL using IEEE standard libraries
use IEEE.Std_Logic_1164.all ; -- import Std_Logic from the IEEE Library
generic (n, m, k:Natural); -- declare constants n and m two natural numbers
                           ;-- n is the number of elements in inputs A in MUXnto1
                           ;-- m is the number of elements in A_Select
n:= 2;
m:= 1;
k:= 1);

```

```
entity MUX_2to1 is
    port (A: in Std_Logic_Vector (n - 1 downto 0);
          B: in Std_Logic_Vector (n - 1 downto 0);
          Select_in: in Std_Logic_Vector (m - 1 downto 0); -- m = 1 for two
          input logic vectors
          Y: out in Std_Logic_Vector (n - 1 downto 0);
end MUX_2to1;
architecture RTLMUX_2to1 of MUX_2to1 is
begin
    Process (A, B, Select_in)
        begin
            If (A_Select = "0") then Y <= A
            elsif (Select_in = "1") then Y <= B;
        Process
            end if;
        end Process;
    end RTLMUX_2to1;
```

18.5.2 Test Bench for the Multiplexer (2:1)

Test bench generates stimulus for the simulation of the behaviour of an entity and show on screen or otherwise the situations and actions, which will take place in the entity on synthesis from the logic gates.

When stimulus input Select_in = "0", then Signal Y = A.

When stimulus input Select_in = "1", then Signal Y = B.

The stimulus of a test bench sends the signals for all possible situations and counts the number of errors occurred. If the stimulus report error counts = 0, then the entity behaviour stands tested.

Example

How can you describe RTL design codes for a *test bench* for entity MUX_2to1 and architecture in behavioural model? Select constant $n = 2$ and $m = 1$. Test possible cases.

Solution

Length of Y is 2.

Library IEEE; -- VHDL using IEEE standard libraries

```
use IEEE.Std_Logic_1164.all ; -- import Std_Logic from the IEEE Library
                           ; addition operator +, &, and other arithmetic
                           operators
```

```
generic (n:Natural)
```

```
entity TestBMUX_2to1 is
```

```
end TestBMUX_2to1; -- Declare an entity for the test bench for n-bit counter
entity
```

```
architecture RTLTestBMUX2_1 of TestBMUX_2to1 is
```

```
generic (n:Natural:= 2); -- declare constant n = 2, a natural number
```

```

; -- n is the number of bits added by arithmetic
counter
generic (m: Natural := 1); -- declare constant m = 1, a natural number
Signal A: Std_Logic_Vector (n - 1 downto 0) := "00"; -- Initial Assignment to
A is ('0', '0')
Signal B: Std_Logic_Vector (n - 1 downto 0) := "01"; -- Initial Assignment to
B is ('0', '1')
Signal Select_in: Std_Logic_Vector (m - 1 downto 0) := "1"; -- Initial
Assignment is ('1')
Signal Y: Std_Logic_Vector (n - 1 downto 0) := "1"; -- Initial Assignment to
Y is ('0', '1')
Component TestMUX2_1 is
    port (A, B: in Std_Logic_Vector (n - 1 downto 0);
          Select_in: in Std_Logic_Vector (m - 1 downto 0);
          Y: out Std_Logic_Vector (n - 1 downto 0)); -- Remember n = 2 in entity
end Component TestMUX2_1;
begin
    TestBM21: TestMUX2_1 port map (A, B, Select_in, Y)
Process
begin
    A <= "00"; -- Assign A = ('0', '0')
    B <= "01"; -- Assign B = ('0', '1')
    Wait for 8 ns;
    Select_in <= "1"; -- Assign A = ('1')
    Wait for 1 ns;
    If (Y /= "01") then error_count = error_count+1;
    end If;
    Assert (Y = "01") Report "Y is not from address 1 (channel 1)" Severity
Error;
-----
    B <= "00"; -- Assign B = ('0', '0')
    Wait for 8 ns;
    Select_in <= "1"; -- Assign A = ('1')
    Wait for 1 ns;
    If (Y /= "00") then error_count = error_count+1;
    end If;
    Assert (Y = "00") Report "Y is not from address 1 (channel 1)" Severity
Error;
-----
    B <= "10"; -- Assign B = ('1', '0')
    Wait for 8 ns;
    Select_in <= "1"; -- Assign A = ('1')
    Wait for 1 ns;
    If (Y /= "10") then error_count = error_count+1;
    end If;

```

18.28 Digital Systems: Principles and Design

```
Assert (Y = "10") Report "Y is not from address 1 (channel 1)" Severity
Error;
;-----
B <= "11"; -- Assign B = ('1', '1')
Wait for 8 ns;
Select_in <= "1"; -- address = ('1')
Wait for 1 ns;
If (Y /= "11") then error_count = error_count+1;
end If;
Assert (Y = "11") Report "Y is not from address 1 (channel 1)" Severity
Error;
;-----
B <= "00"; -- Assign B = ('0', '0')
A <= "01"; -- Assign A = ('0', '1')
Wait for 8 ns;
Select_in <= "0"; -- Assign address = ('0')
Wait for 1 ns;
If (Y /= "01") then error_count = error_count+1;
end If;
Assert (Y = "01") Report "Y is not from address 0 (channel 0)" Severity
Error;
;-----
A <= "00"; -- Assign A = ('0', '0')
Wait for 8 ns;
Select_in <= "0"; -- Assign address = ('0')
Wait for 1 ns;
If (Y /= "00") then error_count = error_count+1;
end If;
Assert (Y = "00") Report "Y is not from address 0 (channel 0)" Severity
Error;
;-----
A <= "10"; -- Assign A = ('1', '0')
Wait for 8 ns;
Select_in <= "0"; -- Assign address = ('0')
Wait for 1 ns;
If (Y /= "10") then error_count = error_count+1;
end If;
Assert (Y = "10") Report "Y is not from address 10 (channel 0)" Severity
Error;
;-----
A <= "11"; -- Assign A = ('1', '1')
Wait for 8 ns;
Select_in <= "0"; -- Assign address = ('0')
Wait for 1 ns;
If (Y /= "11") then error_count = error_count+1;
```

```

        end If;
        Assert ( $\underline{Y}$  = "11") Report "Y is not from address 0 (channel 0)" Severity
Error;
-----
If (error_count = 0) then Assert false
    Report "TestbenchMUX_21 completed successfully"
    Severity note; -- Test result at count value
else Assert true
    Report "Mux test unsuccessful"
    Severity Error; "Testbench 2-bit Counter found unsuccessful"
end If;
Wait; -- Wait for next commands
end Process;
end RTLTestBMUX2_1;
-----
Configuration Config_RTLTestBMUX2_1 of TestBMUX2_1 is
    for RTLTestBMUX2_1
    end for;
end Config_RTLTestBMUX2_1;

```

18.6 DEMULTIPLEXER

Assume 1 to n demultiplexer. (Section 13.3) Logic vectors \underline{Y} is output of length n . An element of array $\underline{Y} = \underline{A}$. Each output may be a logic vector of length 1 or 2 or 4 or ... up to m . \underline{Y} output is assigned to input vectors of length m . Select_out is a logic vector of length k . Which element is assigned to the input logic \underline{A} that depends on Select_out logic vector of length k .

A 2 to 1 demultiplexer (deMUX1_2) has $n = 2$ and $k = 1$. 4 to 1 demultiplexer (MUX1_4) has $n = 4$ and $k = 2$. 8 to 1 demultiplexer (MUX1_8) has $n = 8$ and $k = 3$. [$n = 2^k$. $k = 3$ for 8 input logic vectors.]

18.6.1 Demultiplexer Circuit

Assume 1 to 2 demultiplexer (demux1_2). (Section 13.4) \underline{A} is input of Demux12. Outputs are \underline{Y}_0 or \underline{Y}_1 . \underline{A} input is assigned one of \underline{Y}_0 logic states or \underline{Y}_1 logic states as per Select_out input logic vector of length 1.

Example

How will you model RTL design entity and its behaviour for a 1 to 2 demultiplexer (DeMUX12)? Y_0 or Y_1 is assigned to State A is according to Address Select output at Select_out port.

Solution

```

Library IEEE; -- VHDL using IEEE standard libraries
use IEEE . Std_Logic_1164.all; -- import Std_Logic from the IEEE Library
generic (n, m, k:Natural); -- declare constants n and m two natural numbers

```

```
; -- n is the number of elements in outputs A in
; -- DeMUXnto1
; -- m is the number of elements in Y_Select
entity DeMUX_2to1 is
    generic (n: Natural := 2)
    generic (m: Natural := 1, m:Natural := 1);
    generic (k: Natural := 1, k:Natural := 1);
    port (YO: out Std_Logic_Vector (n - 1 downto 0);
          Y1: out Std_Logic_Vector (n - 1 downto 0);
          Select_out: in Std_Logic_Vector (m - 1 downto 0); -- m = 1 for
                           two output logic vectors
          A: in Std_Logic_Vector (k - 1 downto 0);
    end DeMUX_2to1;
architecture RTLDeMUX_2to1 of DeMUX_2to1 is
begin
    Process (YO, Y1, A, Select_out)
        begin
            If(Select_out = "0") then YO := A
            elsif (Select_out = "1") then Y1 := A;
        Process
        end if;
    end Process;
end RTLDeMUX_2to1;
```

18.6.2 Test Bench for the Demultiplexer (2:1)

Test bench generates stimulus for the simulation of the behaviour of an entity and show on screen or otherwise the situations and actions, which will take place in the entity on synthesis from the logic gates.

When stimulus output Select_out = "0", then Signal Y₀ = A.

When stimulus output Select_out = "1", then Signal Y₁ = B.

The stimulus of a test bench sends the signals for all possible situations and counts the number of errors occurred. If the stimulus report error counts = 0, then the entity behaviour stands tested.

Example

How can you describe RTL design codes for a *test bench* for entity DeMUX_2to1 and architecture in behavioural model? Select constant $n = 2$ and $m = 1$. Test possible cases.

Solution

Length of A is 2.

Library IEEE; -- VHDL using IEEE standard libraries

```
use IEEE.Std_Logic_1164.all ; -- import Std_Logic from the IEEE Library
                            ; addition operator +, & and other arithmetic
                            ; operators
```

```
generic (n:Natural)
```

```
entity TestBDeMUX_2to1 is
```

```

end TestBDeMUX_2to1;-- Declare an entity for the test bench for n-bit counter
entity
architecture RTLTestBDeMUX1_2 of TestBDeMUX_2to1 is
generic (n:Natural:= 2); -- declare constant n = 2, a natural number
; -- n is the number of bits added by arithmetic
counter
generic (m:Natural:=1) ; -- declare constant m = 1, a natural number
Signal Y0: Std_Logic_Vector (n - 1 downto 0):= "00"; -- Initial Assignment
to Y0 is ('0', '0')
Signal Y1: Std_Logic_Vector (n - 1 downto 0):= "01"; -- Initial Assignment
to Y1 is ('0', '1')
Signal Select_out: Std_Logic_Vector (m - 1 downto 0):= "1"; -- Initial
Assignment is ('1')
Signal A: Std_Logic_Vector (n - 1 downto 0):= "1"; -- Initial Assignment to
Y0 is ('0', '1')
Component TestDeMUX1_2 is
port (Y0, Y1: out Std_Logic_Vector (n - 1 downto 0);
Select_out: out Std_Logic_Vector (m - 1 downto 0);
A: out Std_Logic_Vector (n - 1 downto 0)) ; -- Remember n = 2 in
entity
end Component TestDeMUX1_2;
begin
TestBDeM12: TestDeMUX1_2 port map (Y0, Y1, Select_out, A)
Process
begin
A<= "01";-- Assign Y0 = ('0', '0')
Wait for 8 ns;
Select_out<= "1"; -- Assign address ('1')
Wait for 1 ns;
If (Y1 /= "01") then error_count = error_count+1;
end If;
Assert (Y1 = "01") Report "Y1 is not addressed at address 1 (channel 1)"
Severity Error;
-----
A<= "00"; -- Assign A= ('0', '1')
Wait for 8 ns;
Select_out<= "1"; -- Assign address ('1')
Wait for 1 ns;
If (Y1 /= "00") then error_count = error_count+1;
end If;
Assert (Y1 = "00") Report "Y1 is not addressed at address 1 (channel 1)"
Severity Error;
-----
A<= "10"; -- Assign A= ('1', '0')
Wait for 8 ns;

```

