

**Eastern  
Economy  
Edition**

**Fourth Edition**

# Fundamentals of Digital Circuits



**A. Anand Kumar**

# **FUNDAMENTALS OF DIGITAL CIRCUITS**

## **FOURTH EDITION**

**A. Anand Kumar**

Principal  
K.L. University College of Engineering  
K.L. University  
Green Fields, Vaddeswaram  
Guntur District  
Andhra Pradesh

**PHI Learning Private Limited**  
Delhi-110092  
2016

**FUNDAMENTALS OF DIGITAL CIRCUITS, Fourth Edition**

A. Anand Kumar

© 2016 by PHI Learning Private Limited, Delhi. All rights reserved. No part of this book may be reproduced in any form, by mimeograph or any other means, without permission in writing from the publisher.

**ISBN-978-81-203-5268-1**

The export rights of this book are vested solely with the publisher.

**Twenty-sixth Printing (Fourth Edition)** ... ... **July, 2016**

Published by Asoke K. Ghosh, PHI Learning Private Limited, Rimjhim House, 111, Patparganj Industrial Estate, Delhi-110092 and Printed by Mohan Makhijani at Rekha Printers Private Limited, New Delhi-110020.

*To  
the memory of  
My parents  
**Shri A. Nagabhushanam and Smt. A. Ushamani**  
(Freedom Fighters)*



# CONTENTS

Preface .....	xxiii
Symbols, Notations .....	xxvii
Abbreviations .....	xxix
<b>1 INTRODUCTION .....</b>	<b>1–27</b>
1.1 DIGITAL AND ANALOG SYSTEMS .....	1
1.2 LOGIC LEVELS AND PULSE WAVEFORMS .....	3
1.3 ELEMENTS OF DIGITAL LOGIC .....	5
1.4 FUNCTIONS OF DIGITAL LOGIC .....	5
1.4.1 Arithmetic Operations .....	5
1.4.2 Encoding .....	6
1.4.3 Decoding .....	7
1.4.4 Multiplexing .....	7
1.4.5 Demultiplexing .....	8
1.4.6 Comparison .....	8
1.4.7 Code Conversion .....	8
1.4.8 Storage .....	8
1.4.9 Counting .....	9
1.4.10 Frequency Division .....	9
1.4.11 Data Transmission .....	9
1.5 DIGITAL INTEGRATED CIRCUITS .....	10
1.5.1 Levels of Integration .....	10
1.6 MICROPROCESSORS .....	11

**vi CONTENTS**

1.7	DIGITAL COMPUTERS .....	12
1.7.1	Major Parts of a Computer .....	12
1.8	TYPES OF COMPUTERS .....	13
	<i>SHORT QUESTIONS AND ANSWERS</i> .....	14
	<i>REVIEW QUESTIONS</i> .....	18
	<i>FILL IN THE BLANKS</i> .....	19
	<i>OBJECTIVE TYPE QUESTIONS</i> .....	20
	<i>VHDL PROGRAMS</i> .....	22
	<i>VERILOG PROGRAMS</i> .....	25
<b>2</b>	<b>NUMBER SYSTEMS .....</b>	<b>28–85</b>
2.1	THE DECIMAL NUMBER SYSTEM .....	28
2.1.1	9's and 10's complements .....	29
2.1.2	9's Complement Method of Subtraction .....	30
2.1.3	10's Complement Method of Subtraction .....	30
2.2	THE BINARY NUMBER SYSTEM .....	31
2.2.1	Counting in Binary .....	31
2.2.2	Binary to Decimal Conversion .....	32
2.2.3	Decimal to Binary Conversion .....	33
2.2.4	Binary Addition .....	37
2.2.5	Binary Subtraction .....	37
2.2.6	Binary Multiplication .....	38
2.2.7	Computer Method of Multiplication .....	39
2.2.8	Binary Division .....	39
2.2.9	Computer Method of Division .....	40
2.3	REPRESENTATION OF SIGNED NUMBERS AND BINARY ARITHMETIC IN COMPUTERS .....	41
2.3.1	Representation of Signed Numbers Using the 2's (or 1's) Complement Method	42
2.3.2	2's Complement Arithmetic .....	45
2.3.3	1's Complement Arithmetic .....	48
2.3.4	Double Precision Numbers .....	55
2.3.5	Floating Point Numbers .....	55
2.4	THE OCTAL NUMBER SYSTEM .....	56
2.4.1	Usefulness of the Octal System .....	56
2.4.2	Octal to Binary Conversion .....	56
2.4.3	Binary to Octal Conversion .....	57
2.4.4	Octal to Decimal Conversion .....	57
2.4.5	Decimal to Octal Conversion .....	57
2.4.6	Octal Arithmetic .....	59
2.5	THE HEXADECIMAL NUMBER SYSTEM .....	59
2.5.1	Hexadecimal Counting Sequence .....	60
2.5.2	Binary to Hexadecimal Conversion .....	60
2.5.3	Hexadecimal to Binary Conversion .....	61

2.5.4	Hexadecimal to Decimal Conversion .....	61
2.5.5	Decimal to Hexadecimal Conversion .....	62
2.5.6	Octal to Hexadecimal Conversion .....	63
2.5.7	Hexadecimal to Octal Conversion .....	63
2.5.8	Hexadecimal Arithmetic .....	64
	<i>SHORT QUESTIONS AND ANSWERS</i> .....	67
	<i>REVIEW QUESTIONS</i> .....	70
	<i>FILL IN THE BLANKS</i> .....	70
	<i>OBJECTIVE TYPE QUESTIONS</i> .....	71
	<i>PROBLEMS</i> .....	73
	<i>VHDL PROGRAMS</i> .....	76
	<i>VERILOG PROGRAMS</i> .....	81
<b>3</b>	<b>BINARY CODES .....</b>	<b>86–131</b>
3.1	CLASSIFICATION OF BINARY CODES .....	86
3.1.1	Numeric and Alphanumeric Codes .....	86
3.1.2	Weighted and Non-weighted Codes .....	86
3.1.3	Positively-weighted and Negatively-weighted Codes .....	87
3.1.4	Error Detecting and Error Correcting Codes .....	87
3.1.5	Sequential Codes .....	88
3.1.6	Self-complementing Codes .....	88
3.1.7	Cyclic Codes .....	88
3.1.8	Reflective Codes .....	88
3.1.9	Straight Binary Code .....	88
3.2	THE 8421 BCD CODE (NATURAL BCD CODE) .....	88
3.2.1	BCD Addition .....	89
3.2.2	BCD Subtraction .....	90
3.2.3	BCD Subtraction Using 9's and 10's Complement Methods .....	90
3.3	THE EXCESS THREE (XS-3) CODE .....	92
3.3.1	XS-3 Addition .....	92
3.3.2	XS-3 Subtraction .....	93
3.3.3	XS-3 Subtraction Using 9's and 10's Complement Methods .....	93
3.4	THE GRAY CODE (REFLECTIVE-CODE) .....	96
3.4.1	Binary-to-Gray Conversion .....	97
3.4.2	Gray-to-Binary Conversion .....	98
3.4.3	The XS-3 Gray Code .....	99
3.5	ERROR-DETECTING CODES .....	100
3.5.1	Parity .....	100
3.5.2	Check Sums .....	101
3.5.3	Block Parity .....	101
3.5.4	Five-bit Codes .....	103
3.5.5	The Biquinary Code .....	104
3.5.6	The Ring-counter Code .....	104

**viii CONTENTS**

3.6	ERROR-CORRECTING CODES .....	105
3.6.1	The 7-bit Hamming Code .....	105
3.6.2	The 15-bit Hamming Code .....	108
3.6.3	The 12-bit Hamming Code .....	109
3.7	ALPHANUMERIC CODES .....	110
3.7.1	The ASCII Code .....	110
3.7.2	The EBCDIC Code .....	111
	<i>SHORT QUESTIONS AND ANSWERS</i> .....	112
	<i>REVIEW QUESTIONS</i> .....	118
	<i>FILL IN THE BLANKS</i> .....	118
	<i>OBJECTIVE TYPE QUESTIONS</i> .....	119
	<i>PROBLEMS</i> .....	122
	<i>VHDL PROGRAMS</i> .....	124
	<i>VERILOG PROGRAMS</i> .....	128
<b>4</b>	<b>LOGIC GATES .....</b>	<b>132–175</b>
4.1	INTRODUCTION .....	132
4.2	THE AND GATE .....	133
4.2.1	Realization of AND Gate (DL AND Gate and RTL AND Gate) .....	134
4.3	THE OR GATE .....	135
4.3.1	Realization of OR Gate (DL OR Gate and RTL OR Gate) .....	135
4.4	THE NOT GATE (INVERTER) .....	136
4.4.1	Realization of NOT Gate (RTL Logic) .....	137
4.5	THE UNIVERSAL GATES .....	137
4.5.1	The NAND Gate .....	138
4.5.2	The NOR Gate .....	140
4.6	THE EXCLUSIVE-OR (X-OR) GATE .....	142
4.6.1	X-OR Gate as an Inverter .....	143
4.7	PROPERTIES OF EXCLUSIVE-OR .....	143
4.8	THE EXCLUSIVE-NOR (X-NOR) GATE .....	143
4.8.1	X-NOR Gate as an Inverter .....	144
4.9	INHIBIT CIRCUITS .....	145
4.10	PULSED OPERATION OF LOGIC GATES .....	148
	<i>SHORT QUESTIONS AND ANSWERS</i> .....	154
	<i>REVIEW QUESTIONS</i> .....	157
	<i>FILL IN THE BLANKS</i> .....	157
	<i>OBJECTIVE TYPE QUESTIONS</i> .....	157
	<i>PROBLEMS</i> .....	162
	<i>VHDL PROGRAMS</i> .....	163
	<i>VERILOG PROGRAMS</i> .....	169
<b>5</b>	<b>BOOLEAN ALGEBRA .....</b>	<b>176–231</b>
5.1	INTRODUCTION .....	176

5.2	LOGIC OPERATIONS .....	177
5.2.1	AND Operation .....	177
5.2.2	OR Operation .....	177
5.2.3	NOT Operation .....	177
5.2.4	NAND Operation .....	177
5.2.5	NOR Operation .....	177
5.2.6	X-OR and X-NOR Operations .....	178
5.3	AXIOMS AND LAWS OF BOOLEAN ALGEBRA .....	178
5.3.1	Complementation Laws .....	178
5.3.2	AND Laws .....	178
5.3.3	OR Laws .....	179
5.3.4	Commutative Laws .....	179
5.3.5	Associative Laws .....	179
5.3.6	Distributive Laws .....	180
5.3.7	Redundant Literal Rule (RLR) .....	182
5.3.8	Idempotence Laws .....	182
5.3.9	Absorption Laws .....	183
5.3.10	Consensus Theorem (Included Factor Theorem) .....	183
5.3.11	Transposition Theorem .....	184
5.3.12	De morgan's Theorem .....	185
5.3.13	Shannon's Expansion Theorem .....	186
5.3.14	Additional Theorems .....	188
5.4	DUALITY .....	188
5.4.1	Duals .....	189
5.5	REDUCING BOOLEAN EXPRESSIONS .....	189
5.6	FUNCTIONALLY COMPLETE SETS OF OPERATIONS .....	193
5.7	BOOLEAN FUNCTIONS AND THEIR REPRESENTATION .....	194
5.8	EXPANSION OF A BOOLEAN EXPRESSION IN SOP FORM TO THE STANDARD SOP FORM .....	198
5.9	EXPANSION OF A BOOLEAN EXPRESSION IN POS FORM TO 5 4 STANDARD POS FORM .....	198
5.9.1	Conversion between Canonical Forms .....	199
5.10	COMPUTATION OF TOTAL GATE INPUTS .....	202
5.11	BOOLEAN EXPRESSIONS AND LOGIC DIAGRAMS .....	203
5.11.1	Converting Boolean Expressions to Logic .....	203
5.11.2	Converting Logic to Boolean Expressions .....	203
5.12	DETERMINATION OF OUTPUT LEVEL FROM THE DIAGRAM .....	206
5.13	CONVERTING AND/OR/INVERT LOGIC TO NAND/NOR LOGIC .....	206
5.13.1	Active-Low Notation .....	211
5.14	MISCELLANEOUS EXAMPLES .....	215
	<i>SHORT QUESTIONS AND ANSWERS</i> .....	224
	<i>REVIEW QUESTIONS</i> .....	228
	<i>FILL IN THE BLANKS</i> .....	228
	<i>OBJECTIVE TYPE QUESTIONS</i> .....	229
	<i>PROBLEMS</i> .....	230

**X CONTENTS**

<b>6 MINIMIZATION OF SWITCHING FUNCTIONS.....</b>	<b>232–235</b>
6.1 INTRODUCTION .....	232
6.2 TWO-VARIABLE K-MAP .....	233
6.2.1 Mapping of SOP Expressions .....	234
6.2.2 Minimization of SOP Expressions .....	234
6.2.3 Mapping of POS Expressions .....	236
6.2.4 Minimization of POS Expressions .....	237
6.3 THREE-VARIABLE K-MAP .....	238
6.3.1 Minimization of SOP and POS Expressions .....	239
6.3.2 Reading the K-maps .....	240
6.4 FOUR-VARIABLE K-MAP .....	245
6.4.1 Prime Implicants, Essential Prime Implicants, Redundant Prime Implicants and Selective Prime Implicants .....	249
6.4.2 False Prime Implicants, Essential False Prime Implicants, Redundant False Prime Implicants and Selective False Prime Implicants .....	251
6.5 FIVE-VARIABLE K-MAP .....	253
6.6 SIX-VARIABLE K-MAP .....	257
6.7 DON'T CARE COMBINATIONS .....	260
6.8 HYBRID LOGIC .....	266
6.9 MAPPING WHEN THE FUNCTION IS NOT EXPRESSED IN MINTERMS (MAXTERMS) .....	267
6.10 MINIMIZATION OF MULTIPLE OUTPUT CIRCUITS .....	270
6.10.1 Don't Care Conditions .....	273
6.11 VARIABLE MAPPING .....	275
6.11.1 Incompletely Specified Functions .....	278
6.12 LIMITATIONS OF KARNAUGH MAPS .....	280
6.13 IMPLEMENTATION OF LOGIC FUNCTIONS .....	280
6.13.1 Two-level Implementation .....	280
6.13.2 Other Two-level Implementations .....	282
6.14 NONDEGENERATE FORMS .....	282
6.14.1 AND-OR-INVERT Implementation .....	283
6.14.2 OR-AND-INVERT Implementation .....	283
6.15 QUINE–MCCLUSKEY METHOD .....	287
6.15.1 The Decimal Representation .....	289
6.15.2 Don't Cares .....	289
6.15.3 The Prime Implicant Chart .....	289
6.15.4 Essential Prime Implicants .....	289
6.15.5 Dominating Rows and Columns .....	290
6.15.6 Determination of Minimal Expressions in Complex Cases .....	290
6.15.7 The Branching Method .....	290
<i>SHORT QUESTIONS AND ANSWERS</i> .....	307
<i>REVIEW QUESTIONS</i> .....	311
<i>FILL IN THE BLANKS</i> .....	311