```
Select_out <= "1"; -- Assign address ('1')
Wait for 1 ns;
If (Y1 /= "10") then error_count = error_count+1;
end If;
Assert (T = "10") Report "Y1 is not addressed at address 1 (channel 1)"
Severity Error;
;-----
A <= "11";-- Assign Y1 = ('1', '1')
Wait for 8 ns;
Select_out <= "1"; -- address = ('1')
Wait for 1 ns;
If (Y1 /= "11") then error_count = error_count+1;
end If;
Assert (Y1 = "11") "Y1 is not addressed at address 1 (channel 1)" Severity
Error;
;-----
A <= "01"; -- Assign Y0 = ('0', '0')
Wait for 8 ns;
Select_out <= "0"; -- Assign address ('0')
Wait for 1 ns;
If (Y0 /= "01") then error_count = error_count + 1;
end If;
Assert (Y0 = "01") Report "Y0 is not addressed at address 0 (channel 0)"
Severity Error;
;-----
A <= "00"; -- Assign A = ('0', '0')
Wait for 8 ns;
Select_out <= "0"; -- Assign address ('0')
Wait for 1 ns;
If (Y0 /= "00") then error_count = error_count+1;
end If;
Assert (Y0 = "00") Report "Y0 is not addressed at address 0 (channel 0)"
Severity Error;
;-----
A <= "10"; -- Assign A = ('1', '0')
Wait for 8 ns;
Select_out <= "0"; -- Assign address ('0')
Wait for 1 ns;
If (Y0 /= "10") then error_count = error_count+1;
end If;
Assert (Y0 = "10") Report "Y0 is not addressed at address 1 (channel 1)"
Severity Error;
;-----
A <= "11"; -- Assign Y1 = ('1', '1')
```

```

Wait for 8 ns;
Select_out <= "0"; -- address = ('0')
Wait for 1 ns;
If (Y0 /= "11") then error_count = error_count+1;
end If;
Assert (Y0 = "11") "Y1 is not addressed at address 1 (channel 1)" Severity
Error;
-----
If (error_count =0) then Assert false
Report "TestbenchDeMUX_12 completed successfully"
Severity note; -- Test result at count value
else Assert true
Report "DeMux test unsuccessful"
Severity Error; "Testbench deMux found unsuccessful"
end If;
Wait; -- Wait for next commands
end Process;
end RTLTestBDeMUX1_2;
-----
Configuration Config_RTLTestBDeMUX1_2 of TestBDeMUX1_2 is
for RTLTestBDeMUX1_2
end for;
end Config_RTLTestBDeMUX1_2;

```

■ EXAMPLES

Example 18.1

What are the design difficulties without VHDL for synthesizing an adder?

Solution

If the hardware description language is not used for the gate level synthesis then following will be the procedure to synthesize.

An n -bit adder needs n full adders components as shown in Figure 3.2 for $n = 4$, 0th adder functioning as half adder. This can be observed as follows:

Assume $n = 8$ and A is 01100100 (= 100d). 0 to $n - 1$ elements of A are 0,0,1,0,0,1,1,0,0 (0th element is rightmost). Assume B is 11100100 (= 228d), 0 to $n - 1$ elements are 0,0,1,0,0,1,1,0,1 (0th element is on the right and 7th element leftmost).

A carry will be generated 01100100 (= 100d) + 11100100 (= 228d) > 255 d and answer is 01001000 (= 72 d) and Cy_out is 1. The output is logic vectors.

Eight full adder components are required for 8-bit addition. Therefore, n full adder components are required for n -bit addition.

Each full adder needs the XOR and AND gates. Chapter 3 explained the details of binary arithmetic—addition. Chapter 5 showed that a 4-bit adder component for the 0th bit (lowest bit of A and B) may add just two bits (half_adder). (Carry at input = 0). (Figure 3.1(a)). Adder (full_adder) component was shown in Figure 3.1(b).

18.34 Digital Systems: Principles and Design

Full adder can also function for 0th element as the half adder if input carry is always 0 (Figure 3.1(b)). Figure 3.2 showed the circuit of 4-bit adder consisting of four full adders with 0th full adder input carry is always 0. Eq. 3.14 describes $n = 4$ adder truth table.

Gate level synthesis using the RTL model of 4-bit adder will give the circuits same as previously described in Figures 3.1 and 3.2.

Example 18.2

Nbit_AdderCumSub is an n -bit adder-cum subtractor circuit. How will you model RTL design codes for the entity Nbit_AdderCumSub? Use behavioural model.

Solution

```
Library IEEE; -- VHDL using IEEE standard libraries
use IEEE . Std_Logic_1164.all ; -- import Std_Logic from the IEEE Library
use IEEE . Std_Logic_unsigned.all ; -- Std_Logic functions from the IEEE Library
                                ; -- for the unsigned arithmetic operations
use IEEE . Std_Logic_arith.all ; -- Std_Logic from the IEEE Library
                                ; -- Std_Logic_arith functions enable use of
                                ; addition operator +, &, and subtraction – arithmetic
                                ; operators
generic (n : Natural) ; -- declare constant n, a natural number
                        ; -- n is the number of bits added by arithmetic adder or
                        ; -- subtracted by subtractor when Signal Sub = 1
entity Nbit_AdderCumSub is
    port (A, B: in Std_Logic_Vector (n - 1 downto 0);
          Sub : in Std_Logic;
          Y_N: out Std_Logic_Vector (n - 1 downto 0),
          CY_Out: out Std_Logic);
end Nbit_AdderCumSub;
```

Example 18.3

Nbit_AdderCumSub entity adds n -bits with n -bits, and generates a carry in the output when an input Sub = 0. It subtracts n -bits with n -bits, and generates a borrow bit in the output when an input Sub = 1. How will you model RTL design codes for the architecture of entity Nbit_AdderCumSub? Use behavioural model.

```
architecture RTLBehaviourNbit_AdderCumSub of Nbit_AdderCumSub is
    Signal YA: Std_Logic_Vector (n downto 0): = ('0' & A);
                ; -- YA signal of extended vector of A from n - 1 to n elements
                ; -- extended to  $n^{\text{th}}$  length.  $n^{\text{th}}$  element of YA = logic 0.
    Signal YB: Std_Logic_Vector (n downto 0): = ('0' & B);
                ; -- YB signal of extended vector of B from n - 1 to n elements
                ; -- extended to  $n^{\text{th}}$  length.  $n^{\text{th}}$  element of YA = logic 0.
    Signal Y1_N: Std_Logic_Vector (n downto 0); -- Y1_N signal of extended vector
                                                ; -- length, n.
                ; -- Y_N n - 1 to 0 extends to Signal Y1_N
```

```

begin
  If (Sub = '0') then
    Y1_N <= A + B; -- Addition of n + 1 length YA vector and n + 1 length YB
    vector.
    Y_N <= Y1_N (n - 1 downto 0); -- only n elements, n - 1 downto 0 are
                                     ; -- the result of addition, Y_N
    CY_Out <= Y1_N (n) -- nth element is the carry result after the sum
  elsif (Sub = '1') then
    Y1_N <= A - B; -- Subtraction of n + 1 length YA vector and n + 1 length YB
    vector.
    Y_N <= Y1_N (n - 1 downto 0) ; -- only n elements, n - 1 downto 0 are
                                     ; -- the result of subtraction, Y_N
    CY_Out <= Y1_N (n); -- nth element is the carry result after the sum
  end if
end RTLBehaviourNbit_AdderCumSub;

```

Example 18.4

- How will you concatenate a Concatenation operator, & the vector A elements n-1 downto 0: ('1', '1', '0', '0', '0', '1', '0', '0') such that concatenate vector YA is ('1', '1', '1', '1', '1', '1', '1', '1', '1', '0', '0', '1', '0', '0')?
- How will you concatenate a Concatenation operator, & the vector A elements n-1 downto 0: ('1', '1', '0', '0', '0', '1', '0', '0') such that concatenate vector YB is ('1', '1', '0', '0', '0', '1', '0', '0', '1', '1', '1', '1', '1', '1', '1', '1')?

Solution

- Signal YA: in Std_Logic_Vector (n downto 0): = ('1', '1', '1', '1', '1', '1', '1', '1', & A); -- YA 16 elements in the input are '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '0', '0', '1', '0', '0' in order (n downto 0) after the concatenation operation.
- Signal YB: in Std_Logic_Vector (n downto 0): = (A & '1', '1', '1', '1', '1', '1', '1', '1', '1',); -- YB sixteen elements in the input are '1', '1', '0', '0', '0', '1', '0', '0', '1', '1', '1', '1', '1', '1' in order (n downto 0) after the concatenation operation.

Example 18.5

How will you test Adder Cum Subtractor?

Solution

Test bench generates stimulus for the simulation of the behaviour of an entity and show on screen or otherwise the situations and actions, which will take place in the entity on synthesis from the logic gates.

When stimulus sets Signals A = 00 and B = 01 to a two-bit AdderCumSub. If Y_N shows 01 and CY_Out shows 0 when Sub = 0, the AdderCumSub is tested for addition of one set of vectors, 00 and 01. The error count parameter do not increase.

When stimulus sets Signals A = 00 and B = 01 to a two-bit AdderCumSub. If Y_N shows 11 and CY_Out shows 1 when Sub = 1, the AdderCumSub is tested

for subtraction of one set of vectors, 00 and 01. The error count parameter do not increase.

When stimulus sets Signals $\underline{A} = 11$ and $\underline{B} = 11$ to a two-bit AdderCumSub. If Y_N shows 10 and CY_Out shows 1 when Sub = 0, the AdderCumSub is tested for addition of one set of vectors, 11 and 11. The error count parameter do not increase.

When stimulus sets Signals $\underline{A} = 11$ and $\underline{B} = 11$ to a two-bit AdderCumSub. If Y_N shows 00 and CY_Out shows 0 when Sub = 1, the AdderCumSub is tested for subtraction of one set of vectors, 11 and 11. The error count parameter do not increase.

The stimulus of a test bench sends the signals for all possible combinations and counts the number of errors occurred for Sub = 0. Then stimulus of a test bench sends the signals for all possible combinations and counts the number of errors occurred for Sub = 1. If the stimulus report error counts = 0, then the entity adder as well as subtractor behaviour stands tested.

Example 18.6

How can you describe RTL design codes for a *test bench* for entity Nbit_AdderCumSub and **architecture** in behavioural model? Select constant $n = 2$. Test two cases for Sub = 0 and two cases for Sub = 1.

Solution

Length of \underline{A} is 2, \underline{B} is 2, and YA is 3. Length of Y_N is 2 and CY_Out is logic ‘0’ or ‘1’.

Library IEEE; -- VHDL using IEEE standard libraries