<i>OBJECTIVE TYPE QUESTIONS</i> .....	313
<i>PROBLEMS</i> .....	315
<i>VHDL PROGRAMS</i> .....	318
<i>VERILOG PROGRAMS</i> .....	321
<b>7 COMBINATIONAL LOGIC DESIGN .....</b>	<b>326–459</b>
7.1 INTRODUCTION .....	326
7.2 DESIGN PROCEDURE .....	327
7.3 ADDERS .....	327
7.3.1 The Half-Adder .....	327
7.3.2 The Full-Adder .....	329
7.4 SUBTRACTORS .....	332
7.4.1 The Half-Subtractor .....	332
7.4.2 The Full-Subtractor .....	334
7.5 BINARY PARALLEL ADDER .....	336
7.5.1 The Ripple Carry Adder .....	337
7.6 4-BIT PARALLEL SUBTRACTOR .....	337
7.7 BINARY ADDER-SUBTRACTOR .....	338
7.8 THE LOOK-AHEAD-CARRY ADDER .....	338
7.9 IC PARALLEL ADDERS .....	340
7.9.1 Cascading IC Parallel Adders .....	341
7.10 2'S COMPLEMENT ADDITION AND SUBTRACTION USING PARALLEL ADDERS .....	341
7.11 SERIAL ADDER .....	342
7.11.1 Difference between Serial and Parallel Adders .....	343
7.12 BCD ADDER .....	343
7.13 EXCESS-3 (XS-3) ADDER .....	345
7.14 EXCESS-3 (XS-3) SUBTRACTOR .....	346
7.15 BINARY MULTIPLIERS .....	347
7.16 CODE CONVERTERS .....	350
7.16.1 Design of a 4-bit Binary-to-Gray Code Converter .....	351
7.16.2 Design of a 4-bit Gray-to-Binary Code Converter .....	352
7.16.3 Design of a 4-bit Binary-to-BCD Code Converter .....	354
7.16.4 Design of a 4-bit BCD-to-XS-3 Code Converter .....	355
7.16.5 Design of a BCD-to-Gray Code Converter .....	356
7.16.6 Design of an SOP Circuit to Detect the Decimal Numbers 5 through 12 in a 4-bit Gray Code Input .....	357
7.16.7 Design of an SOP Circuit to Detect the Decimal Numbers 0, 2, 4, 6, and 8 in a 4-bit 5211 BCD Code Input .....	358
7.16.8 Design of a Combinational Circuit to Produce the 2's Complement of a 4-bit Binary Number .....	359

**xii CONTENTS**

7.16.9	Design of a Circuit to Detect the Decimal Numbers 0, 1, 4, 6, 7, and 8 in a 4-bit XS-3 Code Input .....	360
7.16.10	Code Converters Using ICs .....	366
7.17	PARITY BIT GENERATORS/CHECKERS .....	367
7.17.1	Parallel Parity Bit Generator for Hamming Code .....	368
7.17.2	Design of an Even Parity Bit Generator for a 4-bit Input .....	369
7.17.3	Design of an Odd Parity Bit Generator for a 4-bit Input .....	370
7.18	COMPARATORS .....	371
7.18.1	1-bit Magnitude Comparator .....	372
7.18.2	2-bit Magnitude Comparator .....	373
7.18.3	4-bit Magnitude Comparator .....	373
7.19	IC COMPARATOR .....	374
7.20	ENCODERS .....	375
7.20.1	Octal-to-Binary Encoder .....	376
7.20.2	Decimal-to-BCD Encoder .....	377
7.21	KEYBOARD ENCODERS .....	378
7.22	PRIORITY ENCODERS .....	379
7.22.1	4-Input Priority Encoder .....	379
7.22.2	Decimal-to-BCD Priority Encoder .....	380
7.22.3	Octal-to-Binary Priority Encoder .....	382
7.23	DECODERS .....	382
7.23.1	3-Line-to-8-Line Decoder .....	384
7.23.2	Enable Inputs .....	384
7.23.3	BCD-to-Decimal Decoder (7442) .....	384
7.23.4	2-Line-to-4-Line Decoder with NAND Gates .....	385
7.23.5	Combinational Logic Implementation .....	386
7.23.6	4-to-16 Decoder from Two 3-to-8 Decoders .....	387
7.23.7	Decoder Applications .....	387
7.23.8	BCD-to-Seven Segment Decoders .....	388
7.24	MUXPLEXERS (DATA SELECTORS) .....	390
7.24.1	Basic 2-Input Multiplexer .....	390
7.24.2	The 4-Input Multiplexer .....	391
7.24.3	The 16-Input Multiplexer from Two 8-Input Multiplexers .....	392
7.25	APPLICATIONS OF MUXPLEXERS .....	392
7.25.1	Logic Function Generator .....	393
7.26	DEMULTIPLEXERS (DATA DISTRIBUTORS) .....	399
7.26.1	1-Line to 4-Line Demultiplexer .....	399
7.26.2	1-Line to 8-Line Demultiplexer .....	399
7.26.3	Demultiplexer Tree .....	401
7.27	MODULAR DESIGN USING IC CHIPS .....	402
7.27.1	Design of a 16:1 Mux Using 4:1 Mux Modules .....	402
7.27.2	Design of a 32:1 Mux Using Two 16:1 Muxs and One 2:1 Mux Modules ....	403
7.27.3	Design of a $10 \times 1k$ Decoder Using Chips of $8 \times 256$ Decoder and Additional Logic .....	403

7.27.4	Design of a $2k \times 1$ Mux Using 1k Modules .....	404
7.27.5	Design of a 2-bit Comparator Using Two 1-bit Comparator Modules .....	405
7.27.6	Design of a 4-bit Comparator Using Four 1-bit Comparator Modules .....	405
7.28	HAZARDS AND HAZARD-FREE REALIZATIONS .....	406
7.28.1	Static Hazards .....	407
7.28.2	Hazard-free Realization .....	408
7.28.3	Essential Hazards .....	409
7.29	MISCELLANEOUS EXAMPLES .....	410
	<i>SHORT QUESTIONS AND ANSWERS</i> .....	415
	<i>REVIEW QUESTIONS</i> .....	420
	<i>FILL IN THE BLANKS</i> .....	421
	<i>OBJECTIVE TYPE QUESTIONS</i> .....	422
	<i>PROBLEMS</i> .....	425
	<i>VHDL PROGRAMS</i> .....	427
	<i>VERILOG PROGRAMS</i> .....	442
<b>8</b>	<b>PROGRAMMABLE LOGIC DEVICES .....</b>	<b>460–511</b>
8.1	INTRODUCTION .....	460
8.2	READ-ONLY MEMORY (ROM) .....	462
8.3	ROM ORGANIZATION .....	462
8.4	COMBINATIONAL CIRCUIT IMPLEMENTATION .....	465
8.5	TYPES OF ROMS .....	467
8.6	COMBINATIONAL PROGRAMMABLE LOGIC DEVICES .....	468
8.7	PROGRAMMABLE ARRAY LOGIC (PAL) .....	469
8.7.1	PAL Programming Table .....	471
8.8	PROGRAMMABLE LOGIC ARRAY (PLA) .....	476
8.8.1	PLA Programming Table .....	481
8.9	PROGRAMMABLE ROM (PROM) .....	489
8.10	PROGRAMMING .....	494
8.11	PROGRAMMABLE LOGIC DEVICES—A COMPARISON .....	494
	<i>SHORT QUESTIONS AND ANSWERS</i> .....	495
	<i>REVIEW QUESTION</i> .....	499
	<i>FILL IN THE BLANKS</i> .....	499
	<i>OBJECTIVE TYPE QUESTIONS</i> .....	499
	<i>PROBLEMS</i> .....	502
	<i>VHDL PROGRAMS</i> .....	503
	<i>VERILOG PROGRAMS</i> .....	507
<b>9</b>	<b>THRESHOLD LOGIC .....</b>	<b>512–545</b>
9.1	INTRODUCTION .....	512
9.2	THE THRESHOLD ELEMENT .....	512
9.3	CONSTRUCTION OF THRESHOLD GATE .....	513

**xiv CONTENTS**

9.4 CAPABILITIES OF THRESHOLD GATE .....	515
9.5 UNIVERSALITY OF A T-GATE .....	526
9.6 IMPLEMENTATION OF BOOLEAN FUNCTION USING THRESHOLD GATE .....	529
9.7 UNATE FUNCTIONS .....	531
9.8 SYNTHESIS OF THRESHOLD FUNCTION .....	533
9.9 MULTI-GATE SYNTHESIS .....	536
<i>SHORT QUESTIONS AND ANSWERS</i> .....	540
<i>REVIEW QUESTIONS</i> .....	543
<i>FILL IN THE BLANKS</i> .....	543
<i>OBJECTIVE TYPE QUESTIONS</i> .....	543
<i>PROBLEMS</i> .....	545
<b>10 FLIP-FLOPS .....</b>	<b>546–604</b>
10.1 INTRODUCTION .....	546
10.2 CLASSIFICATION OF SEQUENTIAL CIRCUITS .....	547
10.3 LEVEL MODE AND PULSE MODE ASYNCHRONOUS SEQUENTIAL CIRCUITS ..	548
10.4 LATCHES AND FLIP-FLOPS .....	549
10.4.1 The S-R Latch .....	550
10.4.2 Gated Latches (Clocked Flip-Flops) .....	553
10.4.3 Edge-Triggered Flip-Flops .....	555
10.4.4 Triggering and Characteristic Equations of Flip-Flops .....	561
10.5 ASYNCHRONOUS INPUTS .....	563
10.6 FLIP-FLOP OPERATING CHARACTERISTICS .....	566
10.7 CLOCK SKEW AND TIME RACE .....	568
10.7.1 Potential Timing Problem in Flip-Flop Circuits .....	569
10.8 RACE AROUND CONDITION .....	569
10.9 MASTER-SLAVE (PULSE-TRIGGERED) FLIP-FLOPS .....	571
10.9.1 The Master-Slave (Pulse-Triggered) S-R Flip-Flop .....	572
10.9.2 The Master-Slave (Pulse-Triggered) D Flip-Flop .....	574
10.9.3 The Master-Slave (Pulse-Triggered) J-K Flip-Flop .....	574
10.9.4 The Data Lock-out Flip-Flop .....	576
10.10 FLIP-FLOP EXCITATION TABLES .....	576
10.11 CONVERSION OF FLIP-FLOPS .....	579
10.12 APPLICATIONS OF FLIP-FLOPS .....	584
<i>SHORT QUESTIONS AND ANSWERS</i> .....	585
<i>REVIEW QUESTIONS</i> .....	590
<i>FILL IN THE BLANKS</i> .....	591
<i>OBJECTIVE TYPE QUESTIONS</i> .....	592
<i>PROBLEMS</i> .....	595
<i>VHDL PROGRAMS</i> .....	598
<i>VERILOG PROGRAMS</i> .....	601

<b>11 SHIFT REGISTERS .....</b>	<b>605-630</b>
11.1 INTRODUCTION .....	605
11.2 BUFFER REGISTER .....	606
11.3 CONTROLLED BUFFER REGISTER .....	607
11.4 DATA TRANSMISSION IN SHIFT REGISTERS .....	607
11.5 SERIAL-IN, SERIAL-OUT, SHIFT REGISTER .....	608
11.6 SERIAL-IN, PARALLEL-OUT, SHIFT REGISTER .....	610
11.7 PARALLEL-IN, SERIAL-OUT, SHIFT REGISTER .....	610
11.8 PARALLEL-IN, PARALLEL-OUT, SHIFT REGISTER .....	611
11.9 BIDIRECTIONAL SHIFT REGISTER .....	611
11.10 UNIVERSAL SHIFT REGISTERS .....	612
11.11 DYNAMIC SHIFT REGISTERS .....	614
11.12 APPLICATIONS OF SHIFT REGISTERS .....	615
<i>SHORT QUESTIONS AND ANSWERS .....</i>	<i>616</i>
<i>REVIEW QUESTIONS .....</i>	<i>619</i>
<i>FILL IN THE BLANKS .....</i>	<i>619</i>
<i>OBJECTIVE TYPE QUESTIONS .....</i>	<i>620</i>
<i>VHDL PROGRAMS .....</i>	<i>621</i>
<i>VERILOG PROGRAMS .....</i>	<i>626</i>
<b>12 COUNTERS .....</b>	<b>631-698</b>
12.1 INTRODUCTION .....	631
12.2 ASYNCHRONOUS COUNTERS .....	633
12.2.1 Two-bit Ripple Up-counter Using Negative Edge-triggered Flip-Flops .....	633
12.2.2 Two-bit Ripple Down-counter Using Negative Edge-triggered Flip-Flops .....	634
12.2.3 Two-bit Ripple Up-down Counter Using Negative Edge-triggered Flip-Flops .....	634
12.2.4 Two-bit Ripple Up-counter Using Positive Edge-triggered Flip-Flops .....	635
12.2.5 Two-bit Ripple Down-counter Using Positive Edge-triggered Flip-Flops .....	635
12.2.6 Two-bit Ripple Up/Down Counter Using Positive Edge-triggered Flip-Flops .....	636
12.3 DESIGN OF ASYNCHRONOUS COUNTERS .....	636
12.3.1 Design of a Mod-6 Asynchronous Counter Using T FFs .....	636
12.3.2 Design of a Mod-10 Asynchronous Counter Using T FFs .....	637
12.4 EFFECTS OF PROPAGATION DELAY IN RIPPLE COUNTERS .....	638
12.4.1 Cascading of Ripple Counters .....	640
12.5 SYNCHRONOUS COUNTERS .....	642
12.5.1 Design of Synchronous Counters .....	642
12.5.2 Design of a Synchronous 3-bit Up-down Counter Using J-K FFs .....	643
12.5.3 Design of Synchronous 3-bit Up-counter .....	645

**xvi CONTENTS**

12.5.4	Design of Synchronous 3-bit Down-counter .....	646
12.5.5	Design of a Synchronous Modulo-10 up/down Counter Using T FFs .....	648
12.5.6	Design a Modulo-9 Synchronous Counter Using T FFs .....	650
12.5.7	Design of a Synchronous Modulo-6 Gray Code Counter .....	651
12.5.8	Design of a Synchronous Modulo-10 Gray Code Counter .....	652
12.5.9	Design of a Synchronous BCD Counter Using J-K FFs .....	654
12.5.10	Design of a Synchronous Mod-6 Counter Using J-K FFs .....	655
12.6	HYBRID COUNTERS .....	663
12.7	PROGRAMMABLE COUNTERS .....	664
12.8	CASCAADING OF SYNCHRONOUS COUNTERS .....	664
12.9	SHIFT REGISTER COUNTERS .....	665
12.9.1	Ring Counter .....	666
12.9.2	Twisted Ring Counter (Johnson Counter) .....	667
12.10	PULSE TRAIN GENERATORS (SEQUENCE GENERATORS) .....	669
12.10.1	Direct Logic .....	669
12.10.2	Indirect Logic .....	672
12.11	PULSE GENERATORS USING SHIFT REGISTERS .....	674
12.12	LINEAR SEQUENCE GENERATOR .....	678
	SHORT QUESTIONS AND ANSWERS .....	680
	REVIEW QUESTIONS .....	685
	FILL IN THE BLANKS .....	685
	OBJECTIVE TYPE QUESTIONS .....	686
	PROBLEMS .....	687
	VHDL PROGRAMS .....	689
	VERILOG PROGRAMS .....	691
<b>13</b>	<b>SEQUENTIAL CIRCUITS-I .....</b>	<b>699–744</b>
13.1	THE FINITE STATE MODEL .....	699
13.1.1	State Diagram .....	701
13.1.2	State Table .....	701
13.1.3	State Reduction .....	702
13.1.4	State Assignment .....	703
13.1.5	Transition and Output Table .....	703
13.1.6	Excitation Table .....	704
13.2	MEMORY ELEMENTS .....	706
13.2.1	D Flip-Flop .....	706
13.2.2	T Flip-Flop .....	706
13.2.3	S-R Flip-Flop .....	707
13.2.4	J-K Flip-Flop .....	708
13.3	SYNTHESIS OF SYNCHRONOUS SEQUENTIAL CIRCUITS .....	708
13.4	SERIAL BINARY ADDER .....	709
13.4.1	Moore Type Finite State Machine for a Serial Adder .....	711

13.5	THE SEQUENCE DETECTOR .....	712
13.5.1	Mealy Type Model .....	712
13.5.2	Moore Type Circuit .....	715
13.6	PARITY-BIT GENERATOR .....	716
13.6.1	Odd Parity-bit Generator .....	716
13.7	COUNTERS .....	718
13.7.1	Design of a 3-bit Gray Code Counter .....	718
	<i>SHORT QUESTIONS AND ANSWERS</i> .....	735
	<i>REVIEW QUESTIONS</i> .....	737
	<i>FILL IN THE BLANKS</i> .....	737
	<i>OBJECTIVE TYPE QUESTIONS</i> .....	737
	<i>PROBLEMS</i> .....	738
	<i>VHDL PROGRAMS</i> .....	740
	<i>VERILOG PROGRAMS</i> .....	741
<b>14</b>	<b>SEQUENTIAL CIRCUITS-II .....</b>	<b>745–798</b>
14.1	FINITE STATE MACHINE .....	745
14.2	CAPABILITIES AND LIMITATIONS OF FINITE STATE MACHINES .....	745
14.3	MATHEMATICAL REPRESENTATION OF SYNCHRONOUS SEQUENTIAL MACHINE .....	746
14.4	MEALY MODEL .....	747
14.5	MOORE MODEL .....	749
14.6	IMPORTANT DEFINITIONS AND THEOREMS .....	751
14.6.1	Finite State Machine—Definitions .....	751
14.6.2	State Equivalence and Machine Minimization .....	751
14.6.3	Distinguishable States and Distinguishing Sequences .....	752
14.7	MINIMIZATION OF COMPLETELY SPECIFIED SEQUENTIAL MACHINES USING PARTITION TECHNIQUE .....	752
14.8	SIMPLIFICATION OF INCOMPLETELY SPECIFIED MACHINES .....	764
14.9	MERGER CHART METHODS .....	766
14.9.1	Merger Graphs .....	766
14.9.2	Merger Table .....	768
14.10	CONCEPT OF MINIMAL COVER TABLE .....	775
14.10.1	Compatibility Graph .....	776
14.10.2	Subgraph of Compatibility Graph .....	779
14.10.3	Minimal Cover Table .....	779
	<i>SHORT QUESTIONS AND ANSWERS</i> .....	783
	<i>REVIEW QUESTIONS</i> .....	786
	<i>FILL IN THE BLANKS</i> .....	786
	<i>PROBLEMS</i> .....	787
	<i>VHDL PROGRAMS</i> .....	790
	<i>VERILOG PROGRAMS</i> .....	792

**xviii CONTENTS**

<b>15 ALGORITHMIC STATE MACHINES .....</b>	<b>799–860</b>
15.1 INTRODUCTION .....	799
15.2 COMPONENTS OF ASM CHART .....	800
15.3 SALIENT FEATURES OF ASM CHARTS .....	802
15.4 INTRODUCTORY EXAMPLES OF ASM CHARTS .....	802
15.5 ASM FOR BINARY MULTIPLIER .....	816
15.5.1 Datapath Subsystem for Binary Multiplier .....	817
15.5.2 ASM Chart for Binary Multiplier .....	818
15.5.3 Control Subsystem Using Logic Gates .....	820
15.6 ASM FOR WEIGHING MACHINE .....	824
15.6.1 System Design .....	826
15.6.2 Datapath Subsystem .....	826
15.6.3 Control Subsystem Implementation .....	827
<i>SHORT QUESTIONS AND ANSWERS</i> .....	850
<i>REVIEW QUESTIONS</i> .....	852
<i>FILL IN THE BLANKS</i> .....	852
<i>OBJECTIVE TYPE QUESTIONS</i> .....	853
<i>PROBLEMS</i> .....	854
<i>VHDL PROGRAMS</i> .....	856
<i>VERILOG PROGRAMS</i> .....	857
<b>16 LOGIC FAMILIES .....</b>	<b>861–909</b>
16.1 INTRODUCTION .....	861
16.2 DIGITAL IC SPECIFICATION TERMINOLOGY .....	862
16.2.1 Threshold Voltage .....	862
16.2.2 Propagation Delay .....	862
16.2.3 Power dissipation .....	862
16.2.4 Fan-in .....	863
16.2.5 Fan-out .....	863
16.2.6 Voltage and Current Parameters .....	864
16.2.7 Noise Margin .....	864
16.2.8 Operating Temperatures .....	865
16.2.9 Speed Power Product .....	865
16.3 LOGIC FAMILIES .....	866
16.4 TRANSISTOR TRANSISTOR LOGIC (TTL) .....	866
16.4.1 Two-input TTL NAND Gate .....	867
16.4.2 Totem-pole Output .....	868
16.4.3 Current Sinking .....	869
16.4.4 Current Sourcing .....	869
16.4.5 TTL Loading and Fan-out .....	869
16.5 OPEN-COLLECTOR GATES .....	872
16.5.1 Wired AND Operation .....	873

16.5.2	Tri-state (3-state) TTL .....	874
16.5.3	Buffer/Drivers .....	875
16.6	TTL SUBFAMILIES .....	875
16.6.1	Standard TTL, 74 Series .....	875
16.6.2	Low Power TTL, 74L Series .....	875
16.6.3	High Speed TTL, 74H Series .....	875
16.6.4	Schottky TTL, 74S Series .....	876
16.6.5	Low Power Schottky TTL, 74LS Series .....	876
16.6.6	Advanced Schottky TTL, 74AS Series .....	877
16.6.7	Advanced Low Power Schottky TTL, 74ALS Series .....	877
16.6.8	F(fast)TTL, 74F Series .....	877
16.6.9	Typical TTL Series Characteristics .....	877
16.7	INTEGRATED INJECTION LOGIC (IIL OR I2L) .....	878
16.7.1	I2L Inverter .....	878
16.7.2	I2L NAND Gate .....	879
16.7.3	I2L NOR Gate .....	880
16.8	EMITTER-COUPLED LOGIC (ECL) .....	880
16.8.1	ECL OR/NOR Gate .....	881
16.8.2	ECL Subfamilies .....	882
16.8.3	Wired OR Connections .....	882
16.8.4	Interfacing ECL gates .....	883
16.9	METAL OXIDE SEMICONDUCTOR (MOS) LOGIC .....	885
16.9.1	Symbols and Switching Action of NMOS and PMOS .....	886
16.9.2	Resistor .....	886
16.9.3	NMOS Inverter .....	887
16.9.4	NMOS NAND Gate .....	888
16.9.5	NMOS NOR Gate .....	888
16.10	COMPLEMENTARY METAL OXIDE SEMICONDUCTOR (CMOS) LOGIC .....	889
16.10.1	CMOS Inverter .....	890
16.10.2	CMOS NAND Gate .....	891
16.10.3	CMOS NOR Gate .....	892
16.10.4	Buffered and Unbuffered gates .....	893
16.10.5	Transmission gate .....	893
16.10.6	Open drain and High impedance outputs .....	894
16.10.7	Interfacing CMOS and TTL devices .....	895
16.10.8	CMOS Series .....	895
16.10.9	Operating and Performance Characteristics of CMOS .....	895
16.11	DYNAMIC MOS LOGIC .....	896
16.11.1	Dynamic MOS Inverter .....	897
16.11.2	Dynamic NAND Gate .....	898
16.11.3	Dynamic NOR Gate .....	898
16.12	INTERFACING .....	899
16.12.1	TTL to ECL .....	900
16.12.2	ECL to TTL .....	900

**xx CONTENTS**

16.12.3 TTL to CMOS .....	900
16.12.4 CMOS to TTL .....	901
<i>SHORT QUESTIONS AND ANSWERS</i> .....	901
<i>REVIEW QUESTIONS</i> .....	906
<i>FILL IN THE BLANKS</i> .....	906
<i>OBJECTIVE TYPE QUESTIONS</i> .....	908
<b>17 ANALOG-TO-DIGITAL AND DIGITAL-TO-ANALOG CONVERTERS... 910–942</b>	
17.1 INTRODUCTION .....	910
17.2 DIGITAL-TO-ANALOG (D/A) CONVERSION .....	911
17.2.1 Parameters of DAC .....	912
17.2.2 DAC Using BCD Input Code .....	914
17.2.3 Bipolar DACs .....	915
17.3 THE R-2R LADDER TYPE DAC .....	915
17.4 THE WEIGHTED-RESISTOR TYPE DAC .....	920
17.5 THE SWITCHED CURRENT-SOURCE TYPE DAC .....	923
17.6 THE SWITCHED-CAPACITOR TYPE DAC .....	924
17.7 ANALOG-TO-DIGITAL CONVERSION .....	926
17.8 THE COUNTER-TYPE A/D CONVERTER .....	926
17.9 THE TRACKING-TYPE A/D CONVERTER .....	928
17.10 THE FLASH-TYPE A/D CONVERTER .....	930
17.11 THE DUAL-SLOPE TYPE A/D CONVERTER .....	933
17.12 THE SUCCESSIVE-APPROXIMATION TYPE ADC .....	934
17.12.1 A Specific A/D Converter .....	935
17.12.2 Voltage-to-Frequency ADC .....	936
<i>SHORT QUESTIONS AND ANSWERS</i> .....	937
<i>REVIEW QUESTION</i> .....	940
<i>FILL IN THE BLANKS</i> .....	940
<i>OBJECTIVE TYPE QUESTIONS</i> .....	941
<i>PROBLEMS</i> .....	941
<b>18 MEMORIES ..... 943–981</b>	
18.1 THE ROLE OF MEMORY IN A COMPUTER SYSTEM .....	943
18.1.1 Program and Data Memory .....	943
18.1.2 Main and Peripheral Memory .....	944
18.2 MEMORY TYPES AND TERMINOLOGY .....	944
18.2.1 Memory Organization and Operation .....	944
18.2.2 Reading and Writing .....	946
18.2.3 RAMs, ROMs and PROMs .....	947
18.2.4 Constituents of Memories .....	947
18.2.5 Applications of ROMs .....	948

18.3	SEMICONDUCTOR RAMS .....	950
18.3.1	Static RAMs (SRAMs) .....	950
18.3.2	ECL RAMs .....	951
18.3.3	Dynamic RAMs (DRAMs) .....	952
18.3.4	Address Multiplexing .....	953
18.3.5	DRAM Refreshing .....	953
18.4	MEMORY EXPANSION .....	953
18.4.1	Combining DRAMs .....	959
18.5	NON-VOLATILE RAMS .....	959
18.6	SEQUENTIAL MEMORIES .....	960
18.6.1	Recirculating Shift Registers .....	960
18.6.2	First In First Out (FIFO) Memories .....	961
18.7	MAGNETIC MEMORIES .....	962
18.7.1	Magnetic Core Memory .....	963
18.7.2	Magnetic Disk Memory .....	963
18.7.3	Magnetic Recording Formats .....	964
18.7.4	Floppy Disks .....	967
18.7.5	Hard Disk Systems .....	968
18.7.6	Magnetic Tape Memory .....	969
18.7.7	Magnetic Bubble Memory .....	970
18.8	OPTICAL DISK MEMORY .....	971
18.9	CHARGE-COUPLED DEVICES .....	972
	<i>SHORT QUESTIONS AND ANSWERS</i> .....	972
	<i>REVIEW QUESTIONS</i> .....	976
	<i>FILL IN THE BLANKS</i> .....	977
	<i>OBJECTIVE TYPE QUESTIONS</i> .....	977
	<i>PROBLEMS</i> .....	978
	<i>VHDL PROGRAMS</i> .....	979
<b>19</b>	<b>TIMING CIRCUITS AND DISPLAY DEVICES .....</b>	<b>982–1003</b>
19.1	SCHMITT TRIGGER .....	982
19.2	MONOSTABLE MULTIVIBRATOR (ONE-SHOT) .....	983
19.2.1	Integrated Circuit One-shots .....	985
19.2.2	Actual Devices .....	986
19.2.3	Applications of Monostable Multivibrators .....	988
19.3	ASTABLE MULTIVIBRATOR .....	988
19.3.1	Astable Multivibrator Using Schmitt Trigger .....	989
19.3.2	Astable Multivibrator Using 555 Timer .....	989
19.3.3	Astable Multivibrator Using Inverters .....	990
19.3.4	Astable Multivibrator Using Op-amps .....	992
19.4	CRYSTAL-CONTROLLED CLOCK GENERATORS .....	993
19.5	DISPLAY DEVICES .....	996
19.5.1	Classification of Displays .....	996

**xxii CONTENTS**

19.5.2 The Light Emitting Diode (LED) .....	997
19.5.3 The Liquid Crystal Display (LCD) .....	998
19.5.4 Operation of Liquid Crystal Displays .....	998
19.5.5 Incandescent Seven Segment Displays .....	1000
<i>SHORT QUESTIONS AND ANSWERS</i> .....	1001
<i>REVIEW QUESTIONS</i> .....	1002
<i>FILL IN THE BLANKS</i> .....	1002
<i>PROBLEMS</i> .....	1003
<b><i>Appendix—Commonly Used TTL ICs</i></b> .....	<b>1005–1010</b>
<b><i>Glossary</i></b> .....	<b>1011–1022</b>
<b><i>Answers</i></b> .....	<b>1023–1059</b>
<b><i>Index</i></b> .....	<b>1061–1070</b>

## **PREFACE**

Reflecting over 40 years of experience in the classroom, the fourth edition of this comprehensive textbook on digital circuits is developed to provide a solid grounding in the foundations of basic design techniques of digital systems. Using a student-friendly writing style, the text introduces the students to digital concepts in a simple and lucid manner with an emphasis on practical treatment and real-world applications. A large number of typical examples have been worked out, so that the students can understand the related concepts clearly. Most of the problems in the book are classroom tested. The book blends basic digital electronic theory with the latest developments in digital technology. The text is, therefore, suitable for use as course material by undergraduate students of electronics and communication engineering, electrical and electronics engineering, computer science and engineering, electronics and computers engineering, instrumentation engineering, telecommunications engineering, biomedical engineering, and information technology. As there is no specific prerequisite to understand the book except for an elementary knowledge of basic electronics, it can also be used by students of polytechnics and undergraduate and postgraduate science students pursuing courses in electronics and computer science. It can also be used by AMIE, grad IETE and MCA students.