```
use IEEE.Std_Logic_1164.all      ; -- import Std_Logic from the IEEE Library
use IEEE.Std_Logic_unsigned.all  ; -- Std_Logic functions from the IEEE Library
                                ; -- for the unsigned arithmetic operations
use IEEE.Std_Logic_arith.all     ; -- Std_Logic from the IEEE Library
                                ; -- Std_Logic_arith functions enable use of
                                ; addition operator +, & and other arithmetic
                                ; operators
```

generic (n:Natural)

entity TestBTwobit_AdderCumSub is

generic (n:Natural:= 2) ; -- declare constant n = 2, a natural number

; -- n is the number of bits added by arithmetic
 AdderCumSub

end TestBTwobit_AdderCumSub; -- Declare an entity for the test bench for
n-bit AdderCumSub entity

architecture RTLTestBTwobit_AddSub of TestBTwobit_AdderCumSub
is

component Twobit_AddSub is

; -- Remember n = 2 in entity TestBTwobit_AdderCumSub

port (\underline{A} , \underline{B} : in Std_Logic_Vector (n - 1 downto 0);

Sub: in Std_Logic;

Y_N : out Std_Logic_Vector (n - 1 downto 0),

```

    CY_Out: out Std_Logic);
end component Twobit_AddSub;
Signal A, B: Std_Logic_Vector (n - 1 downto 0);
Signal Y_N: Std_Logic_Vector (n - 1 downto 0);
Signal CY_Out: Std_Logic;
begin
    TestBTwo_ADDERCUMLSUB: Twobit_AddSub port map (A, B,
    CY_Out, Y_N)
Process
    variable error_count: Integer := 0; -- Initial value of counts
    of error is 0
begin
    If (Sub = '0') then
        A <= "11"; B <= "11"; -- Test for first case addition A and B
        both 11
        Wait for 8 ns; -- Assume AdderCumSub entity gate delays < 8 ns
        If (CY_Out /= '1' or Y_N /= "10") and then
            error_count := error_count + 1; -- Increment counts of
            error
        end If;
        Assert (CY_Out = '1') Report "Error in CY_Out" Severity Error;
        Assert (Y_N = "10") Report "Error in Y_N" Severity Error;
        end If

        If (Sub = '1') then
            A <= "11"; B <= "11"; -- Test for first case subtraction A and B
            both 11
            Wait for 8 ns; -- Assume AdderCumSub entity gate delays < 8 ns
            If (CY_Out /= '0' or Y_N /= "00") and then
                error_count := error_count + 1; -- Increment counts of
                error
            end If;
            Assert (CY_Out = '0') Report "Error in CY_Out" Severity Error;
            Assert (Y_N = "00") Report "Error in Y_N" Severity Error;
            end If
    ;-----
    If (Sub = '0') then
        A <= "01"; B <= "10"; -- Test for second case addition A and
        B = 01 and 10
        Wait for 8 ns; -- Assume AdderCumSub entity gate delays < 8 ns
        If (CY_Out /= '0' or Y_N /= "11") then
            error_count := error_count + 1; -- Increment counts of
            error
        end If;
        Assert (CY_Out = '0') Report "Error in CY_Out" Severity Error;
        Assert (Y_N = "11") Report "Error in Y_N" Severity Error;
    end If;
end process;

```

```
If (Sub = '1') then
    A <= "01"; B <= "10"; -- Test for second case subtraction A and
    B = 01 and 10
    Wait for 8 ns; -- Assume AdderCumSub entity gate delays < 8 ns
    If (CY_Out /= '1' or Y_N /= "11") then
        error_count := error_count + 1; -- Increment counts of
        error
    end If;
    Assert (CY_Out = '1') Report "Error in CY_Out" Severity Error;
    Assert (Y_N = "11") Report "Error in Y_N" Severity Error;
    -----
    Wait; -- Wait for next commands
end Process;
end RTLTestBTwobit_AddSub;
-----
Configuration Config_RTLTestBTwobit_AddSub of TestBTwobit_
AdderCumSub is
    for RTLTestBTwobit_AddSub
        end for;
end Config_RTLTestBTwobit_AddSub;
-----
```

Example 18.7 How will you design down counter?

Solution

Assume Count_Enable when '1' enables counter to decrease on +ve edge. \underline{Q} is array with n elements from $n - 1$ downto 0 and logic of each element is 0 or 1.

An subtraction operator does the arithmetic subtraction of 1 from vector 0^{th} , 1^{st} , 2^{nd} ... up to $(n - 1)^{\text{th}}$ output vector elements of \underline{Q} . In VHDL RTL entity Nbit_Counter can be described as follows. We use arithmetic addition, '-' operator. We can describe RTL design codes for an entity Nbit_Counter.

Example 18.8 Nbit_DCounter is an n -bit down-counter circuit, which on a +ve edge clock decrements counts value at \underline{Q} provided an input Count_Enable is active. A reset input resets the counter. When next decrement input is applied beyond minimum possible value of Q (all elements of vector Q are 0s) then also the counter resets to all element of $\underline{Q} = 1$ s. How will you model RTL design codes for the entity Nbit_DCounter and architecture? Use behavioural model.

Solution

```
Library IEEE; -- VHDL using IEEE standard libraries
use IEEE.Std_Logic_1164.all      ; -- import Std_Logic from the IEEE Library
use IEEE.Std_Logic_unsigned.all ; -- Std_Logic functions from the IEEE
                                Library
                                ; -- for the unsigned arithmetic operations
```

```

use IEEE . Std_Logic_arith.all      ; -- Std_Logic from the IEEE Library
                                         ; -- Std_Logic_arith functions enable use of
                                         ; addition operator +, &, and other arithmetic
                                         ; operators
generic (n:Natural) ;-- declare constant n, a natural number
                     ;-- n is the number of bits decrements
entity Nbit_DCounter is
  port (Count_Enable, Reset0, SeqClk: in Std_Logic;
        Q: out Std_Logic_Vector (n-1 downto 0), );
end Nbit_DCounter;

architecture RTLBehaviourNbit_DCounter of Nbit_DCounter is
Signal Present_Q: Std_Logic_Vector (n downto 0); -- Present_Q is present
value of Counts Q at output
begin
  Process (Count_Enable, Reset0, SeqClk)
  begin
    If (Reset0 = '1') then
      Present_Q <= Present_Q or (not Present_Q) -- Present_Q =
      All 1s
    elsif (SeqClk = '1' and SeqClk'event) then
      If (Count_Enable = '1') then Present_Q <= Present_Q - 1;
      -- Present_Q decrements
      end If;
    end If;
  end Process
  Q<= Present_Q; -- Signal Present_Q concurrently assigns to Q
end RTLBehaviourNbit_DCounter;

```

Example 18.9

Show use of subtraction operator '-' fpr decrement in the counter.

Solution

Signal Present_Q is ('0', '1', '0', '0', '0', '1', '0', '0')

Subtraction Present_Q - 1 gives ('0', '1', '0', '0', '0', '0', '1', '1').

Example 18.10

How will you design test for a down counter?

Solution

Test bench generates stimulus for the simulation of the behaviour of an entity and show on screen or otherwise the situations and actions, which will take place in the entity on synthesis from the logic gates.

1. When stimulus resets two-bit down Counter on Reset0 = '1', then Signal Present_Q = 11 (decimal 3).

2. When Count_Enable = 1 and a positive edge is input when SeqClk = '1' and SeqClk'event is true then Present_Q becomes 10 (decimal 2).
3. When Count_Enable = 1 and next positive edge input occurs when SeqClk = '1' and SeqClk'event is true then Present_Q becomes 10 (decimal 1).
4. When Count_Enable = 1 and next positive edge input occurs when SeqClk = '1' and SeqClk'event is true then Present_Q becomes 11 (decimal 0).
5. When Count_Enable = 1 and next positive edge input occurs when SeqClk = '1' and SeqClk'event is true then Present_Q becomes 00 (decimal 3).

The stimulus of a test bench sends the signals for all possible situations and counts the number of errors occurred. If the stimulus report error counts = 0, then the entity behaviour stands tested.

Example 18.11 How can you describe RTL design codes for a *test bench* for entity Nbit_DCounter and architecture in behavioural model? Select constant $n = 2$. Test possible cases. Assume down counter.

Solution

Length of Present_Q is 2.

Library IEEE; -- VHDL using IEEE standard libraries

```
use IEEE.Std_Logic_1164.all      ; -- import Std_Logic from the IEEE Library
use IEEE.Std_Logic_unsigned.all  ; -- Std_Logic functions from the IEEE Library
                                ; -- for the unsigned arithmetic operations
use IEEE.Std_Logic_arith.all     ; -- Std_Logic from the IEEE Library
                                ; -- Std_Logic_arith functions enable use of
                                ; addition operator +, & and other arithmetic
                                ; operators

generic (n:Natural)
entity TestBTwobit_DCounter is
generic (n:Natural:=2)          ; -- declare constant n = 2, a natural number
                                ; -- n is the number of bits subtracted by 1 at the
                                ; counter
end TestBTwobit_DCounter; -- Declare an entity for the test bench for n-bit
                           ; counter entity
architecture RTLTestBTwobit_DCount of TestBTwobit_DCounter
is

component Twobit_DCount is
    ; -- Remember n = 2 in entity TestBTwobit_DCounter
    port(Count_Enable, Reset0, SeqClk: in Std_Logic;
         Q: out Std_Logic_Vector(n - 1 downto 0), );
end component Twobit_DCount;
```

```

Signal Count_Enable, Reset0, SeqClk: Std_Logic;
Signal Present_Q: Std_Logic_Vector (n - 1 downto 0);
begin
    TestBTTwo_DCOUNTER: Twobit_Count port map (Count_Enable,
        Reset0, SeqClk, Present_Q)
Process
begin
    begin
        SeqClk <= '0'; Wait for 4 ns;
        SeqClk <= '1'; Wait for 4 ns;
    end Process; -- Declare Tclk (clock cycle time) = 8 ns
    Process
        variable error_count: Integer := 0; -- Initial error_count reset to 0
    begin
        Reset0 <= '1'; -- Active reset for initial condition Present_Q = 0
        Count_Enable <= '1'; -- Enable Counting
        Wait for 16 ns; -- wait for interval 2 × Tclk
        Reset0 <= '0'; -- Inactive reset Initial Counts increment enable
        ;-----
        Wait for 8 ns; -- wait for interval Tclk for case 1
        If (Present_Q /= 2) then error_count := error_cnt+1;
        end If;
        Assert (Present_Q = 2) Report "Test for Down Counter Decrement to
        2 Unsuccessful" Severity Error;
        ;-----
        Wait for 8 ns; -- wait for interval Tclk for case 2
        If (Present_Q /= 1) then error_count := error_cnt+1;
        end If;
        Assert (Present_Q = 1) Report "Test for Down Counter decrement to
        1 Unsuccessful" Severity Error;
        ;-----
        Wait for 8 ns; -- wait for interval Tclk for case 3
        If (Present_Q /= 0) then error_count := error_cnt+1;
        end If;
        Assert (Present_Q = 0) Report "Test for Down Counter decrement to
        1 Unsuccessful" Severity Error;
        ;-----
        Wait for 8 ns; -- wait for interval Tclk for case 4
        If (Present_Q /= 3) then error_count := error_cnt+1;
        end If;
        Assert (Present_Q = 3) Report "Test for Down Counter 0 after 3
        counts Unsucc.l" Severity Error;
        ;-----
        Wait for 16 ns; --wait for interval Tclk for case 5
        Reset0 <= '1';
        Wait 8 ns
        If (Present_Q /= 3) then error_count := error_cnt+1;

```

```
        end If;
Assert (Present_Q =3) Report "Test for Down Counter Reset on
Reset0 = 1 Unsuccessful" Severity Error;
;-----
If (error_count = 0) then Assert false
    Report "Testbench 2-bit Down Counter completed successfully"
    Severity note; -- Test result at count value
else Assert true
    Report "Something wrong, try again"
    Severity Error; "Testbench 2-bit Down Counter found unsuccessful"
    end If;
    Wait; -- Wait for next commands
    end Process;
end RTLTestBTwobit_DCount;
Configuration Config_RTLTestBTwobit_Count of TestBTwobit_
DCounter is
    for RTLTestBTwobit_DCount
    end for;
end Config_RTLTestBTwobit_DCount;
```

Example 18.12 How will you model RTL design entity and its behaviour for **Register0** consisting of m D-FF, which transfers logic vector \underline{A} to output logic vector \underline{Q} on +ve clock edge? Q_1 is complementary logic state of \underline{Q} .

RTL entity Reg0 can be described as follows in VHDL.

Solution

```
Library IEEE; -- VHDL using IEEE standard libraries
use IEEE.Std_Logic_1164.all ; -- import Std_Logic from the IEEE Library
use work.all;
entity Reg0 is
    port ( $\underline{A}$ : in Std_Logic_Vector ( $m - 1$  down to 0);
           $\underline{Q}$ ,  $Q_1$ : out Std_Logic_Vector ( $m - 1$  down to 0);
          SeqClk: in Std_Logic;
    end Reg0;
architecture RTLBehaviourReg0 of Reg0 is
begin
    Process
        If (SeqClk = '1' and SeqClk'event) then
             $\underline{Q} \leqslant \underline{A}$ ; -- +ve edge event assigns  $\underline{Q}$  state from  $\underline{A}$  logic state.
        end Process
         $Q_1 \leqslant \text{not } \underline{Q}$ ; -- Signal Q1 concurrently assigns to not of  $\underline{Q}$ 
    end Process;
    RTLBehaviourReg0;
```

Example 18.13 How will you test RTL design entity and its behaviour for Reg0 consisting of m D-FF which transfers logic vector \underline{A} to output logic vector \underline{Q} on +ve clock edge?

Solution

Test bench generates stimulus for the simulation of the behaviour of an entity and show on screen or otherwise the situations and actions, which will take place in the entity on synthesis from the logic gates.

1. When SeqClock –ve edge event occurs then if logic vector $\underline{A} = 0$ then \underline{Q} is same as previous \underline{Q} and $\underline{Q1}$ is same as previous \underline{Q} .
2. When SeqClock –ve edge event occurs then if $\underline{A} = 1$ then \underline{Q} is same as previous \underline{Q} and $\underline{Q1}$ is same as previous \underline{Q} .
3. When SeqClock = ‘0’ then if $\underline{A} = 0$ then \underline{Q} is same as previous \underline{Q} and $\underline{Q1}$ is same as previous \underline{Q} .
4. When SeqClock = ‘0’ then if $\underline{A} = 1$ then \underline{Q} is same as previous \underline{Q} and $\underline{Q1}$ is same as previous \underline{Q} .
5. When SeqClock = ‘1’ then if $\underline{A} = 0$ then \underline{Q} is same as previous \underline{Q} and $\underline{Q1}$ is same as previous \underline{Q} .
6. When SeqClock = ‘1’ then if $\underline{A} = 1$ then \underline{Q} is same as previous \underline{Q} and $\underline{Q1}$ is same as previous \underline{Q} .
7. When SeqClock +ve edge event occurs then if $\underline{A} = 0$ at that instance then after delay $\underline{Q} = 0$ and $\underline{Q1} = 1$.
8. When SeqClock +ve edge event occurs then if $\underline{A} = 1$ at that instance then after delay $\underline{Q} = 1$ and $\underline{Q1} = 0$.

The above situations can be tested as follows:

Consider a Process by Tclk is set as 8 ns. (Tclk = interval between successive +ve edge transitions, 0.5 times Tclk = interval between successive –ve edge transitions, 0.25 times Tclk = interval between successive +ve edge transitions and middle of period for 1 or interval between successive +ve edge transitions and middle of period for 1).

Case 1: Test after Tclk (= 8 ns)

Assume that the previous value of $\underline{A} = 0$ and \underline{Q} is also 0 or $\underline{A} = 1$ and \underline{Q} is also 1. Whatever may be the case, if \underline{A} sets to 1 then a +ve edge must occur or must have occurred during interval, Tclk. If we test \underline{Q} after Tclk, then error should be reported if it is found 0, not 1.

Case 2: Test after 1.25 Tclk (= 10 ns)

Assume that the previous value of $\underline{A} = 0$ and \underline{Q} is also 0 or $\underline{A} = 1$ and \underline{Q} is also 1. Whatever may be the \underline{A} or \underline{Q} , if \underline{A} resets to 0 now then a +ve edge must have occurred during interval, Tclk. If we test \underline{Q} after 1.25 times Tclk then error should be reported if it is found 1, not 0.

Case 3: Test after 2 times Tclk (= 16 ns)

Assume that the previous value of $\underline{A} = 0$ and \underline{Q} is also 0 or $\underline{A} = 1$ and \underline{Q} is also 1. Whatever may be \underline{A} or \underline{Q} , if \underline{A} sets to 1 now then a second +ve edge must have occurred and second –ve edge must have occurred during interval = 2.Tclk. \underline{Q} should remain 1 after 2 Tclk. If we test \underline{Q} after 2 times Tclk, then error should be reported if it is found 0, not 1.

Case 4: Test after 0.25 Tclk (= 2 ns)

Assume that the previous value of $\underline{A} = 1$ and \underline{Q} is also 1. When \underline{A} resets to 0, then \underline{Q} should still be 1 because only after next +ve edge. Next positive edge can occur at least after 0.5 Tclk. If we test \underline{Q} after 0.25 Tclk too short interval, then error should be reported if it is found 0.

Case 5: Test after 2.75 times Tclk (= 22 ns)

Assume that the previous value of $\underline{A} = 0$ and \underline{Q} is also 0 or $\underline{A} = 1$ and \underline{Q} is also 1. Whatever may be \underline{A} or \underline{Q} , if \underline{A} resets to 1 now, then a second +ve edge and two -ve edges must have also occurred during interval = 2.75 times Tclk. \underline{Q} should remain 1 after 2.75.Tclk also. If we test \underline{Q} after 2.75 times Tclk then error should be reported if it is found 1, not 0.

The stimulus of a test bench sends the signals for all possible situations and counts the number of errors occurred. If the stimulus report error counts = 0, then the entity behaviour stands tested.

Example 18.14

How can you describe RTL design codes for a *test bench* for entity Reg0 and architecture in behavioural model? Test possible cases.

Solution

```
entity TestBReg0 is
end TestBReg0; -- Declare an entity for the test bench for flip-flop entity
architecture RTLTestBReg0 of TestBReg0 is
generic (m: Natural:=8) ; -- declare constant m = 8 , a natural number
                           ; -- n is the number of bits store in 8-bit register circuit
                           of D-FFs
Signal  $\underline{A}$ : in Std_Logic_Vector (m - 1 downto 0);
Signal SeqClk: in Std_Logic;
Signal  $\underline{Q}$ ,  $\underline{Q1}$ : out Std_Logic);
Component Reg001 is; -- Reg0 component
port ( $\underline{A}$ : in Std_Logic_Vector (m - 1 down to 0);
       $\underline{Q}$ ,  $\underline{Q1}$ : out Std_Logic_Vector (m - 1 down to 0);
      SeqClk: in Std_Logic;
end component Reg001;
begin
  Test_Reg0: port map ( $\underline{A}$ , SeqClk,  $\underline{Q}$ ,  $\underline{Q1}$ );
  Process
    begin
      SeqClk <= '0'; Wait for 4 ms;
      SeqClk <= '1'; Wait for 4 ms;
    end Process ;-- clock +ve edge, '1', -ve edge, '0' and +ve edge,
                  periodically
                  ;--occurs after 8 ms
  Process
    variable error_count: Integer:=0;
```

```

begin
  A <= "10000000"; Wait 8 ns; -- wait for period 1 interval = Tclk
  If (Q!="10000000") then error_count := error_count + 1; -- error Case 1 If
    Q differs
      ; -- and does not become '1'
  end If;
  Assert (Q = "10000000") Report "Q changed and does not show '1' after
  Tclk" Severity Error;
  ;-----
  A <= "10110000"; Wait 10 ns;-- wait for period 1.25 interval = Tclk
  If (Q!="10110000") then error_count := error_count + 1; -- error Case 2 If
    Q changed
      ; -- and does not remain '0'
  end If;
  Assert (Q = "10110000") Report "Q changed and not found '0' after 1.25
  Tclk" Severity Error;
  ;-----
  A <= "01001111"; Wait 16 ns;-- wait for period 2 intervals of Tclk
  If (Q!="01001111") then error_count := error_count + 1; -- error Case 3 If
    Q changed
      ; -- and does not remain '1'
  end If;
  Assert (Q = "01001111") Report "Q changed and does not show '1'
  after 2.Tclk" Severity Error;
  ;-----
  A <= "0000000"; Wait 2 ns; -- wait for period 0.25 interval = Tclk
  If (Q>= "00000000") then error_count := error_count + 1 ; -- error Case 4
    If Q found
      changed
        ; even before the next +ve edge
  end If;
  Assert (Q = "00000000") Report "Q found changed even before next +ve
  edge"
  Severity Error;
  ;-----
  A <= ="01001111"; Wait 22 ns; -- wait for period 2.75 intervals of Tclk
  If (Q=="01001111") then error_count := error_count + 1; -- error Case 5 If
    Q changed
      ; --and does not remain '1'
  end If;
  Assert (Q == "01001111") Report "Q changed and does not show '1' after
  2.75.Tclk" Severity Error;
  ;-----
  If (error_count = 0) then Assert false
    Report "Testbench 2-bit Counter completed successfully"
    Severity note; -- Test results at error count value 0

```