Systems can be analog or digital. Analog and digital systems are compared and various digital systems are introduced in Chapter 1.

The switching devices used in digital systems are generally two-state devices. So, it is natural to use binary numbers in digital systems. Human beings can interpret and understand data which are available in decimal form. Binary data can be represented concisely using the octal and hexadecimal notations. For this reason, decimal, binary, octal, and hexadecimal number systems, conversion of numbers from one system to another and arithmetic operations in those systems are discussed in Chapter 2.

To provide easy communication between man and machine, and also for ease of use in various devices and for transmission, decimal numbers, symbols, and letters are coded in various ways. Several codes and arithmetic operations involving some of those codes are presented in Chapter 3.

The basic building blocks used to construct combinational circuits are logic gates. Various logic gates and the functions performed by them are described in Chapter 4.

The logic designer must determine how to interconnect the logic gates in order to convert the circuit input signals to the desired output signals. The relationship between the input and output signals can be described mathematically using Boolean algebra. Chapter 5 introduces the basic laws and theorems of Boolean algebra. It also deals with how to convert algebra to logic and logic to algebra. Starting from a given problem statement, the first step in designing a combinational logic circuit is to derive a table or formulate algebraic logic equations which describe the circuit for the realization of the output function. The logic equations which describe the circuit output must generally be simplified. The simplification of logic equations using Boolean algebraic methods is presented in Chapter 5.

The simplification of complex functions cannot be performed by the algebraic methods. More systematic methods of simplification of logic expressions, such as the Karnaugh map method and the Quine-McClusky method are introduced in Chapter 6.

Various types of digital circuits used for processing and transmission of data such as arithmetic circuits, comparators, code converters, parity checkers/generators, encoders, decoders, multiplexers, and demultiplexers are discussed in detail in Chapter 7. Generally, large digital systems are designed using IC modules. Modular design using ICs is also discussed in Chapter 7. Hazards may occur in digital systems. Various hazards and hazard free realization are also discussed in that chapter.

Logic design using Programmable Logic Devices (PLDs) has got many advantages over design using fixed function ICs. Logic design using various combinational PLDs (ROMs, PALs, PLAs, and PROMs) is discussed in Chapter 8.

A single threshold gate can be used in place of many logic gates to realize a Boolean expression. Synthesis of threshold and non-threshold functions using threshold gates is discussed in Chapter 9.

The basic memory element used in the design of sequential circuits is called flip-flop. Various types of latches and flip-flops, their parameters and the conversion of one flip-flop into another are discussed in Chapter 10.

The flip-flops can be interconnected to make registers for data storage and shifting. Various types of shift registers are described in Chapter 11.

The counter is a very widely used digital circuit. The flip-flops can be interconnected with gates to form counters. Asynchronous, synchronous, and ring counters and sequence generators are discussed in Chapter 12.

The systematic design of sequential machines is very essential. The design procedures of synchronous sequential machines using state diagrams and state tables are outlined in Chapter 13.

Sequential circuits (machines) are of two types—Mealy type and Moore type. Minimization of completely specified sequential machines using the partition technique and incompletely specified machines using merger tables and merger graphs are discussed in Chapter 14.

The algorithmic state machine (ASM) is the other name of the synchronous sequential machine. Special flow charts that have been developed to define digital hardware algorithms called ASM charts are discussed in Chapter 15.

Most of the gates, flip-flops, counters, shift registers, arithmetic circuits, encoders, decoders, etc. are available in several digital logic families. The TTL, ECL, IIL, MOS, and CMOS class of logic families are introduced in Chapter 16.

Data processing requires conversion of signals from analog to digital form and from digital to analog form. Various types of analog to digital (A/D) and digital to analog (D/A) converters are discussed in Chapter 17.

Modern data processing systems require the permanent or semi-permanent storage of large amounts of data. Both semiconductor and magnetic memories for this purpose are discussed in Chapter 18.

Timing circuits are very essential in digital circuit analysis and also display devices find wide applications. Timing circuits and display devices are discussed in Chapter 19.

The **fourth edition** of this book includes **VERILOG programs** in addition to **VHDL programs** at the end of chapters. It also presents short questions with answers, review questions, fill in the blanks with answers, multiple choice questions with answers and exercise problems at the end of each chapter.

I express my profound gratitude to all those without whose assistance and cooperation, the fourth edition of this book would not have been successfully completed. I thank my former colleagues Mr. N.S. Rane, who patiently drew all the figures in the first edition of this book in corel, and Mr. L. Krishnananda, who typed most of the portions of the original manuscript. I also thank Smt. S. Uma Maheswari for typing the additional matter in the revision of this book.

I thank Mr. B. Murali Krishna, Mr. T. Narendra Babu, and Mr. M. Venkateswara Rao, Assistant Professors, K.L. University College of Engineering, K.L. University, Vijayawada for helping me to include VHDL programs. I specially thank Mr. B. Murali Krishna for helping me to include VERILOG programs also.

I am grateful to Mr. Burugupalli Venugopala Krishna, President and Mr. B. Ravi Kumar, Vice President, Sasi Educational Society, Velivennu, West Godavari (Dt), Andhra Pradesh for encouraging and providing me with all the facilities for bringing out the second edition of this book.

I thank Mr. Koneru Satyanarayana, President, Mr. Koneru Lakshman Havish and Mr. Koneru Raja Harin, Vice Presidents and Smt. Koneru Siva Kanchana Latha, Secretary of Koneru Lakshmaniah Education Foundation (KLEF), K.L. University, Vijayawada, Andhra Pradesh for their constant support.

I thank Dr. K. Raja Rajeswari former Professor and Head, ECE Department and Dr. K.S. Lingamoorthy, former Professor and Head, EEE Department of Andhra University College of Engineering, Visakhapatnam for their encouragement.

I express my sincere appreciation to my brother Mr. A. Vijaya Kumar and to my friends, Dr. K. Koteswara Rao, Chairman, Gowtham Educational Society, Gudivada, Krishna (Dt), Andhra Pradesh and Mr. Y. Ramesh Babu and Smt. Y. Krishna Kumari of Detroit for their constant encouragement.

I thank my publisher PHI Learning and their staff, in particular Mr. Darshan Kumar, former senior editor who meticulously edited the manuscript for the first edition and Mr. Sudarshan Das,

**xxvi PREFACE**

former senior editor who made the second edition possible. I am also thankful to Mr. Ajai Kumar Lal Das, Assistant Production Manager, Ms Shivani Garg, Senior Editor, and Ms Babita Misra, Editorial Coordinator for bringing out the third and fourth editions of the book in short time.

Finally, I am deeply indebted to my wife, Smt. A. Jhansi, without whose cooperation and support this project would not have materialized. I appreciate my sons Dr. A. Anil Kumar and Mr. A. Sunil Kumar and daughters-in-law Dr. A. Anureet Kaur and Smt. A. Apurupa and granddaughters Khushi Arekapudi, Shreya Arekapudi, and Krisha Arekapudi for their constant words of encouragement.

The author will gratefully acknowledge suggestions from both students and teachers for further improvement of this book.

**A. Anand Kumar**

## SYMBOLS, NOTATIONS

$\bullet, \wedge, \cap$	And operation	n	Nano
$+, \vee, \cup$	OR operation	$t_r$	Rise time
V	Volt	$t_f$	Fall time
$\Omega$	Ohm	$t_w$	Pulse width
-	Inversion	$C_{\text{out}}$	Carry out
$\oplus$	X-OR operation	$C_{\text{in}}$	Carry in
$\odot$	X-NOR operation	$b_i$	Borrow in
F	Farad	$t_s$	Set up time
M	Maxterm, Mega	$t_h$	Hold time
m	Minterm, milli	$t_{pd}$	Propagation delay time
$d, x$	Don't care	C, CLK	Clock
T	Threshold, Toggle	EN	Enable
Q	Normal output	PRE	Preset
$\bar{Q}$	Complemented output	CLR	Clear
$Q_n$	Present state	I/P	Input
$Q_{n+1}$	Next state	O/P	Output
f	Frequency	$S_D$	Direct set
$\mu$	Micro	$R_D$	Direct reset



## **ABBREVIATIONS**

A/O	AND-OR	CSL	Current steering logic
ADC	Analog to digital converter	D/A	Digital to analog
ALU	Arithmetic logic unit	DAC	Digital to analog converter
ANSI	American National Standard Institution	DCF	Disjunctive canonical form
AOI	AND-OR-INVERT	DCTL	Direct coupled transistor logic
ASCII	American Standard Code for Information Interchange	DEMUX	Demultiplexer
ASIC	Application specific integrated circuit	DIP	Dual-in-line package
ASM	Algorithmic state machine	DL	Diode logic
BCD	Binary coded decimal	DTL	Diode transistor logic
BJT	Bipolar junction transistor	EAROM	Electrically alterable read-only-memory
CCD	Charge coupled device	EBCDIC	Extended binary coded decimal interchange code
CCF	Conjunctive canonical form	ECL	Emitter coupled logic
CD-ROM	Compact disc ROM	EEPROM	Electrically erasable programmable read-only-memory
CE	Chip enable	EFPI	Essential false prime implicant
CML	Current mode logic	EPI	Essential prime implicant
CMOS	Complementary metal oxide semiconductor	EPLD	Erasable programmable logic device
CO	Carry out	EPROM	Erasable programmable read-only-memory
CPU	Central processing unit	FA	Full adder
CS	Chip select		

### XXX ABBREVIATIONS

FF	Flip-flop	PI	Prime implicant
FIFO	First in first out	PLA	Programmable logic array
FM	Frequency modulation	PLD	Programmable logic device
FPI	False prime implicant	PMOS	P channel MOS
FSM	Finate state machine	POS	Product-of-sums
HA	Half adder	PROM	Programmable read only memory
IC	Integrated circuit	PS	Present state
IIL	Integrated injection logic	RAM	Random access memory
IRAM	Integrated random access memory	RFPI	Redundant false prime implicant
K-map	Karnaugh map	ROM	Read only memory
LCD	Liquid crystal display	RTL	Resistor transistor logic
LED	Light emitting diode	R/W	Read/write
LSB	Least significant bit	RWM	Read/write memory
LSD	Least significant digit	RZ	Return-to-zero
LSI	Large scale integration	SAC	Successive approximation type converter
LTL	Lower threshold level	SBD	Schottky barrier diode
MBM	Magnetic bubbled memory	SFPI	Selected false prime implicant
MOS	Metal oxide semiconductor	SOP	Sum-of-products
MOSFET	Metal oxide semiconductor field effect transistor	S-R	Set-reset
MQ	Multiplier/Quotient	SRAM	Static random access memory
MROM	Mask programmable read-only-memory	SSI	Small scale integration
MS	Master slave	T-gate	Threshold gate
MSB	Most significant bit	TTL	Transistor-transistor logic
MSD	Most significant digit	UART	Universal asynchronous receiver transmitter
MSI	Medium scale integration	ULSI	Ultra large scale integration
MUX	Multiplexer	UTL	Upper threshold level
NGT	Negative going transition	UVEPROM	Ultra violet erasable, programmable read only memory
NM	Noise margin	VCO	Voltage controlled oscillator
NMOS	N channel MOS	VLSI	Very large scale integration
NRZ	Non-return-to-zero	WORM	Write-once read memory
NS	Next state	X-NOR	Exclusive NOR
NVRAM	Non volatile RAM	X-OR	Exclusive OR
OROM	Optical ROM	XS	Excess
PAL	Programmable array logic		
PGT	Positive going transition		

# 1

## INTRODUCTION

### 1.1 DIGITAL AND ANALOG SYSTEMS

Electronic circuits and systems are of two kinds—*analog* and *digital*. The distinction between them is not so much in the types of semiconductor devices used in these circuits as it is in voltage and current variations that occur when each type of circuit performs the function for which it is designed. Analog circuits are those in which voltages and currents vary continuously through the given range. They can take infinite values within the specified range. For example, the output voltage from an audio amplifier might be any one of the infinite values between  $-10\text{ V}$  and  $+10\text{ V}$  at any particular instant of time. Other examples of analog devices include signal generators, radio frequency transmitters and receivers, power supplies, electric motors and speed controllers, and many analog type instruments—those having *pointers* that move in a continuous arc across a calibrated scale. By contrast, a digital circuit is one in which the voltage levels assume a finite number of distinct values. In virtually all modern digital circuits, there are just two discrete voltage levels. However, each voltage level in a practical digital system can actually be a narrow *band* or *range* of voltages.

Digital circuits are often called switching circuits, because the voltage levels in a digital circuit are assumed to be switched from one value to another instantaneously, that is, the transition time is assumed to be zero.

Switching circuits may be combinational switching circuits or sequential switching circuits. In combinational switching circuits, the output depends only on the present inputs, whereas in sequential switching circuits, the output depends on the present inputs as well as the present state of the circuit, i.e. on the past inputs also. In other words we can say that sequential circuits have memory and combinational circuits have no memory. In fact sequential circuits are nothing but combinational circuits with memory.

## 2 FUNDAMENTALS OF DIGITAL CIRCUITS

Sequential switching circuits may be synchronous sequential circuits or asynchronous sequential circuits. In synchronous sequential switching circuits, state transitions can take place only when the inputs are applied along with a clock pulse, whereas in asynchronous sequential circuits state transitions can take place any time the inputs are applied.

Digital circuits are also called *logic* circuits, because each type of digital circuit obeys a certain set of logic rules. The manner in which a logic circuit responds to an input is referred to as the circuit's logic.

Digital systems are used extensively in computation and data processing, control systems, communications and measurement. Digital systems have a number of advantages over analog systems. Many tasks formally done by analog systems are now being performed digitally. The chief reasons for the shift to digital technology are summarized below:

**Digital systems are easier to design:** The switching circuits in which there are only two voltage levels, HIGH and LOW, are easier to design. The exact numerical values of voltages are not important because they have only logical significance; only the range in which they fall is important. In analog systems, signals have numerical significance; so, their design is more involved.

**Information storage is easy:** There are many types of semiconductor and magnetic memories of large capacity which can store digital data for periods as long as necessary.

**Accuracy and precision are greater:** Digital systems are much more accurate and precise than analog systems, because digital systems can be easily expanded to handle more digits by adding more switching circuits. Analog systems will be quite complex and costly for the same accuracy and precision.

**Digital systems are more versatile:** It is fairly easy to design digital systems whose operation is controlled by a set of stored instructions called the *program*. Any time the system operation is to be changed, it can easily be accomplished by modifying the program. Even though analog systems can also be programmed, the variety of the available operations is severely limited.

**Digital circuits are less affected by noise:** Unwanted electrical signals are called noise. Noise is unavoidable in any system. Since in analog systems the exact values of voltages are important and in digital systems only the range of values is important, the effect of noise is more severe in analog systems. In digital systems, noise is not critical as long as it is not large enough to prevent us from distinguishing a HIGH from a LOW.

**More digital circuitry can be fabricated on IC chips:** The fabrication of digital ICs is simpler and economical than that of analog ICs. Moreover, higher densities of integration can be achieved in digital ICs than in analog ICs, because digital design does not require high value capacitors, precision resistors, inductors and transformers (which cannot be integrated economically) like the analog design.

**Reliability is more:** Digital systems are more reliable than analog systems.

### ***Limitations of digital techniques***

Even though digital techniques have a number of advantages, they have only one major drawback.

#### THE REAL WORLD IS ANALOG

Most physical quantities are analog in nature, and it is these quantities that are often the inputs and outputs and continually monitored, operated and controlled by a system. When these quantities

are processed and expressed digitally, we are really making a digital approximation to an inherently analog quantity. Instead of processing the analog information directly, it is first converted into digital form and then processed using digital techniques. The results of processing can be converted back to analog form for interpretation. Because of these conversions, the processing time increases and the system becomes more complex. In most cases, these disadvantages are outweighed by numerous advantages of digital techniques. However, there are situations where use of only analog techniques is simpler and more economical. Both the analog and digital techniques can be employed in the same system to advantage. Such systems are called *hybrid systems*. But the tendency today is towards employing digital systems because the economic benefits of integration are of overriding importance.

The design of digital systems may be roughly divided into three stages—SYSTEM DESIGN, LOGIC DESIGN, and CIRCUIT DESIGN. System design involves breaking the overall system into subsystems and specifying the characteristics of each subsystem. For example, the system design of a digital computer involves specifying the number and type of memory units, arithmetic units and input-output devices, as well as specifying the interconnection and control of these subsystems. Logic design involves determining how to interconnect basic logic building blocks to perform a specific function. An example of logic design is determining the interconnection of logic gates and flip-flops required to perform binary addition. Circuit design involves specifying the interconnection of specific components such as resistors, diodes and transistors to form a gate, flip-flop or any other logic building block. This book is largely devoted to a study of logic design and the theory necessary for understanding the logic design process.

## 1.2 LOGIC LEVELS AND PULSE WAVEFORMS

Digital systems use the binary number system. Therefore, two-state devices are used to represent the two binary digits 1 and 0 by two different voltage levels, called HIGH and LOW. If the HIGH voltage level is used to represent 1 and the LOW voltage level to represent 0, the system is called the *positive logic system*. On the other hand, if the HIGH voltage level represents 0 and the LOW voltage level represents 1, the system is called the *negative logic system*. Normally, the binary 0 and 1 are represented by the logic voltage levels 0 V and + 5 V. So, in a positive logic system, 1 is represented by + 5 V (HIGH) and 0 is represented by 0 V (LOW); and in a negative logic system, 0 is represented by + 5 V (HIGH) and 1 is represented by 0 V (LOW). Both positive and negative logics are used in digital systems, but the positive logic is more common. For this reason, we will use only the positive logic system in this book.

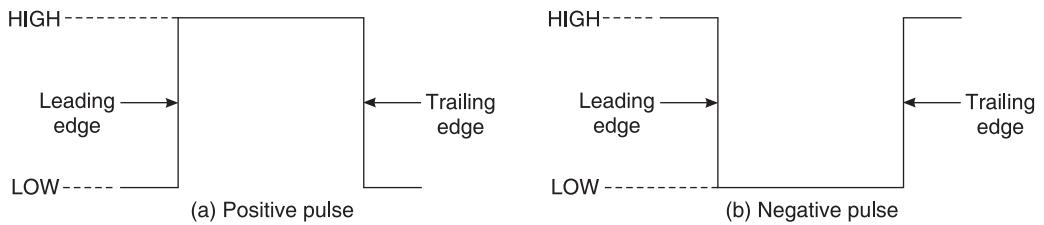
In reality, because of circuit variations, the 0 and 1 would be represented by voltage ranges instead of particular voltage levels. Usually, any voltage between 0 V and 0.8 V represents the logic 0 and any voltage between 2 V and 5 V represents the logic 1. Normally, all input and output signals fall within one of these ranges except during transition from one level to another. The range between 0.8 V and 2 V is called the *indeterminate range*. If the signal falls between 0.8 V and 2 V, the response is not predictable.

Digital circuits are designed to respond predictably to input voltages that are within the specified range. That means, the exact values of voltages are not important and the circuit gives the same response for all input voltages in the allowed range, i.e. a voltage of 0 V gives the same

## 4 FUNDAMENTALS OF DIGITAL CIRCUITS

response as a voltage of 0.4 V or 0.6 V or 0.8 V. Similarly, a voltage of 2 V gives the same response as a voltage of 2.8 V or 3.6 V or 4.7 V or 5 V.

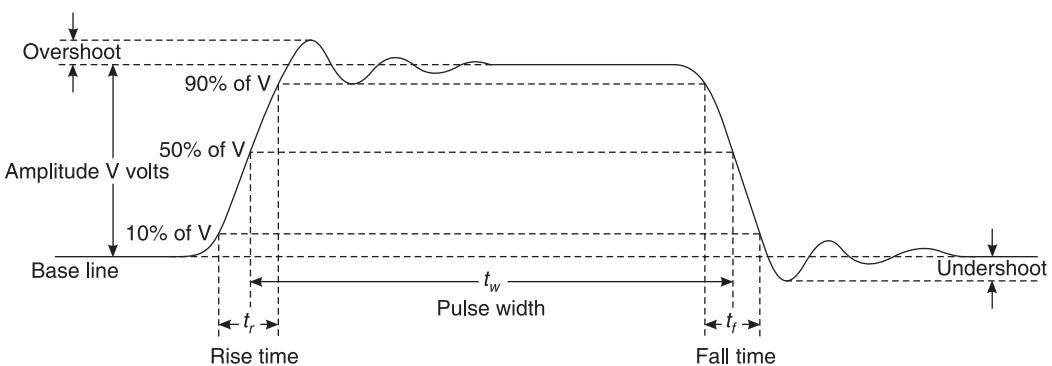
In digital circuits and systems, the voltage levels are normally changing back and forth between the HIGH and LOW states. So, pulses are very important in their operation. A pulse may be a positive pulse or a negative pulse. A single positive pulse is generated when a normally LOW voltage goes to its HIGH level and then returns to its normal LOW level as shown in Figure 1.1a. A single negative pulse is generated when a normally HIGH voltage goes to its LOW level and then returns to its normal HIGH level as shown in Figure 1.1b.



**Figure 1.1** Ideal positive and negative pulses.

As indicated in Figure 1.1, a pulse has two edges: a leading edge and a trailing edge. For a positive pulse, the leading edge is a positive going transition (PGT or rising edge) and the trailing edge is a negative going transition (NGT or falling edge), whereas for a negative pulse, the leading edge is a negative going transition (NGT) and the trailing edge is a positive going transition (PGT). The pulses shown in Figure 1.1 are ideal, because the rising and falling edges change instantaneously, i.e. in zero time. Practical pulses do not change instantaneously from LOW to HIGH or from HIGH to LOW.

A non-ideal pulse is shown in Figure 1.2. It has finite rise and fall times. The time taken by the pulse to rise from LOW to HIGH is called the *rise time* and the time taken by the pulse to go from HIGH to LOW is called the *fall time*. Because of the nonlinearities that commonly occur at the bottom and top of the pulse, the rise time is defined as the time taken by the pulse to rise from 10% to 90% of the pulse amplitude, and the fall time is defined as the time taken by the pulse to fall from 90% to 10% of the pulse amplitude. The duration of the pulse is usually indicated by pulse width  $t_w$ , which is defined as the time between the 50% points on the rising and falling edges.



**Figure 1.2** Non-ideal pulse characteristics.

Most waveforms encountered in digital systems are composed of a series of pulses and can be classified as periodic waveforms and non-periodic waveforms. A *periodic waveform* is one which repeats itself at regular intervals of time called the period,  $T$ . A *non-periodic waveform*, of course, does not repeat itself at regular intervals and may be composed of pulses of different widths and/or differing time intervals between the pulses. The reciprocal of the period is called the *frequency* of the periodic waveform. Another important characteristic of the periodic pulse waveform is its duty cycle which is defined as the ratio of the ON time (pulse width  $t_w$ ) to the period of the pulse waveform. Thus,

$$f = \frac{1}{T} \text{ and duty cycle} = \frac{t_w}{T}$$

### 1.3 ELEMENTS OF DIGITAL LOGIC

In our daily life, we make many logic decisions. The term *logic* refers to something which can be reasoned out. In many situations, the problems and processes that we encounter can be expressed in the form of propositional or logic functions. Since these functions are true/false, yes/no statements, digital circuits with their two-state characteristics are extremely useful. Several logic statements when combined form logic functions. These logic functions can be formulated mathematically using Boolean algebra (which is a system of mathematical logic) and the minimal expression for the function can be obtained using minimization techniques. There are four basic logic elements using which any digital system can be built. They are the three basic gates—NOT, AND and OR gates, and a flip-flop. In fact, a flip-flop can be constructed using gates. So, we can say that any digital circuit can be constructed using only gates. In addition to the three basic gates, there are two universal gates called NAND and NOR gates. They are called universal gates, because any circuit of any complexity can be constructed using only NAND gates or only NOR gates. In addition, there are two more gates called X-OR and X-NOR gates. We will learn about the characteristics of these gates and flip-flops in the later chapters.

Using logic gates and flip-flops, more complex logic circuits like counters, shift registers, arithmetic circuits, comparators, encoders, decoders, code converters, multiplexers, demultiplexers, memories, etc. can be constructed. These more complex logic functions can then be combined to form complete digital systems to perform specified tasks.

### 1.4 FUNCTIONS OF DIGITAL LOGIC

Many operations can be performed by combining logic gates and flip-flops. Some of the more common operations are arithmetic operations, comparison, code conversion, encoding, decoding, multiplexing, demultiplexing, shifting, counting and storing. These are all discussed thoroughly in the later chapters. The block diagram operation is given below.

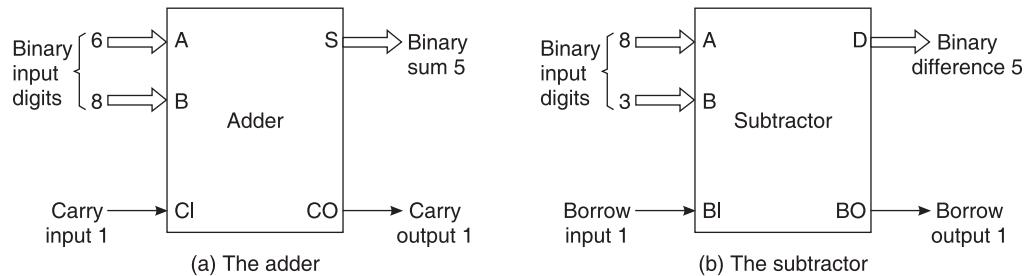
#### 1.4.1 Arithmetic Operations

The basic arithmetic operations are addition, subtraction, multiplication and division.

The arithmetic operation *addition* is performed by a digital logic circuit called the *adder*. Its function is to add two numbers *addend* (A) and *augend* (B) with a carry input (CI), and generate a

## 6 FUNDAMENTALS OF DIGITAL CIRCUITS

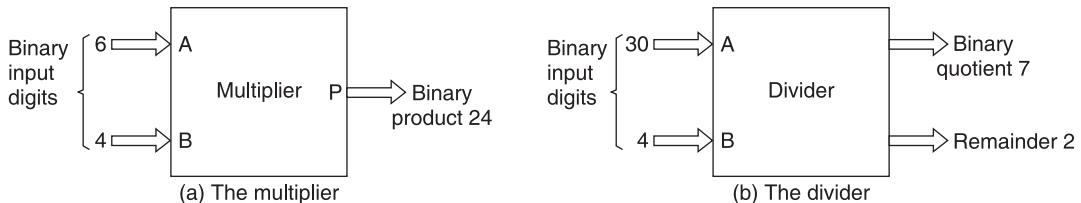
sum term (S) and a carry output term (CO). Figure 1.3a is a block diagram of an adder. It illustrates the addition of the binary equivalents of 8 and 6 with a carry input of 1, which results in a binary sum term 5 and a carry output term 1.