```

        else Assert true
            Report "Errors found in D-FF behaviour"
            Severity Error; "Testbench D-FF found unsuccessful"
        end If;
        Wait; -- Wait for next commands
    end Process;
end RTLTestBReg0;
-----
Configuration Config_RTLTestBReg0 of TestBReg0 is
    for RTLTestBReg0
    end for;
end Config_RTLTestBReg0;
-----

```

Example 18.15 Consider a left shift register of 4-bits. How do you design sequential circuit for it?

Solution

A finite state machine (FSM) means that is a circuit, which generates next state output depending on its present state. A sequential circuit made up from sequential memory section and combinational circuit can function as an FSM. A shift register generates the states such that on each sequential clock event the element at position 0 is reset to 0 and elements at position i changes to $i + 1$ for $i = 0, 1$ and 2 .

$$Q_0 \leq 0; Q_1 \leq Q_0; Q_2 \leq Q_1; Q_3 \leq Q_2;$$

Figure 18.2 shows a sequential circuit entity FSMSLSeqCircuit for four-bit shift register. There are two Processes. The Processes at SeqM and Circuit_C execute concurrently.

1. Memroy section SeQM may consist of 4-Dflip-flops. A state transition from Y_i to Q takes place at SeqM after an interval, tSeqM after an SeQClk event. The Output states \underline{Q} are given as inputs \underline{Q}_i to Circuit_C.
2. Circuit _C generates output Y after an interval = tComCircuit such that $Y_0 \leq 0; Y_1 \leq Q_0; Y_2 \leq Q_1; Y_3 \leq Q_2;$
3. Minimum interval for generating new Y_i = tSeqplusComCircuit = Time delay at SeqM plus in Circuit_C = tSeqM + tComCircuit.
4. A new input Y_i enables a next sequence of the state at Q after interval tSeqM from event at SeqClk and $Q_0 \leq 0; Q_1 \leq Q_0; Q_2 \leq Q_1; Q_3 \leq Q_2;$
5. An input-output port is Reset0. An input event at Reset0 resets the state output \underline{Q} to an initial state after tSeqM.

The sequential circuits FSMSLSeqCircuit two concurrent Processes have following inputs and outputs in RTL design model.

1. One port for SeqClk logic input at SLSqm (memory section).
2. One input-output port \underline{Y}_i is at SeqM from Circuit_C
3. An input port Reset0. This is for resetting the state to an initial state at \underline{Q} .

4. \underline{Y}_1 input of SLSeqM also interconnects to output port \underline{Y} so that a next state can be generated by SeqM. $\underline{Y}_1 = \underline{Y}$. (\underline{Y} is output of Circuit_C)
5. SLSeqM (memory section) output is logic vector \underline{Q} .
6. logic vector \underline{Q}_1 also interconnects to output port logic vector \underline{Q} . ($\underline{Q}_1 \leq \underline{Q}$)
7. Circuit_S inputs are logic vector at port \underline{Q}_1 and logic input at port Shift_Enable.

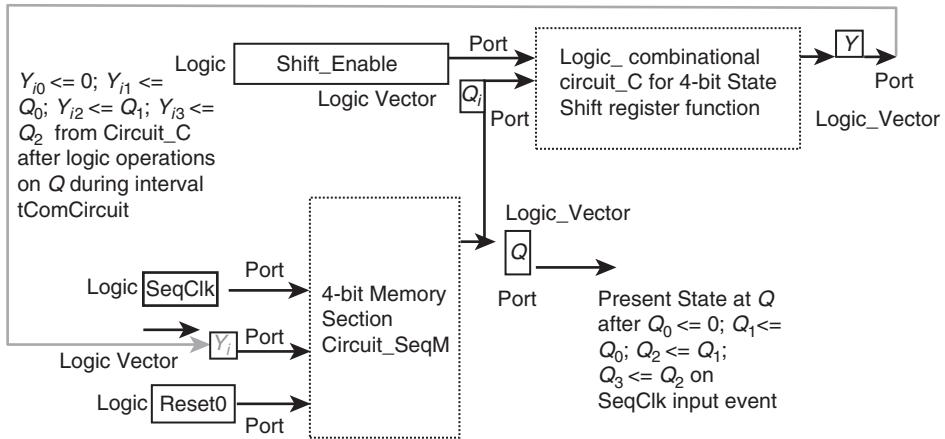


FIGURE 18.2 FSM sequential circuit for shift register consisting of FSM_StateSLSeqM and the Output states \underline{Q} are given as inputs \underline{Y}_1 after a Logic_SLFunctrionalCircuit_S generates output \underline{Y} for next sequence of the state.

Example 18.16 How can you describe FSMSLSeqCircuit entity and components (memory section and combinational circuit)?

Solution

The entity can be described follows:

```

Library ieee;
use ieee.Std_Loic_1164.all;
Tclk, Tdelay: Time;
Tclk:= 50 ns; -- Interval T of Clock inputs
Tdelay:= 4 ns;
entity FSMSLSeqCircuit is
  n, m, i, j: Positive;
  ; -- Sequential circuit entity has two components (memory section and
  -- combinational circuit).
  Port (Yi: inout Std_Loic_Vector (k - 1 downto 0);
        ShiftLClk: in Std_Loic;
        Reset0: in Std_Loic;
        Q: Out Std_Loic_Vector (k - 1 downto 0); --for example, k = 4 for a
        -- 4-bit counter.
  ; -- Logic_CombinationalCircuit_C

```

```

Qi: inout Std_Logic_Vector (k - 1 downto 0);
shift_enable: in Std_Logic;
Y: out Std_Logic_Vector (k - 1 downto 0);
end entity FSMSLSeqCircuit;

```

Example 18.17

How can you describe architecture RTLFSMSLSeqC of FSMSLSeqCircuit? Assume FSMSL generates four states, S_0, S_1, S_2 and S_3 . These are four finite states. Next state to $S_0 \Rightarrow S_0$ (1 to 3) & '0', $S_1 \Rightarrow S_1$ (1 to 3) & '0', $S_2 \Rightarrow S_2$ (1 to 3) & '0' and $S_3 \Rightarrow S_3$ (1 to 3) & '0'. All 0s on successive four events at sequencing input clock. Assume +ve edge at ShiftLClk causing state transition of Y_i to Q at memory section (flip-flops section).

Solution

```

architecture RTLFSMSLSeqM of FSMSLSeqCircuit is
    tSeqM, tSeqplusComCircuit, tComCircuit: Time
    tSeqM: = 20 ns; --Delay in sequential circuit memory section
    tComCircuit: = 10 ns; -- Delay in combinational logic section in sequential
    circuit
    tSeqplusComCircuit := tSeqM+ tComCircuit; -- Delay of Sequential Circuit
    both sections;
    ; -- Present state at  $Q$  and next state at  $Q'$  is in one of the states,
    type Q_Type is (S0, S1, S2, S3);
    Signal S0: Std_Logic_Vector (k-1 down to 0);
    Signal S1: Std_Logic_Vector (k-1 down to 0);
    Signal S2: Std_Logic_Vector (k-1 down to 0);
    Signal S3: Std_Logic_Vector (k-1 down to 0);
    generic (k : Natural:=4); -- declare constant k = 4 , a natural number
    Signal Next_State, Present_state: Q_type;
    Signal Present_State: Q_Type: = S0;
    Signal Next_State: Q_Type: = S1;
    Process (Q, Present_State)
        ; -- Define the assignments with Present state input wired connection to  $Q$ 
        begin
            Q<= PresentState' Delayed(0); -- Interconnect Y output to  $Y_i$  input
    end Process;
    Process (Q, Qi); -- Define the assignments with  $Qi$  as input from connection
                      to  $Q$ 
        begin
            Qi<= Q' Delayed (0); -- Interconnect  $Qi$  input to  $Q$  output
    end Process;
    Process (Y, Next_State)
        ; -- Define the assignments with Present state input  $Y_i$  wired
           connection to  $Y$ 
        begin
            Y<= NextState' Delayed (0); -- Interconnect Y output to  $Y_i$  input
    end Process;

```

```

Process (Yi , Y) -- Define the assignments with Yi as input from connection
to Y
begin
    Yi<= Y' Delayed (0); -- Interconnect Y output to Yi input
end Process;
Process (ShiftLClk, Yi, Reset0, Present_State,
Next_State);
    ; -- ShiftLClk, Yi, Reset0 are inputs nd Present_State is output
begin
    ; -- Process body for SeqM
If (Reset0= '1') then Present_State <= S0 after tSeqM
elsif (ShiftLClk' event and ShiftLClk = '1') then
Present_State <= Next_State; ; -- ShiftLClk event results in state
transition of Yi to Q to next state.
end If
end Process
Process (shift_enable,Next_State,Qi)
    ; -- Process body for Logic_CombinationalCircuit_C
begin
If (Qi = S0 and shift_enable =0) then Next_State <= S0
elsif (Qi = S0 and shift_enable ='1' then Next_State <= S0
(1 to k - 1) & '0';
end if;
-----
If (Qi = S1 and shift_enable =0) then Next_State <= S1
elsif (Qi = S1 and shift_enable ='1' then Next_State <= S1
(1 to k - 1) & '0';
end if;
-----
If (Qi = S2 and shift_enable =0) then Next_State <= S2
elsif (Qi = S2 and shift_enable ='1' then Next_State <= S2
(1 to k - 1) & '0';
end if;
-----
If (Qi = S3 and shift_enable = 0) then Next_State <= S3
elsif (Qi = S3 and shift_enable ='1' then
Next_State <= S3 (0 to k - 1) - S3(0 to k - 1)) ; -- All '0s' after k
shifts.
end if;
end Process;
end RTLFSMSLSeqC

```

Example 18.18

Test bench generates stimulus for the simulation of the behaviour of an entity and show on screen or otherwise the situations and actions which will take place in the

entity on synthesis from the logic gates. Assume an FSMSL for state transitions between S_0, S_1, S_2 and S_4 states. Assume $S_0 = S_0$ (1 to $k - 1$) & '0' operation takes place for each state. Firstly following is tested.

1. When $\text{Reset}_0 = 1$ then $\underline{Q} = ('0', '0', '0', '0')$ after 2 T_{clk} where T_{clk} is sequencing clock period.
2. When $\text{Reset}_0 = 0$ then \underline{Q} remains same as previous state at Q .
3. When $\text{Reset}_0 = 0$ then $Q = ('0', '0', '0', '0')$ after 2 T_{clk} where T_{clk} is sequencing clock period.
4. Then $\text{Reset}_0 \leq 0$ and the following sequential actions are tested.
5. When SeqClock +ve edge event occurs then FSMSL memory section of FFs state transfers to Q . Q becomes Q (1 to $k - 1$) & '0'.
6. When SeqClock +ve edge event occurs then FSMSL memory section of FFs state transfers to Q . Q becomes Q (1 to $k - 1$) & '0'.
7. When SeqClock +ve edge event occurs then FSMSL memory section of FFs state transfers to Q . Q becomes Q (1 to $k - 1$) & '0'
8. When SeqClock +ve edge event occurs then FSMSL memory section of FFs state transfers to Q . Q becomes $Q - Q$, which means all $Q_s = '0'$.

State machine keep generating next states on each sequencing clock event.

The stimulus of a test bench sends the signals for all possible situations and counts the number of errors occurred. If the stimulus report error counts = 0, then the entity behaviour stands tested.

Example 18.19

How can you describe RTL design codes for a *test bench* for entity FSMSL for state transitions to S_0, S_1, S_2 and S_3 ? Describe **architecture** in behavioural for test bench? Test possible cases.

Solution

```
Library ieee;
Use ieee.Std_Logic_1164.all;
entity TestBFMSLS0_S201 is
end TestBFMSLS0_S2;-- Declare an entity for the test bench for flip-flop entity
architecture RTLTestBFMSLS0_S2 of TestBFMSLS0_S201 is
    Tclk, tSeqM, tSeqplusComCircuit, tComCircuit: Time
    Tclk:=50 ns;
    tSeqM:= 20 ns; --Delay in sequential circuit memory section
    tComCircuit:= 10 ns; -- Delay in combinational logic section in sequential
                        circuit
    tSeqplusComCircuit := tSeqM+ tComCircuit; -- Delay of Sequential Circuit
                                                both sections;
    generic (k : Natural:=2); -- declare constant k = 2 , a natural number
    Signal S0 : Std_Logic_Vector (k-1 down to 0)
    Signal S1 : Std_Logic_Vector (k-1 down to 0);
    Signal S2 : Std_Logic_Vector (k-1 down to 0);
```