**Figure 1.3** The adder and the subtractor.

The arithmetic operation *subtraction* can be performed by a digital logic circuit called the *subtractor*. Its function is to subtract the subtrahend (A) from the minuend (B) considering the borrow input (BI) and to generate a difference term (D) and a borrow output term (BO). Since subtraction is equivalent to addition of a negative number, subtraction can be performed by using an adder. Figure 1.3b is a block diagram of a subtractor. It illustrates the subtraction of the binary equivalent of 8 from the binary equivalent of 3 with a borrow input of 1, which results in a binary difference term 5 and a borrow output term 1.

The arithmetic operation *multiplication* can be performed by a digital logic circuit called the *multiplier*. Its function is to multiply the *multiplicand* (A) by the *multiplier* (B) and generate the product term (P). Since multiplication is simply a series of additions with shifts in the positions of the partial products, it can be performed using an adder. Figure 1.4a is a block diagram of a multiplier. It illustrates the multiplication of 6 by 4, which results in the product term 24.



**Figure 1.4** The multiplier and the divider.

The arithmetic operation *division* can be performed by a digital logic circuit called the *divider*. Division can also be performed by an adder itself, since division involves a series of subtractions, comparisons and shifts. Its function is to divide the *dividend* (A) by the divisor (B) and generate a quotient term (Q) and a remainder term (R). Figure 1.4b is a block diagram of a divider. It illustrates the division of the binary equivalent of 30 by the binary equivalent of 4, which results in a binary quotient term 7 and a remainder term 2.

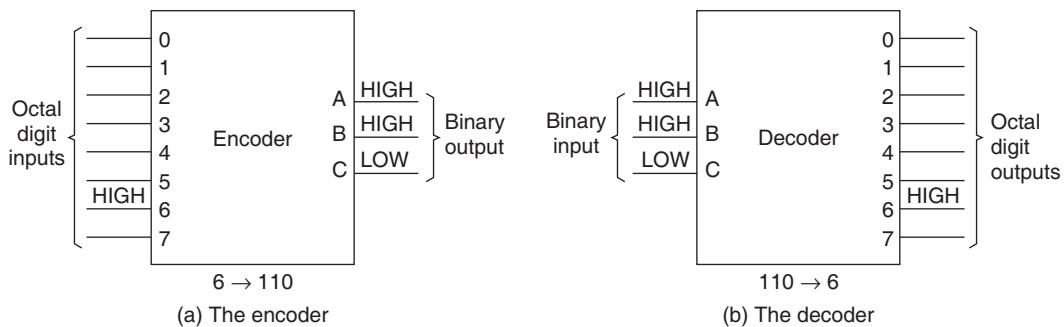
### 1.4.2 Encoding

Encoding is the process of converting a familiar number or symbol to some coded form. An *encoder* is a digital device that receives digits (decimal, octal, etc.), or alphabets, or special symbols and

converts them to their respective binary codes. In the octal-to-binary encoder shown in Figure 1.5a, a HIGH level on a given input corresponding to a specific octal digit produces the appropriate 3-bit code (ABC) on the output levels. The figure illustrates encoding of the octal digit 6 to binary 110.

### 1.4.3 Decoding

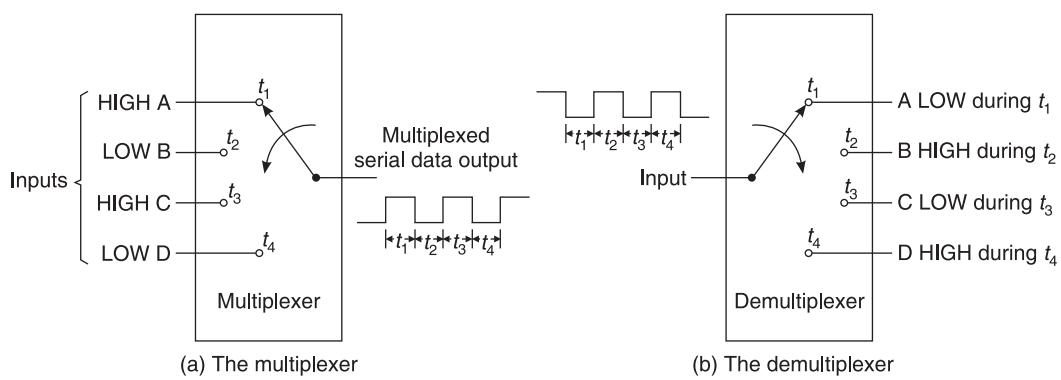
Decoding is the inverse operation of encoding. A *decoder* converts binary-coded information (ABC) to unique outputs such as decimal, octal digits, etc. In the binary-to-octal decoder shown in Figure 1.5b, a combination of specific levels on the input lines produces a HIGH on the corresponding output line. The figure illustrates decoding of the binary 110 to octal digit 6.



**Figure 1.5** The encoder and the decoder.

### 1.4.4 Multiplexing

Multiplexing means sharing. It is the process of switching information from several lines on to a single line in a specified sequence. A multiplexer or data selector is a logic circuit that accepts several data inputs and allows only one of them to get through to the output. It is an  $N$ -to-1 device. In the multiplexer shown in Figure 1.6a, if the switch is connected to input A for time  $t_1$ , to input B for time  $t_2$ , to input C for time  $t_3$  and to input D for time  $t_4$ , the output of the multiplexer will be as shown in the figure. This figure illustrates a 4-to-1 multiplexer.



**Figure 1.6** The multiplexer and the demultiplexer.

### 1.4.5 Demultiplexing

Demultiplexing operation is the inverse of multiplexing. Demultiplexing is the process of switching information from one input line on to several output lines. A demultiplexer is a digital circuit that takes a single input and distributes it over several outputs. It is a 1-to- $N$  device. In the demultiplexer shown in Figure 1.6b, if the switch is connected to output A for time  $t_1$ , to output B for time  $t_2$ , to output C for time  $t_3$  and to output D for time  $t_4$ , the output of the demultiplexer will be as shown in the figure. The figure illustrates a 1-to-4 demultiplexer.

### 1.4.6 Comparison

A logic circuit used to compare two quantities and give an output signal indicating whether the two input quantities are equal or not, and if not, which of them is greater, is called a *comparator*. Figure 1.7a shows the block diagram of a comparator. The binary representations of the quantities A and B to be compared are applied as inputs to the comparator. One of the outputs, A < B, A = B or A > B goes HIGH, depending on the magnitudes of the input quantities. The figure illustrates comparison of 8 and 4, and the result is HIGH (8 > 4).

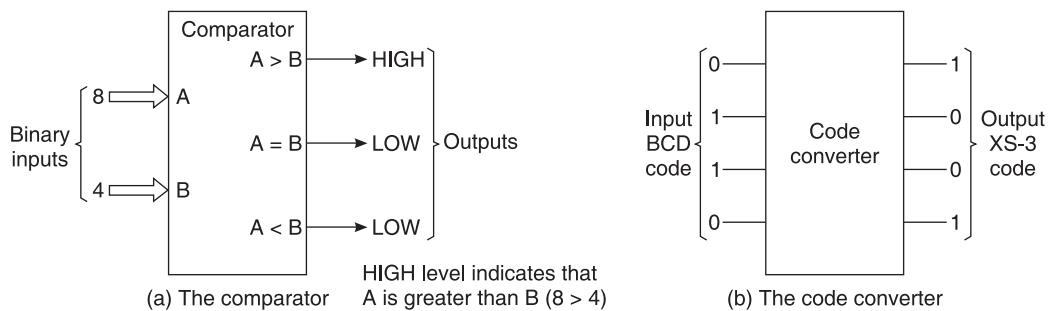


Figure 1.7 The comparator and the code converter.

### 1.4.7 Code Conversion

A logic circuit used to convert information coded in one form to another form is called a *code converter*. Figure 1.7b shows the block diagram of a BCD to XS-3 code converter. The figure illustrates conversion of decimal digit 6 coded as 0110 in 8421 BCD form to 1001 in XS-3 form.

### 1.4.8 Storage

Storage and shifting of information is very essential in digital systems. Digital circuits used for temporary storage and shifting of information (data), are called *registers*. Registers are made up of storage elements called flip-flops. Figure 1.8a shows the shifting or loading of data into a register made up of four flip-flops. After each clock pulse, the input bit is shifted into the first flip-flop and the content of each flip-flop is shifted to the flip-flop to its right. Figure 1.8b shows the shifting out of data from the register. The content of the last flip-flop is shifted out and lost.

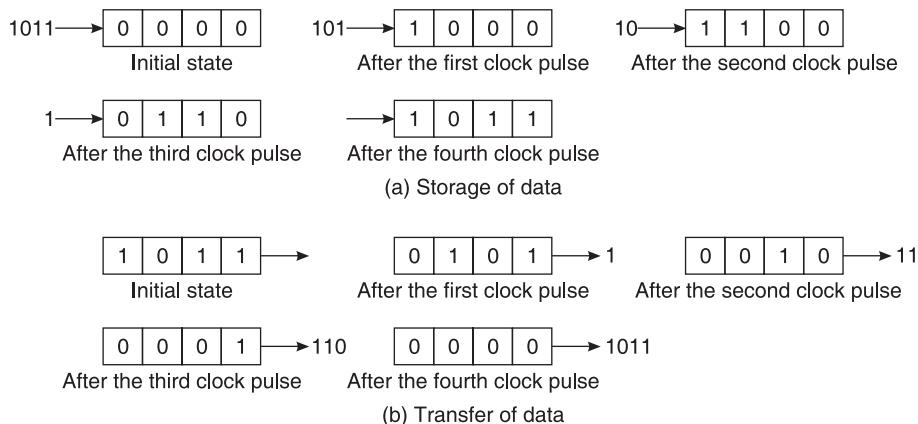


Figure 1.8 Storage and transfer of data.

#### 1.4.9 Counting

The counting operation is very important in digital systems. A logic circuit used to count the number of pulses inputted to it, is called a *counter*. The pulses may represent some events. In order to count, the counter must remember the present number, so that it can go to the next proper number in the sequence when the next pulse comes. So, storage elements, i.e. flip-flops are used to build counters too. Figure 1.9a shows the block diagram of a counter.

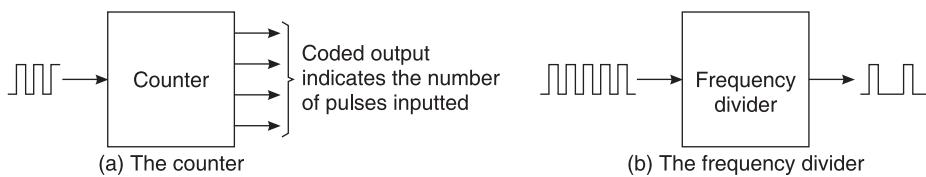


Figure 1.9 The counter and the frequency divider.

#### 1.4.10 Frequency Division

A counter can also be used to perform *frequency division*. To divide a signal of frequency  $f$  by  $N$ , the signal is applied to a mod- $N$  counter. The output of the counter will be of frequency  $f/N$ . Figure 1.9b shows the block diagram of a frequency divider.

#### 1.4.11 Data Transmission

One of the most common operations that occurs in any digital system is the transmission of information (data) from one place to another. The distance over which information is transmitted may be very small or very large. The information transmitted is in binary form, representing voltages at the outputs of a sending circuit which are connected to the inputs of a receiving circuit. There are two basic methods for transmission of digital information: *parallel* and *serial*.

In parallel data transmission, all the bits are transmitted simultaneously. So, one connecting line is required for each bit. Though data transmission is faster, the number of lines used between the transmitter and the receiver is more. This system is therefore complex and costly. On the other

## 10 FUNDAMENTALS OF DIGITAL CIRCUITS

hand, in serial transmission, the information is transmitted bit-by-bit. So, only one connecting line is sufficient between the transmitter and the receiver. Hence, serial transmission is simpler and cheaper, but slower. The principal trade-off between parallel and serial transmissions is, therefore, one of speed versus circuit simplicity. Figure 1.10a shows parallel data transmission of 8 bits and Figure 1.10b shows serial data transmission.

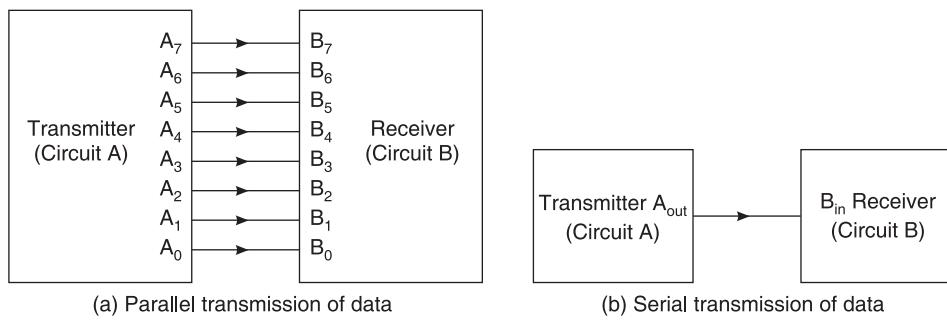


Figure 1.10 Parallel and serial data transmission.

## 1.5 DIGITAL INTEGRATED CIRCUITS

All of the logic functions that we have enumerated above (and many more) are available in the integrated circuit (IC) form. Modern digital systems utilize integrated circuits in their design. A monolithic IC is an electronic circuit, that is constructed entirely on a single piece of semiconductor material (usually silicon) called *substrate*, which is commonly referred to as a *chip*.

ICs have the advantages of low cost, low power, smaller size and high reliability over discrete circuitry (except in very specialized applications where a circuit must be ‘custom made’ to meet the unique requirements).

ICs are principally used to perform low power circuit operations such as information processing. ICs cannot handle very large voltages or currents as the heat generated in such tiny devices would cause the temperature to rise beyond acceptable limits resulting in burning out of ICs. ICs cannot easily implement certain electrical devices such as precision resistors, inductors, transformers, and large capacitors.

ICs may be classified as analog (linear) and digital. Digital ICs are complete functioning blocks as no additional components are required for their operation. The output may be obtained by applying the input. The output is a logic level 0 or 1. For analog ICs, external components are required. Digital ICs are a collection of resistors, diodes, and transistors fabricated on a single chip. The chip is enclosed in a protective plastic or ceramic package from which pins extend for connecting ICs to other devices. There are two main types of packages: dual-in-line package (DIP) and the flat package.

### 1.5.1 Levels of Integration

Digital ICs are often categorized according to their circuit complexity as measured by the number of equivalent logic gates on the substrate. There are currently five standard levels of complexity.

**Small scale integration (SSI):** The least complex digital ICs with less than 12 gate circuits on a single chip. Logic gates and flip-flops belong to this category.

**Medium scale integration (MSI):** With 12 to 99 gate circuits on a single chip, the more complex logic circuits such as encoders, decoders, counters, registers, multiplexers, demultiplexers, arithmetic circuits, etc. belong to this category.

**Large scale integration (LSI):** With 100 to 9999 gate circuits on a single chip, small memories and small microprocessors fall in this category.

**Very large scale integration (VLSI):** ICs with complexities ranging from 10,000 to 99,999 gate circuits per chip fall in this category. Large memories and large microprocessor systems, etc. come in this category.

**Ultra large scale integration (ULSI):** With complexities of over 100,000 gate circuits per chip, very large memories and microprocessor systems and single-chip computers come in this category.

Digital ICs can also be categorized according to the principal type of the electronic component used in their circuitry. They are:

- (a) Bipolar ICs—which use BJTs.
- (b) Unipolar ICs—which use MOSFETs.

Several integrated-circuit fabrication technologies are used to produce digital ICs. Presently, digital ICs are fabricated using TTL, ECL, IIL, MOS and CMOS technologies. Each differs from the other in the type of circuitry used to provide the desired logic operation. While TTL, ECL and IIL use bipolar transistors as their main circuit elements, MOS and CMOS use MOSFETs as their main circuit elements. These technologies are also called *logic families*. Several sub-families of these main logic families are also available.

## 1.6 MICROPROCESSORS

A *microprocessor* is an LSI/VLSI device that can be programmed to perform arithmetic and logic operations and other functions in a prescribed sequence for the movement and processing of data. Microprocessors are available in word lengths of 4, 8, 16, 32 and 64 bits. Presently, 128-bit microprocessors are being used in some prototype computers. The 4-bit processors are virtually obsolete. Because of their small size, low cost and low power consumption, microprocessors have revolutionized the digital computer system technology. The microprocessor is used as the central processing unit in microcomputer systems. The speed of the microprocessor determines the maximum speed of a microcomputer.

The arrangement of circuits within the microprocessor (called its architecture) permits the system to respond correctly to each of the many different instructions. In addition to arithmetic and logic operations, the microprocessor controls the flow of signals into and out of the computer, routing each to its proper destination in the required sequence to accomplish a specific task. The interconnections or paths along which signals flow are called *buses*. Figure 1.11 shows a block diagram of the microprocessor.

## 12 FUNDAMENTALS OF DIGITAL CIRCUITS

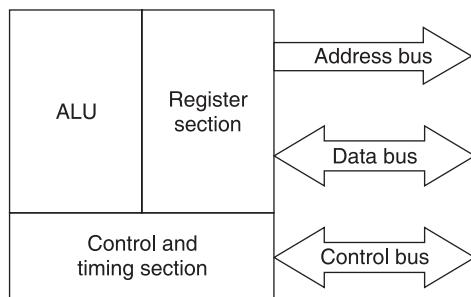


Figure 1.11 Block diagram of the microprocessor.

## 1.7 DIGITAL COMPUTERS

The digital computer is a system of hardware that performs arithmetic operations, manipulates data (usually in binary form) and makes decisions. Even though human beings can do most of the things which a computer can do, the computer does the things with much greater speed and accuracy. The computer, however, has to be given a complete set of instructions that tell it exactly what to do at each step of its operation. This set of instructions is called a *program*. Programs are placed in the computer's memory unit in binary coded form with each instruction having a unique code. The computer takes these instruction codes from memory one at a time and performs the operation called for by the code.

### 1.7.1 Major Parts of a Computer

There are several types of computer systems, but each can be broken down into the same functional units. Each unit performs specific functions, and all the units function together to carry out the instructions given in the program. Figure 1.12 shows the five major functional units of the digital computer and their interconnections. The solid lines with arrows represent the flow of information. The dashed lines with arrows represent the flow of timing and control signals. The major functions of each unit are described below:

**Input unit:** Through this unit, a complete set of instructions and data is fed into the memory unit to be stored there until needed. The information typically enters the input unit by means of a magnetic tape, or a keyboard.

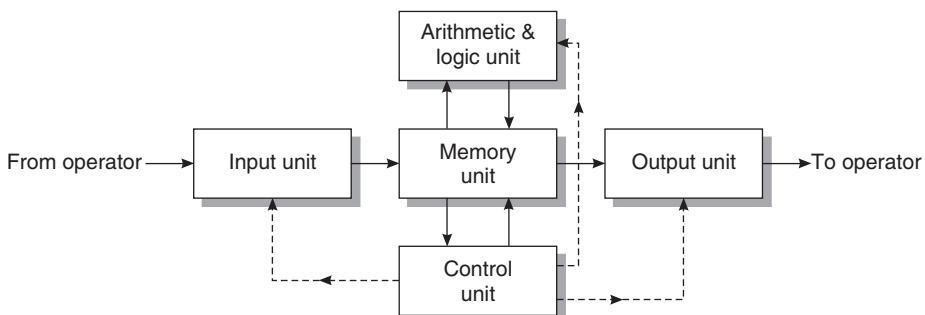


Figure 1.12 Block diagram of the digital computer.

**Memory unit:** In addition to the instructions and data received from the input unit, the memory stores the results of arithmetic and logic operations received from the arithmetic and logic unit. It also supplies information to the output unit.

**Control unit:** This unit takes instructions from the memory unit one at a time and interprets them. It then sends the appropriate signals to all the other units to cause the specific instruction to be executed.

**Arithmetic and logic unit:** All arithmetic calculations and logical decisions are performed in this unit. It then sends the results to the memory unit to be stored there.

**Output unit:** This unit takes data from the memory unit and prints out, displays or otherwise presents information to the operator.

## 1.8 TYPES OF COMPUTERS

The number of computer types depends on the criteria used to classify them. Computers differ in their physical size, operating speed, memory capacity and processing capability as well as in respect of other characteristics. The most common way to classify computers is by their physical size, which usually but not always is indicative of their relative capabilities. The three basic classifications are: microcomputer, minicomputer, and mainframe.

The microcomputer is the smallest type of computer. It generally consists of several IC chips including a microprocessor chip, memory chips, and input-output interface chips along with input-output devices such as a keyboard and video display. Microcomputers resulted from the tremendous advances in IC fabrication technology that has made it possible to pack more and more circuitry into a small space. Figure 1.13 shows a block diagram of the microcomputer system.

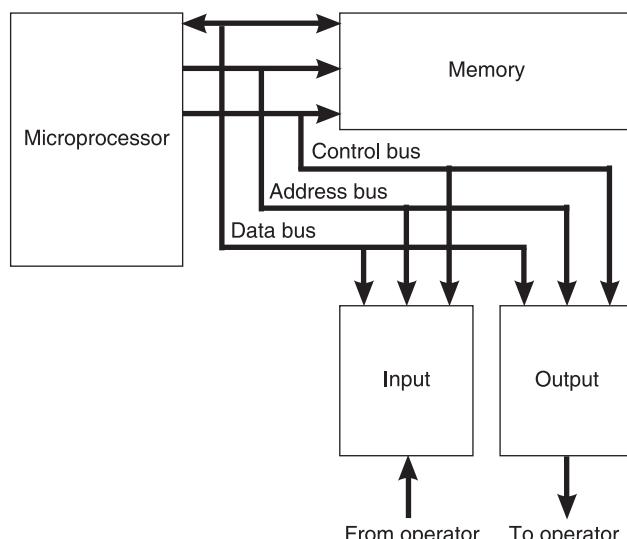


Figure 1.13 Block diagram of the microcomputer system.

## 14 FUNDAMENTALS OF DIGITAL CIRCUITS

Minicomputers are larger than microcomputers and are widely used in industrial control systems, research laboratories, etc. They are generally faster and possess more processing capabilities than microcomputers.

Mainframes are the largest computers. These maxicomputers include complete systems of peripheral equipment such as magnetic tape units, magnetic disk units, card punchers and readers (now obsolete), keyboards, printers and many more. Applications of mainframes range from computation-oriented science and engineering problem-solving to data-oriented business applications, where emphasis is on monitoring and updating of large quantities of data and information.

### SHORT QUESTIONS AND ANSWERS

1. What is an analog signal? Give two examples of analog signals.  
A. A signal which can assume any value in a given range is known as analog signal. A sinusoidal signal and amplitude modulated signal are analog signals.
2. What is a digital signal? Give two examples of digital signals.  
A. A signal which can assume only two possible values is known as a digital signal. Voltage levels 0 V and 5 V, and presence or absence of pulse are digital signals.
3. Differentiate between analog and digital signals.  
A. An analog signal is continuous and can assume any value in a given range, whereas a digital signal can have only two discrete values.
4. What are the two kinds of electronic circuits?  
A. The two kinds of electronic circuits and systems are—analog and digital.
5. What are analog circuits? Give a few examples.  
A. Analog circuits are those electronic circuits in which voltages and currents vary continuously through the given range. They can take infinite values within the specified range. Signal generators, radio frequency transmitters and receivers, power supplies, electric motors and speed controllers are some examples of analog devices, or we can say analog circuits are electronic circuits meant to process analog signals.
6. What are digital circuits?  
A. Digital circuits are those electronic circuits in which the voltage levels can assume only a finite number of distinct values.
7. Why are digital circuits called switching circuits?  
A. Digital circuits are often called switching circuits, because the voltage levels in a digital circuit are assumed to be switched from one level to another instantaneously, that is, the transition time is assumed to be zero.
8. How are switching circuits classified?  
A. Switching circuits are classified as (a) combinational switching circuits and (b) sequential switching circuits.
9. What are combinational circuits?  
A. The switching circuits whose output depends only on the present inputs are called combinational circuits.

**10.** What are sequential circuits?

- A. The switching circuits whose output depends not only on the present inputs but also on the present state (past history, i.e. on past inputs also) are called sequential circuits.

**11.** What are synchronous sequential circuits?

- A. Synchronous sequential circuits are those sequential circuits in which the state transitions takes place only if the inputs are applied along with a clock pulse.

**12.** How are sequential switching circuits classified?

- A. Sequential switching circuits are classified as (a) synchronous sequential switching circuits and (b) asynchronous sequential switching circuits.

**13.** What are asynchronous sequential circuits?

- A. Asynchronous sequential circuits are those sequential circuits in which the state transitions can take place any time the inputs are applied.

**14.** Why are digital circuits called logic circuits?

- A. Digital circuits are also called logic circuits, because each type of digital circuit obeys a certain set of logic rules.

**15.** What do you mean by circuit's logic?

- A. The manner in which a logic circuit responds to an input is referred to as the circuit's logic.

**16.** What are the advantages of digital systems over the analog systems? What is the chief limitation to the use of digital techniques?

- A. Digital systems have a number of advantages over analog systems. Digital systems are more versatile, easier to store data, easier to design, less affected by noise, more accurate and precise than analog systems. The only one major drawback of digital techniques is that "real world is not digital, it is analog".

**17.** What are hybrid systems?

- A. Systems in which both analog and digital techniques are applied in the same system are called hybrid systems.

**18.** Name the three stages of digital system design?

- A. The three stages of digital system design are: system design, logic design and circuit design.

**19.** What do you mean by system design? Give an example.

- A. System design involves breaking the overall system into subsystems and specifying the characteristics of each such system. For example, the system design of a digital computer involves specifying the number and type of memory units, arithmetic units, and input output devices, as well as specifying the interconnection and control of those subsystems.

**20.** What do you mean by logic design? Give an example.

- A. Logic design involves determining how to interconnect basic logic building blocks to perform a specific function. An example of logic design is determining the interconnection of logic gates, and flip-flops required to perform binary addition.

**21.** What is circuit design?

- A. Circuit design involves specifying the interconnection of specific components such as resistors, transistors and diodes to form a gate, flip-flop, or any other logic building block.

**22.** What is a positive logic system?

- A. A positive logic system is one in which the higher of the two voltage levels represents logic 1, and the lower of the two voltage levels represents logic 0.

## 16 FUNDAMENTALS OF DIGITAL CIRCUITS

**23.** What is a negative logic system?

- A. A negative logic system is one in which the higher of the two voltage levels represents logic 0 and the lower of the two voltage levels represents logic 1.

**24.** What do you mean by the amplitude of a pulse?

- A. The amplitude of a pulse is the height of the pulse, usually expressed in volts.

**25.** Define rise time and fall time of a pulse.

- A. The time taken by the pulse to rise from LOW to HIGH is called the rise time and the time taken by the pulse to fall from HIGH to LOW is called the fall time, but because of the nonlinearities that commonly occur at the bottom and top of the pulse, the rise time is defined as the time taken by the pulse to rise from 10% to 90% of the pulse amplitude, and the fall time is defined as the time taken by the pulse to fall from 90% to 10% of the pulse amplitude.

**26.** Define pulse width.

- A. The pulse width is defined as the time between 50% points on the rising and falling edges.

**27.** What are logic levels?

- A. In a positive logic system, the logic levels are usually +5 V for logic 1 and 0 V for logic 0 and in a negative logic system, they are +5 V for logic 0 and 0 V for logic 1. In practice 0 V to 0.8 V is treated as logic 0 and 2 V to 5 V is treated as logic 1. The 0.8 V to 2 V range is invalid and the response is unpredictable.

**28.** What is the difference between periodic and non-periodic pulse waveforms?

- A. A periodic pulse waveform is one which repeats itself at regular intervals of time called the period (T). It has the same pulse width throughout. A non-periodic pulse waveform is one which does not repeat itself at regular intervals of time and may be composed of pulses of differing pulse widths and/or differing time intervals between the pulses.

**29.** What is the duty cycle?

- A. The duty cycle of a periodic pulse waveform is the ratio of the ON time to the period of the waveform.

**30.** What is frequency?

- A. The reciprocal of the time period of a waveform is called frequency.

**31.** What is encoding? What is an encoder?

- A. Encoding is the process of converting a familiar number or symbol to some coded form. An encoder is a digital device that performs the operation of encoding, i.e. an encoder is a digital device that receives digits or alphabets, or special symbols and converts them into their respective binary codes.

**32.** What is decoding? What is a decoder?

- A. Decoding is the inverse operation of encoding, i.e. it is the process of converting the binary coded information to unique outputs. A decoder is a digital device that performs the operation of decoding, i.e. a decoder converts binary coded information to unique outputs such as digits, symbols, alphabetic characters, etc.

**33.** What is multiplexing? What is a multiplexer?

- A. Multiplexing means sharing. It is the process of switching information from several lines onto a single line in a specified sequence. A multiplexer or data selector is a logic circuit that accepts several data inputs and allows only one of them to get through to the output. It is an  $N$ -to-1 device.