```

Signal S3 : Std_Logic_Vector ( $k - 1$  down to 0);
begin
  Component TestFSMSLS0_S2 is
    n, m, i, j: Positive;
    ; -- Sequential circuit entity has two components (memory section and
       combinational circuit).
  Port (
    SeqClk: in Std_Logic;
    Reset0: in Std_Logic;
    Q: Out Std_Logic_Vector ( $k - 1$  downto 0);
    ; -- Logic_CombinationalCircuit_C
    transition_enable: in Std_Logic;
  end Component TestFSMSLS0_S2;
  begin
    Test_FSMSLS0S2: port map (Q, SeqClk, Reset0);
    Process
      begin
        SeqClk <= '0'; Wait for 25 ms;
        SeqClk <= '1'; Wait for 25 ms;
      end Process ; -- clock +ve edge, '1' after 4 ms from -ve edge and
                    -- -ve edge after 4 ms,
                    ; -- periodically. Tclk = 8 ms
    Process
      variable error_count: Integer := 0;
      Signal S: Std_Logic_Vector ( $k - 1$  down to 0):
      begin
        Reset0 <= '1'; Wait 80 ms;
        Reset0 <= '0'; Wait 30 ms; Wait for at least 0.6 Tclk
        S0 = Q(1 to  $k - 1$ ) & '0';
        If (Q /= S0) then error_count := error_count + 1; -- error Case 0
        Reset0 does not reset Q
      end If;
      Assert (Q = '0') "Shift transition on k-shifts unsuccessful in Tclk"
      Severity Error; -----
      Wait 60 ns; Wait for 1.2 Tclk
      S1 = Q(1 to  $k - 1$ ) & '0';
      If (Q /= S1) then error_count := error_count + 1; -- error Case 1
      end If;
      Assert (Q = S1) "Shift transition on k-shifts unsuccessful in Tclk"
      Severity Error;
      -----
      Wait 50 ms; Wait for Tclk
      S2 = Q(1 to  $k - 1$ ) & '0';
      If (Q /= S2) then error_count := error_count + 1; -- error Case 2
      end If;
    end;
  end;
end;

```

```
Assert ( $Q = S2$ ) Report “Shift transition on k-shifts unsuccessful in
Tclk” Severity Error;
;-----
Wait 50 ms; Wait for Tclk
S3=  $Q(1 \text{ to } k-1) \& '0'$ ;
If ( $Q=S3$ ) then error_count: = error_count + 1; -- error Case 3
end If;
Assert ( $Q = S3$ ) Report “Shift transition on k-shifts unsuccessful in
Tclk”
Severity Error;
;-----
Wait 50 ns; Wait for Tclk
S0 =  $Q(1 \text{ to } k-1) \& '0'$ ;
If ( $Q=S0$ ) then error_count: = error_count + 1; -- error Case 4:
end If;
Assert ( $Q = S0$ ) “Shift transitions on  $k$ -shifts unsuccessful in Tclk
also” Severity Error;
;-----
If(error_count = 0) then Assert false
Report “Testbench FSMSLS0_S2 completed successfully”
Severity note; -- Test results at error count value 0
else Assert true
Report “Errors found in JK-FF behaviour”
Severity Error; “Testbench 2-bit Counter found unsuccessful”
end If;
Wait; -- Wait for next commands
end Process;
end RTLTestBFMSLS0_S2;
;-----
Configuration Config_RTLTestBFMSLS0_S2 of TestBFMSLS0_S201is
  for RTLTestBFMSLS0_S2
  end for;
end Config_RTLTestBFMSLS0_S2;
;-----
```

Example 18.20 How will you model RTL design entity for a 4 to 1 multiplexer (MUX41)? State Y is according to the input Select_in for selecting the addressed channel.

Solution

```
Library IEEE; -- VHDL using IEEE standard libraries
use IEEE . Std_Logic_1164.all; -- import Std_Logic from the IEEE Library
generic (n, m, k:Natural) ; -- declare constants n and m two natural numbers
; -- n is the number of elements in inputs A in
  MUXnto1
; -- m is the number of elements in A_Select
```

```

; 4 to 1 multiplexer (MUX41) has n = 4 and
k = 2 . 8 to 1

n:=4;
m:=2;
k:=1;
entity MUX_4to1 is
port (A: in Std_Logic_Vector (n - 1 downto 0);
      B: in Std_Logic_Vector (n - 1 downto 0);
      Select_in: in Std_Logic_Vector (m - 1 downto 0); -- m = 1 for two
      input logic vectors
      Y: out in Std_Logic_Vector (n - 1 downto 0));
end MUX_4to1;
architecture RTLMUX_4to1 of MUX_4to1 is
begin
Process (A,B,Select_in)
begin
If(A_Select = "0") then Y<= A
elsif(Select_in = "1") then Y<= B;
Process
end if;
end Process;
end RTLMUX_4to1;
end MUX_4to1;

```

Example 18.21 How will you model RTL design behaviour for a 4 to 1 multiplexer (MUX41)? State Y is according to the input Select_in for selecting the addressed channel.

Solution

```

architecture RTLMUX_4to1 of MUX_4to1 is
begin
Process (A,B,C,D,Select_in)
begin
If(A_Select = "0") then Y<= A
elsif(Select_in = "1") then Y<= B;
Process
end if;
end Process;
end RTLMUX_4to1;

```

Example 18.22 How will you model RTL design entity and its behaviour for a 1 to 4 demultiplexer (DeMUX14)? Y_0 or Y_1 is assigned to State A is according to Address Select output at Select_outport.

Solution

Library IEEE; -- VHDL using IEEE standard libraries
use IEEE . Std_Logic_1164 . all; -- import Std_Logic from the IEEE Library

```
generic (n, m, k:Natural) ; -- declare constants n and m two natural numbers
                           ; -- n is the number of elements in outputs A in
                           ; -- m is the number of elements in Y_Select
entity DeMUX_4to1 is
  generic (n: Natural:= 2)
  generic (m: Natural:= 1, m:Natural:=1);
  generic (k: Natural:= 1, k:Natural:=1);
  port (Y0: out Std_Logic_Vector (n - 1 downto 0);
        Y1: out Std_Logic_Vector (n - 1 downto 0);
        Y2: out Std_Logic_Vector (n - 1 downto 0);
        Y3: out Std_Logic_Vector (n - 1 downto 0);
        Select_out: in Std_Logic_Vector (m - 1 downto 0); -- m = 1 for two
        output logic vectors
        A: in Std_Logic_Vector (k - 1 downto 0);
  end DeMUX_4to1;
architecture RTLDeMUX_4to1 of DeMUX_2to1 is
begin
  Process (Y0,Y1, A, Select_out)
    begin
      If(Select_out = "00") then Y0<= A
      elsif(Select_out = "01") then Y1<= A
      elsif(Select_out = "10") then Y2<= A
      elsif(Select_out = "10") then Y3<= A;
      Process
    end if;
  end Process;
end RTLDeMUX_4to1;
```

Example 18.23 How will you model Test bench stimulus for the simulation of the behaviour of an entity for demultiplexer 4 to 1 and show on screen or otherwise the situations and actions which will take place in the entity on synthesis from the logic gates?

Solution

When stimulus output Select_out = "0", then Signal Y0 = A.

When stimulus output Select_out = "1", then Signal Y1 = A.

When stimulus output Select_out = "0", then Signal Y2 = A.

When stimulus output Select_out = "1", then Signal Y3 = A.

The stimulus of a test bench sends the signals for all possible situations and counts the number of errors occurred. If the stimulus report error counts = 0, then the entity behaviour stands tested.

Example 18.24 How can you describe RTL design codes for a *test bench* for entity DeMUX_4to1 and **architecture** in behavioural model? Select constant $n = 2$ and $m = 2$. Test possible cases.

Solution

Length of A is 2.

```

Library IEEE; -- VHDL using IEEE standard libraries
use IEEE . Std_Logic_1164.all; -- import Std_Logic from the IEEE Library
                                ; addition operator +, & and other arithmetic
                                operators

generic (n:Natural)
entity TestBDeMUX_1to4 is
end TestBDeMUX_4to1; -- Declare an entity for the test bench for n-bit counter
entity
architecture RTLTestBDeMUX1_2 of TestBDeMUX_2to1 is
generic (n :Natural:=2) ; -- declare constant n = 2, a natural number
                        ; -- n is the number of bits added by arithmetic
                        counter
generic (m :Natural:=2) ; -- declare constant m = 1, a natural number
Signal Y0: Std_Logic_Vector (n - 1 downto 0): = "00"; -- Initial Assignment
to Y0 is ('0', '0')
Signal Y1: Std_Logic_Vector (n - 1 downto 0): = "01"; -- Initial Assignment
to Y1 is ('0', '1')
Signal Y2: Std_Logic_Vector (n - 1 downto 0): = "00"; -- Initial Assignment
to Y0 is ('1', '1')
Signal Y3: Std_Logic_Vector (n - 1 downto 0): = "01"; -- Initial Assignment
to Y1 is ('0', '1')
Signal Select_out: Std_Logic_Vector (m - 1 downto 0): = "1"
                        ; -- Initial Assignment is ('1')
Signal A: Std_Logic_Vector (n - 1 downto 0): = "1"; -- Initial Assignment to
Y0 is ('0', '1')

Component TestDeMUX1_2 is
port(Y0, Y1, Y2, Y3: out Std_Logic_Vector (n - 1 downto 0);
      Select_out: out Std_Logic_Vector (m - 1 downto 0);
      A: out Std_Logic_Vector (n - 1 downto 0)) ; -- Remember n = 2 in
entity
end Component TestDeMUX1_2;
begin
    TestBDeM12: TestDeMUX1_2 port map (Y0, Y1, Select_out, A)
Process
begin
    A <= "01"; -- Assign Y0 = ('0', '0')
    Wait for 8 ns;
    Select_out <= "1"; -- Assign address ('1')
    Wait for 1 ns;
    If (Y1 = "01") then error_count = error_count+1;
    end If;
    Assert (Y1 = "01") Report "Y1 is not addressed at address 1 (channel 1)"
Severity Error;
```

```
A <= "11"; -- Assign Y1 = ('1', '1')
Wait for 8 ns;
Select_out <= "0"; -- address = ('0')
Wait for 1 ns;
If (Y0 /= "11") then error_count = error_count+1;
end If;
Assert (Y0 = "11") "Y1 is not addressed at address 1 (channel 1)"
Severity Error;
;-----
; Write tests for other possible combinations in order to test thoroughly
If (error_count =0) then Assert false
Report "TestbenchDeMUX_14 completed successfully"
Severity note; -- Test result at count value
else Assert true
Report "DeMux test unsuccessful"
Severity Error; "Testbench deMux found unsuccessful"
end If;
Wait; -- Wait for next commands
end Process;
end RTLTestBDeMUX1_4;
-----
Configuration Config_RTLTestBDeMUX1_4 of TestBDeMUX1_4 is
  for RTLTestBDeMUX1_4
    end for;
end Config_RTLTestBDeMUX1_4;
```

■ EXERCISES

1. Describe example of entity and architecture of 16-bit adder with output carry after the operations. The addition occurs only when Signal Add1 = 1 is applied.
2. Describe example of entity and architecture of 4-bit adder with output carry after the operations. The addition occurs only when Signal Add1 = 1 is applied. The addition takes place when Sub1 = 0 and Add1 = 1. The subtraction takes place when Sub1 = 1 and Add1 = 0.
3. Repeat exercise for generation of two's complement output from the entity in adder-cum-subtractor.
4. A sequential circuit clock give outputs as follows: (Clock outputs are used inputs in synchronous sequential circuits)
SeqClk: = 0 if RST_Clk = 1 and SeqClk_Start = 0.
SeqClk: = 1 if SET_Clk = 1 and SeqClk_Start = 0
SeqClk is a series of sequences of 1 and 0 and sequence repeats after 50 ns
if SeqClk_Start = 1 (RST_Clk can be 0 or 1 or SET_Clk can be 0 or 1.)

Assume clock interval T for one sequence of 1 is 10 ns and 0 for 40 ns each.

SeqClk1 is clock output which is not SeqClk means 180 degree out of phase SeqClk.

How do you entity SeqClk design unit? How do you architecture RTLSeqClk design unit of SeqClk? What is testbench for RTLSeqClk? What will be the waveform of the sequential circuit clock give outputs after the Process executes?

5. Draw a sequential circuit for 8-bit D -latch. Model the entity and behaviour.
6. Model test bench for a sequential circuit for 8-bit D -latch. Model the entity and behaviour.
7. Draw a sequential circuit for 8-bit counter. Model the entity and behaviour using FSM concept.
8. Model test bench for a sequential circuit for 8-bit counter. Model the entity and behaviour.
9. Draw an FSM for (i) Mealy_StatesCircuit_SeqM2 and (ii) Combinational logic. Combinational logic operation is X xor Q_i . Mealy_StatesCircuit_M2 (memory section) One input-output port is \underline{Y}_i . (\underline{Y}_i also interconnects to output port \underline{Y} .) ($\underline{Y}_i \leq \underline{Y}$ delayed (0).) \underline{Y} is output of the combinational logic xor operations of X and \underline{Q}_i inputs. Mealy_StatesCircuit_M2 output is at port \underline{Q} . \underline{Q} interconnects the combinational logic input at port Q_i . $Q_i = Q'_i$ delayed(0).
10. Describe encoder Encod_C. The encoder has Signal Y in the output and sixteen inputs, \underline{A} . An input is selected when control input $G = 0$. $Y = \text{Selected } i^{\text{th}} \text{ input } A_i$. Write the truth tables. Describe RTL entity and architecture for the synthesis of the encoder.
11. Describe three multiplexer multiplexers, MUX1_C, MUX2_C and MUX3_C. An Input is X . A multiplexer has Signal Y in the output and 16 address inputs, \underline{A} . An address is selected by four addr_sel_In inputs. $Y = \text{Selected address } i^{\text{th}} \text{ output} = X$ and remaining address outputs are 1 in MUX 1, 0 in MUX2 and in tri-state in MUX3. Write the truth tables. Describe RTL entity and architecture for the synthesis of each MUX.
12. Describe a demultiplexer which has Signal X and sixteen channel outputs, \underline{Y} . A channel is selected by four ch_sel_In inputs. Selected channel i^{th} output Signal $Y_i = X$ and remaining channel outputs are in tri-state.

■ QUESTIONS

1. How to model RTL entity, behaviour and test bench for an adder?
2. How to model RTL entity, behaviour and test bench for a two's complement generator circuit. hgtm adder.
3. How do you model sequential circuit entity, behaviour and test bench for a divider by 4 counter?

18.58 Digital Systems: Principles and Design

4. How do you model sequential circuit entity, behaviour and test bench for a JK-FF +ve edge triggered?
5. Model test bench for a sequential circuit for decade counter. Model the entity and behaviour.
6. Draw an FSM for (i) Mealy_StatesCircuit_SeqM2 and (ii) Combinational logic for states in an 8-bit shift register. Assume Combinational logic operation is X xor Q_i . Mealy_StatesCircuit_M2 (memory section) One input-output port is \underline{Y}_1 . \underline{Y}_1 also interconnects to output port \underline{Y} . ($\underline{Y}_1 \leq \underline{Y}$ delayed(0).) \underline{Y} is output of the combinational logic xor operations of X and \underline{Q}_1 inputs. Mealy_StatesCircuit_M2 output is at port \underline{Q} . \underline{Q} interconnects the combinational logic input at port \underline{Q}_i . $\underline{Q}_i = Q$ 'delayed(0).
7. How do you model sequential circuit entity, behaviour and test bench for a 4 to 1 multiplexer?
8. How do you model sequential circuit entity, behaviour and test bench for a 1 to 4 demultiplexer?

SOLVED QUESTION PAPERS

Model Question Paper I

B.E./B.Tech. Degree Examination

Time: Three hours

Maximum: 100 marks

Answer ALL questions

PART A ($10 \times 2 = 20$ marks)

1. Mention the advantages and disadvantages of Analog Circuits.
2. Explain XOR logic operation.
3. Use DeMorgan theorem to simplify $F = \bar{A} \cdot \bar{B} + \bar{C} \cdot \bar{D}$. (Note: This is an example to place the bar over the terms).
4. What is Buffer Gate?
5. What is the difference between half adder and full adder?
6. Implement half adder using gates.
7. What is edge triggering?
8. What is SISO?
9. What is Race Free Assignment?
10. Define Fan Out.

PART B ($5 \times 16 = 80$ marks)

11. (a) (i) Simplify $A \cdot C + A \cdot (C + B) + C \cdot (C + B)$ using Boolean rules and draw the simplest possible logic circuit. (8)
(ii) From the given three inputs, A , B , and C truth table (Table 3.29), construct a Karnaugh map and then construct the SOP functions based on map for output S . Simplify the result and find a Boolean expression. (8)

Q.2 Solved Question Papers

TABLE 3.29 Truth table

A	B	C	Output S
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

or

- (b) (i) Convert $A \cdot B \cdot C + A \cdot D$ expression into standard SOP format. (8)
- (ii) Convert $(A + B + C) \cdot (A + D)$ expression into standard POS format. (8)
12. (a) (i) Define a simple method to convert a binary code to Gray code and show that Gray code converter can be made from adder circuit as a building block. (8)
- (ii) Design a 2 bit magnitude comparator. (8)
- or
- (b) (i) Implement full adder circuit using
- (1) Decoder (6)
 - (2) Multiplexer. (6)
- (ii) How can you convert a decoder into a demultiplexer? (4)
13. (a) (i) Explain Master Slave JK Flip-Flop. (8)
- (ii) If in Figure 6.2(a) circuit S denotes the R input and R denotes S input, show that we get an $\bar{S} - \bar{R}$ latch. Why? (8)
- or
- (b) (i) Perform state reduction or minimization of state table in Table 7.13, if feasible. (8)
- (ii) Formulate the problem and how will you implement a 3-line to 8-line decoder of truth table as per Table 4.5. Output is active at 0 and inactive at = 1 (output when line not selected). (8)
14. (a) Design a Ring counter. (16)
- or
- (b) What is a Shift Register? Explain any two types. (16)
15. (a) Write notes on ROM and its types. (16)
- or
- (b) Write notes on: TTL, ECL and CMOS digital logic families. (16)

Model Question Paper II

B.E./B.Tech. Degree Examination

Time: Three hours

Maximum: 100 marks

Answer ALL questions

PART A ($10 \times 2 = 20$ marks)

1. What is variable mapping?
2. What is Rule of Complementation?
3. Mention the differences between DMUX and MUX.
4. Draw the truth table for 3-line to 8-line decoder.
5. Give the state diagram of JK FF.
6. Convert JK FF to DFF.
7. What is ASCII?
8. What is Static-0 and Static-1 Hazard?
9. Define noise margin.
10. What is FPGA?

PART B ($5 \times 16 = 80$ marks)

11. (a) From the given maxterms in Table 3.34, find the Karnaugh map. Conditions for $A = 1$ are not specified and are to be taken as don't care. Find the POS expression also. (16)

Q.4 Solved Question Papers

TABLE 3.34 Maxterms and POS output function

A	B	C	D	Maxterm	POS Output function
0	0	0	0	$Mx0 \ A + B + C + D$	1
0	0	0	1	$Mx1 \ A + D + C + \bar{D}$	1
0	0	1	0	$Mx2 \ A + B + \bar{C} + D$	0
0	0	1	1	$Mx3 \ A + B + \bar{C} + \bar{D}$	1
0	1	0	0	$Mx4 \ A + \bar{B} + C + D$	0
0	1	0	1	$Mx5 \ A + \bar{B} + C + \bar{D}$	1
0	1	1	0	$Mx6 \ A + \bar{B} + \bar{C} + D$	0
0	1	1	1	$Mx7 \ A + \bar{B} + \bar{C} + \bar{D}$	1

or

- (b) (i) Find minterms and give SOP form of a combinational logic circuit that has the truth table as in Table 2.9. (8)

TABLE 2.9

Term number	A	B	C	D	Output S1
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	1

- (ii) Find maxterms and give POS form of a combinational logic circuit that has the truth table as in Table 2.10. (8)

TABLE 2.10

Term number	A	B	C	D	Output P1
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

12. (a) (i) Give a logic design and implementation using the multiplexers for $F_1 = \Sigma m(3, 7)$ using a 4 of 1 multiplexer. (10)
(ii) Explain the concept and working of quadruple 2 to 1 line multiplexer. (6)
- or
- (b) (i) Find Karnaugh map of $X + \bar{Y} \cdot Z$ in POS standard format (POS cover) and verify the answer by minimizing the map. (8)
(ii) Give Karnaugh map of $X + Y \cdot Z$ in a SOP cover (standard format) and verify the answer by minimizing the map. (8)
13. (a) Design a mod-6 counter using FFS. Draw the state transition diagram of the same. (16)
- or
- (b) How will you make circuit of binary asynchronous up and down (U/\bar{D}) counters? (16)
14. (a) Show how to program the fusible links to get a 4-bit Gray code from the binary inputs using a PAL. Also compare the design requirement with a PROM. (16)
- or
- (b) Show how to program the fusible links to get the four Boolean expressions correspond to the given sets of 4 combinational circuits implemented using PLA. Compare the number of fusible links and gates needed with respect to the PAL and PROM. (16)

Q.6 Solved Question Papers

15. (a) Explain the characteristics and implementation of the following digital logic families.

- (i) TTL. (8)
- (ii) CMOS. (8)

or

(b) Describe the concept, working and applications of the following memories:

- (i) PLD. (6)
- (ii) FPGA. (5)
- (iii) EPROM. (5)

Solutions to Model Question Paper I

PART A ($10 \times 2 = 20$ marks)

1. Refer Page No. 1.3.
2. Refer Page No. 2.4.
3. Refer Page No. 2.16.
4. Refer Page No. 13.3.
5. Refer Page No. 4.2.
6. Refer Page No. 4.3.
7. Refer Page No. 6.25.
8. Refer Page No. 8.5.
9. Refer Page No. 9.18.
10. Refer Page No. 13.10.

PART B ($5 \times 16 = 80$ marks)

11. (a) (i) Refer Page No. 2.19.
(ii) Refer Page No. 3.38.
or
(b) (i) Refer Page No. 2.21.
(ii) Refer Page No. 2.22.
12. (a) (i) Refer Page No. 5.12.
(ii) Refer Page No. 5.5.
or
(b) (i) (1) Refer Page No. 4.7.
(2) Refer Page No. 4.15.
(ii) Refer Page No. 4.21.
13. (a) (i) Refer Page No. 6.22.
(ii) Refer Page No. 6.27.
or

Q.8 Solved Question Papers

- (b) (i) Refer Page No. 7.20.
(ii) Refer Page No. 4.24.
14. (a) Refer Page No. 8.15.
or
(b) Refer Page No. 8.4.
15. (a) Refer Page No. 11.4.
or
(b) Refer Page No. 13.14, 13.19 and 13.27.

Solutions to Model Question Paper II

PART A ($10 \times 2 = 20$ marks)

1. Refer Page No. 3.1.
2. Refer Page No. 2.6.
3. Refer Page No. 4.15 and 4.20.
4. Refer Page No. 4.8.
5. Refer Page No. 6.9.
6. Refer Page No. 6.33.
7. Refer Page No. 5.4.
8. Refer Page No. 10.3.
9. Refer Page No. 13.11.
10. Refer Page No. 14.5.

PART B ($5 \times 16 = 80$ marks)

11. (a) Refer Page No. 3.41.
or
(b) (i) Refer Page No. 2.20.
(ii) Refer Page No. 2.20.
12. (a) (i) Refer Page No. 4.32.
(ii) Refer Page No. 4.15.
or
(b) (i) Refer Page No. 3.42.
(ii) Refer Page No. 3.43.
13. (a) Refer Page No. 8.14.
or
(b) Refer Page No. 8.31.
14. (a) Refer Page No. 12.12.
or
(b) Refer Page No. 12.18.

Q.10 Solved Question Papers

15. (a) (i) Refer Page No. 13.14.
(ii) Refer Page No. 13.27.

or

- (b) (i) Refer Page No. 12.1.
(ii) Refer Page No. 14.5.
(iii) Refer Page No. 11.8.

Index

- 
- adder, 18.1–18.6
 - test bench for, 18.3
 - adder circuit, 18.1–18.2
 - American Standard Code for Information Interchange (ASCII), 5.4–5.5
 - AND gate, 13.2
 - AND logic operation, 2.2–2.3
 - AND rules, 2.5
 - array, 15.16
 - array logic cell, 14.3–14.5
 - ASCII. *See* American Standard Code for Information Interchange
 - assertion, 17.7
 - associative laws, 2.6
 - asynchronous sequential circuit, 7.3, 9.2
 - analysis procedure of, 9.7–9.15
 - binary arithmetic units, 4.1–4.6
 - bipolar junction transistors, 13.3–13.8
 - Boolean expression
 - product of the sums and maxterms for, 2.12–2.16
 - buffer gate, 13.3
 - clock inputs, 6.25
 - edge triggering at, 6.25
 - level clocking of, 6.25
 - clocked sequential circuit, 7.3
 - code converter, 5.1–5.5
 - combination circuits
 - testing of, 17.8–17.13
 - common BCD code, 5.1–5.2
 - commutative laws, 2.6
 - computer language, 15.1
 - concurrent statements, 17.2
 - counter, 8.9–8.11, 18.6–18.10
 - test bench for, 18.7
 - D flip-flop, 6.16–6.19
 - clear and preset and, 6.18
 - latch and, 6.16–6.19
 - D-flip-flop, 18.10
 - D latch, 6.18–6.19
 - data flow model, 15.7
 - data types, 15.14–15.17
 - decimal line decoder, 4.9–4.10
 - decoder, 4.7–4.11
 - logic design and Boolean function implementation using, 4.11
 - decoder IC, 11.3
 - Demorgan theorems, 2.6–2.8
 - demultiplexer, 4.21, 18.29–18.33
 - arranged in tree topology, 4.22
 - as a decoder, 4.21–4.22
 - description language, 15.1

I.2 Index

- design library, 16.13–16.14
digital circuits, 1.2
 vs. analog circuits, 1.2
 popular representations of, 1.2
diode circuit, 13.3
diode-transistor logic (DTL), 13.12–13.14
distributive laws, 2.6
DTL. *See* diode-transistor logic (DTL)
dynamic hazards, 10.12–10.13
- ECL. *See* emitter coupled logic (ECL)
emitter coupled logic (ECL), 13.19–13.21
encoder, 4.12–4.15
encoder IC, 11.3
essential hazards, 10.13–10.14
even parity generators, 5.7–5.8
events, 17.6–17.7
excess-3 (XS-3) BCD code, 5.2
excitation table, 7.5, 9.8
- field programmable gate arrays (FPGAs), 14.5
files, 17.6
finite state machine (FSM), 15.8, 18.18
 test bench, 18.22
five variable Karnaugh maps, 3.10–3.11
flip-flop, 6.2–6.3, 18.10–18.18
 pulse clocking of latches in, 6.26
flow diagram, 9.14–9.15
flow table, 9.12–9.13
four binary input seven line-decoder, 4.10–4.11
four variable Karnaugh map, 3.5–3.10
four-bit encoder, 4.14
four-line decoder, 4.9
FPGAs. *See* field programmable gate arrays (FPGAs)
FSM. *See* finite state machine (FSM)
full adder, 4.2
function specific decoders, 4.9
fundamental mode asynchronous circuit, 9.4–9.7
- gray code converter, 5.3–5.4
- half adder, 4.1–4.2
hazards, 10.2–10.5
 types of, 10.3
hazard-free circuits, 10.13
hexadecimal encoder, 4.14–4.15
- high threshold logic (HTL), 13.24–13.25
HTL. *See* high threshold logic (HTL)
- integrated injection logic, 13.21–13.23
- JK flip-flop, 6.9–6.12, 18.13–18.14
 test bench for, 18.14–18.16
- Johnson counter, 8.16–8.18
- Karnaugh map
 design of, 3.11–3.15
 minimization by finding prime implicants, 3.18–3.20
 minimization using octet of eight adjacent cells, 3.17
 minimization using offset adjacent and diagonal
 adjacencies, 3.18
 minimization using pairs of cells, 3.15–3.16
 minimization using quads of four adjacent cells, 3.16
- latch, 6.2–6.3
level clocked SR latch, 6.8
line decoder, 4.7–4.9
 1 of 2 and 1 of 4, 4.9
line demultiplexer. *See* demultiplexer
line encoder, 4.12–4.13
line selector, 4.15–4.16
logic gates, 13.1–13.33
- master-slave JK flip-flop, 6.22–6.24
master-slave RS flip-flop, 6.19–6.21
Mealy model circuit
 classification of sequential circuits as, 7.4
Moore model circuit, 7.3–7.4
 classification of sequential circuits as, 7.3–7.4
MOS logic circuits (CMOSs), 13.30
MOSFET circuits, 13.6–13.8
multi-line decoder, 4.10
multi-output simplification, 3.28–3.31
multiplexer
 outputs enabling control pins and, 4.16–4.20
multiplexer, 4.15–4.20, 18.25–18.29
- NAND gate, 2.3
negative logic, 1.2
NOR gate, 2.4, 13.2
NOT gate, 13.1
NOT logic operation, 2.1–2.2
NOT rules, 2.6

- NOT-NOT logic operation, 2.5
 NOT-XOR (XNOR) logic operation, 2.4
 NOT-XOR gate, 13.2
 now and wait, 17.5–17.6
 $n\text{-}p\text{-}n$ transistor common emitter circuit, 13.3–13.6
- odd parity generators, 5.7–5.8
 odd sequence Johnson counter, 8.18–8.19
 operators, 15.17–15.20
 OR gate, 13.2
 OR logic operation, 2.3
 OR rules, 2.5
- package, 16.1–16.5
 PAL. *See* programmable array logic (PAL)
 parallel-in parallel-out buffer register, 8.3–8.4
 parallel-in serial-out (PISO) right shift register, 8.8–8.9
 PLA. *See* programming logic arrays (PLA)
 port, 15.7–15.8
 positive logic, 1.1–1.2
 pre-programmed read only memory, 11.4–11.10
 prime implicant, 3.19
 priority encoder, 4.13–4.14
 programmable array logic (PAL), 12.6–12.9
 programmable read only memory (PROM), 11.4–11.10
 programming logic arrays (PLA), 12.9–12.12
 PROM. *See* programmable read only memory
 pulse mode sequential circuit, 10.15
- Quine-McCluskey method, 3.34–3.38
 for finding prime implicants, 3.34–3.37
 finding minimal sum for the multi-output case using, 3.38
- race-free assignments, 9.18–9.22
 races, 9.15–9.18
 cycles of, 9.17–9.18
 register transfer language (RTL), 15.7
 registered PAL, 14.2–14.3
 registers, 8.1–8.4
 bi-stable latches as, 8.3
 number of bits in, 8.4
 report, 17.7
 resistor-transistor logic (RTL), 13.8–13.11
 ring counter, 8.15–8.16
 ripple counter, 8.12–8.19
- ROM special versions, 11.8–11.10
 for programming languages, 11.8–11.10
 ROM. *See* pre-programmed read only memory
 RTL design
 behaviour model for process in, 15.9–15.10
 combination logic and, 15.11–15.13
 RTL design, 15.6–15.9
 RTL. *See* resistor-transistor logic (RTL)
- sequential circuits, 16.14–16.19
 testing of, 17.8–17.13
 sequential statements, 17.2
 serial in serial out (SISO) shift register, 8.4
 serial-in parallel-out (SIPO) right shift register, 8.6–8.7
 serial-in serial-out (SISO) unidirectional shift register, 8.5–8.6
 sensitivity list, 17.6–17.7
 7536, 2421, and 5421 format BCD codes, 5.2–5.3
 severity functions, 17.7
 sharp operation, 3.32–3.33
 shift registers, 8.4–8.9
 SISO. *See* serial in serial out (SISO) shift register
 six variable Karnaugh maps, 3.10–3.11
 16-line decoder, 4.9
 SOPs. *See* sum of the products (SOPs)
 SR latch, 6.3–6.8
 difficulties in using, 6.6
 level clocked SR latch, 6.8
 timing diagrams of an, 6.6
 at various input conditions, 6.4
 stable circuit asynchronous mode operation, 9.4
 star product operation, 3.31–3.32
 state diagram, 7.8, 9.10–9.11
 state table, 7.7, 9.9–9.10
 static hazards
 elimination of, 10.10–10.12
 identification of, 10.5–10.10
 static-0 hazard, 10.3
 static-1 hazard, 10.4–10.5
 structural modeling, 17.7–17.8
 instantiation during, 17.7–17.8
 subprograms, 16.5–16.13
 subtypes, 15.16
 synchronous counter, 8.19–8.20
 using additional logic circuit, 8.19–8.20
 synchronous sequential circuit, 7.2, 9.1–9.2, 16.17–16.18

I.4 Index

- T flip-flop, 6.12–6.15
- test benches, 17.13–17.15
- three-bit encoder, 4.14–4.15
- three-variable Karnaugh map, 3.1–3.5
- timing diagrams, 8.21–8.23
- transistor-transistor logic (TTL), 13.14–13.18
- transition table, 7.7, 9.8–9.9
- TTL. *See* transistor-transistor logic (TTL)
- type checking, 15.17
- unit distance code converter, 5.3
- unstable circuit operation, 9.2–9.4
- vectors, 17.3
- very-high-speed integrated circuits (VHSIC), 15.1
- VHDL data objects, 15.5
- VHDL identifiers, 15.4
- VHDL libraries, 15.4
- VHDL standard IEEE 1076, 15.2–15.3
- VHDL standard IEEE 1164, 15.3–15.4
- VHSIC. *See* very-high-speed integrated circuits (VHSIC)
- XOR gate, 13.2
- XOR logic operation, 2.4