**34.** What is demultiplexing? What is a demultiplexer?

- A. Demultiplexing is the inverse operation of multiplexing, i.e. demultiplexing is the process of switching information from one input line onto several output lines. A demultiplexer is a digital device that performs the operation of demultiplexing, i.e. a demultiplexer is a digital circuit that takes a single input and distributes it over several outputs. It is a 1-to- $N$  device.

**35.** What is a comparator?

- A. A comparator is a logic circuit used to compare two quantities and give an output signal indicating whether the two input quantities are equal or not, and if not, which of them is greater.

**36.** What is a code converter?

- A. A code converter is a logic circuit used to convert information coded in one form to another form.

**37.** What are registers?

- A. Digital circuits used for temporary storage and shifting of information (data) are called registers.

**38.** What are the basic methods for the transmission of digital information?

- A. The two basic methods for the transmission of digital information are (a) serial data transmission and (b) parallel data transmission.

**39.** Compare serial and parallel data transmission?

- A. In parallel data transmission, all the bits are transmitted simultaneously. So one connecting line is required for each bit. Though data transmission is faster, the number of lines used between the transmitter and the receiver is more. This system is therefore complex and costly. On the other hand, in serial transmission, the information is transmitted bit-by-bit. So only one connecting line is sufficient between the transmitter and the receiver. Hence, serial transmission is simpler and cheaper but slower.

**40.** What is a monolithic IC?

- A. A monolithic IC is an electronic circuit that is constructed entirely on a single piece of semiconductor material (usually silicon) called substrate, which is commonly referred to as a chip.

**41.** What are the advantages and drawbacks of ICs? Why cannot ICs handle large voltages and currents?

- A. ICs have the advantages of low cost, low power, smaller size, and high reliability over discrete circuits. The disadvantages of ICs are: ICs cannot handle very large voltages or currents, and ICs cannot easily implement certain electrical devices such as precision resistors, inductors, transformers and large capacitors.

**42.** Why ICs cannot handle large voltages and currents?

- A. ICs cannot handle very large voltages or currents as the heat generated in such devices would cause the temperature to rise beyond acceptable limits resulting in burning out of ICs.

**43.** How are ICs classified?

- A. ICs may be classified as analog (linear) ICs and digital ICs.

**44.** Distinguish between digital and analog ICs.

- A. Digital ICs are complete functioning blocks. No additional components are required for their operation. The output may be obtained by applying the input. Analog ICs are not complete functioning blocks. External components are required for their operation.

## **18 FUNDAMENTALS OF DIGITAL CIRCUITS**

**45.** What are the two main types of IC packages?

- A. The two main types of IC packages are: dual-in-line package and the flat-package.

**46.** How are digital ICs classified?

- A. Depending on the principal components used, digital ICs may be classified as unipolar ICs and bipolar ICs. In terms of the gate circuits per chip, digital ICs may be classified as SSI, MSI, LSI, VLSI and ULSI.

**47.** What are the different levels of integration? Give examples?

- A. The different levels of integration based on the number of gate circuits per chip are as follows:

- (a) Small scale integration SSI (less than 12 gate circuits per chip). Logic gates and flip-flops belong to this category.
- (b) Medium scale integration MSI (12 to 99 gate circuits per chip). Encoders, decoders, multiplexers, demultiplexers, counters, registers, and arithmetic circuits fall in this category.
- (c) Large scale integration LSI (100 to 9999 gate circuits per chip). Small memories and small microprocessors fall into this category.
- (d) Very large scale integration VLSI (10,000 to 99,999 gate circuits per chip). Large memories and large microprocessor systems belong to this category.
- (e) Ultra large scale integration ULSI (more than 1,00,000 gate circuits per chip). Very large memories and large microprocessor systems and single chip computers fall into this category.

**48.** What is a microprocessor?

- A. A microprocessor is an LSI/VLSI device that can be programmed to perform arithmetic and logic operations and other functions in a prescribed sequence for the movement and processing of data.

**49.** What is a digital computer?

- A. A digital computer is a system of hardware that performs arithmetic operations, manipulates data and makes decisions.

**50.** What are the major parts of a digital computer?

- A. The five major functional units of a digital computer are: (1) input unit, (2) memory unit, (3) control unit, (4) arithmetic logic unit and (5) output unit.

**51.** What are the basic classifications of computers?

- A. The three basic classifications of computers are: micro computer, mini computer, and mainframe. This classification is based on their physical size.

**52.** What is a program?

- A. The set of instructions that tells a computer what to do is called a program.

**53.** Name the IC fabrication technologies currently in use?

- A. The integrated circuit fabrication technologies currently in use are TTL, ECL, IIL, MOS and CMOS technologies.

### **REVIEW QUESTIONS**

1. Discuss the classification of electronic circuits.
2. Discuss the levels of integration with examples.

**FILL IN THE BLANKS**

1. Ordinary electrical switch is a \_\_\_\_\_ device.
2. A train of pulses is a \_\_\_\_\_ signal.
3. Output of a microphone is a \_\_\_\_\_ signal.
4. Electronic circuits may be \_\_\_\_\_ or \_\_\_\_\_.
5. The circuits in which voltages and currents can take infinite values within a given range are called \_\_\_\_\_ circuits.
6. The circuits in which the voltage levels can assume only a finite number of distinct values are called \_\_\_\_\_ circuits.
7. The circuits in which the transition time is zero are called \_\_\_\_\_ circuits.
8. Switching circuits may be \_\_\_\_\_ switching circuits or \_\_\_\_\_ switching circuits.
9. Sequential switching circuits may be \_\_\_\_\_ sequential circuits or \_\_\_\_\_ sequential circuits.
10. Switching circuits in which the output depends only on the present inputs are called \_\_\_\_\_ switching circuits.
11. Switching circuits in which the output depends on the present inputs as well as the present state are called \_\_\_\_\_ switching circuits.
12. In \_\_\_\_\_ sequential circuits, the state transitions can take place any time the inputs are applied.
13. In \_\_\_\_\_ sequential circuits, state transitions can take place only when the inputs are applied along with a clock.
14. Digital circuits are also called \_\_\_\_\_ and \_\_\_\_\_ circuits.
15. The manner in which a logic circuit responds to an input is referred to as \_\_\_\_\_.
16. Systems in which both analog and digital techniques are used are called \_\_\_\_\_ systems.
17. The three stages of system design are \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_.
18. \_\_\_\_\_ design involves breaking the overall system into subsystems and specifying the characteristics of each subsystem.
19. \_\_\_\_\_ design involves interconnection of logic gates to perform a specific function.
20. \_\_\_\_\_ design involves specifying the interconnection of specific components.
21. A system in which the higher of the two voltage levels is logic 1 and the lower of the two voltage levels is logic 0 is called \_\_\_\_\_ system.
22. A system in which the lower of the two voltage levels is logic 1 and the higher of the two voltage levels is logic 0 is called \_\_\_\_\_ system.
23. The \_\_\_\_\_ of a pulse is the height of the pulse usually expressed in volts.
24. In the commonly used logic system, logic 1 is \_\_\_\_\_ volts and logic 0 is \_\_\_\_\_ volt.
25. The ratio of the ON time to the period of the pulse is called its \_\_\_\_\_.
26. The time interval between the 50% points on the leading and trailing edges of a pulse is called the \_\_\_\_\_.
27. The time taken by the pulse to change from low-to-high is called the \_\_\_\_\_.
28. The time taken by the pulse to change from high-to-low is called the \_\_\_\_\_.
29. \_\_\_\_\_ is the process of converting a familiar number or symbol into coded form.
30. \_\_\_\_\_ is the process of converting the binary coded information into familiar number or symbol.
31. The process of switching information from several lines on to a single line is called \_\_\_\_\_.

## 20 FUNDAMENTALS OF DIGITAL CIRCUITS

32. Multiplexing means \_\_\_\_\_.
33. A multiplexer is an \_\_\_\_\_ to \_\_\_\_\_ device.
34. The process of switching information from one input line on to several output lines is called \_\_\_\_\_.
35. A demultiplexer is a \_\_\_\_\_ to \_\_\_\_\_ device.
36. Digital circuits used for temporary storage and shifting of data are called \_\_\_\_\_.
37. The basic methods of data transmission are \_\_\_\_\_ and \_\_\_\_\_.
38. ICs may be classified as \_\_\_\_\_ and \_\_\_\_\_.
39. \_\_\_\_\_ ICs are complete functional blocks.
40. Depending on the principal components, digital ICs may be classified as \_\_\_\_\_, and \_\_\_\_\_.
41. In terms of gate circuits per chip, digital ICs may be classified as \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_, and \_\_\_\_\_.
42. In SSI, the number of gate circuits per chip is \_\_\_\_\_.
43. In MSI, the number of gate circuits per chip is \_\_\_\_\_.
44. In LSI, the number of gate circuits per chip is \_\_\_\_\_.
45. In VLSI, the number of gate circuits per chip is \_\_\_\_\_.
46. In ULSI, the number of gate circuits per chip is \_\_\_\_\_.
47. Based on physical size, the digital computers may be classified as \_\_\_\_\_, \_\_\_\_\_, and \_\_\_\_\_.
48. The set of instructions that tell a computer what to do is called a \_\_\_\_\_.
49. The IC fabrication techniques currently in use are \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_, and \_\_\_\_\_.

### OBJECTIVE TYPE QUESTIONS

1. Logic circuits are
  - (a) analog circuits
  - (b) digital circuits
  - (c) hybrid circuits
  - (d) none of these
2. Switching circuits are
  - (a) analog circuits
  - (b) digital circuits
  - (c) hybrid circuits
  - (d) none of these
3. Breaking the overall system into subsystems and specifying the characteristics of each subsystem is called
  - (a) logic design
  - (b) system design
  - (c) circuit design
  - (d) none of these
4. Specifying the interconnection of specific components is called
  - (a) logic design
  - (b) system design
  - (c) circuit design
  - (d) none of these
5. The interconnection of various building blocks to perform a specific function is called
  - (a) logic design
  - (b) system design
  - (c) circuit design
  - (d) none of these
6. A sinusoidal signal is analog signal, because
  - (a) it can have a number of values between the negative and positive peaks
  - (b) it is negative for one half cycle



## VHDL PROGRAMS

### 1. VHDL PROGRAM FOR 4:2 ENCODER USING BEHAVIORAL MODELING

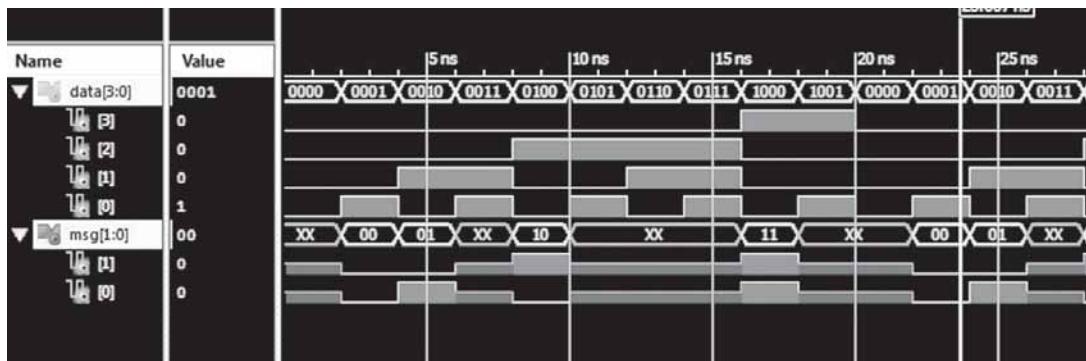
```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Encoder is
    Port ( DATA : in STD_LOGIC_VECTOR (3 downto 0);
           MSG : out STD_LOGIC_VECTOR (1 downto 0));
end Encoder;

architecture Behavioral of Encoder is
begin
process (DATA)
begin
if      (DATA="1000") then MSG<="11";
elsif (DATA="0100")  then MSG<="10";
elsif (DATA="0010")  then MSG<="01";
elsif (DATA="0001")  then MSG<="00";
else                  MSG<="XX";
end if;
end process;
end Behavioral;

```

### SIMULATION OUTPUT:



### 2. VHDL PROGRAM FOR 2:4 DECODER USING BEHAVIORAL MODELING

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

```

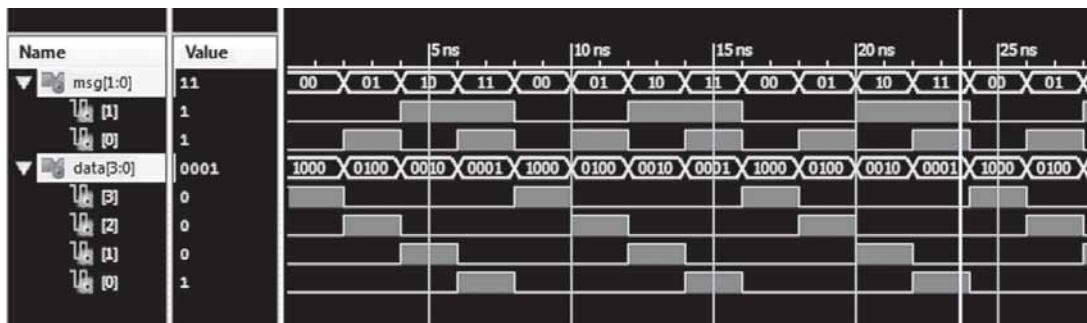
```

entity Decoder is
    Port (  MSG : in  STD_LOGIC_VECTOR (1 downto 0);
            DATA : out  STD_LOGIC_VECTOR (3 downto 0));
end Decoder;

architecture Behavioral of Decoder is
begin
process (MSG)
begin
if      (MSG="00") then DATA<="1000";
elsif  (MSG="01") then DATA<="0100";
elsif  (MSG="10") then DATA<="0010";
elsif  (MSG="11") then DATA<="0001";
else                 DATA<="XXXX";
end if;
end process;
end Behavioral;

```

### SIMULATION OUTPUT:



### 3. VHDL PROGRAM TO ENCODE A SEQUENCE WITH 4:2 ENCODER AND DECODE WITH 2:4 DECODER USING BEHAVIORAL MODELING

```

entity ENCODE_DECODE is
    Port ( DATA : in  STD_LOGIC_VECTOR (3 downto 0);
           MSG : out  STD_LOGIC_VECTOR (1 downto 0);
           DATA_OUT : out  STD_LOGIC_VECTOR (3 downto 0));
end ENCODE_DECODE;

architecture Behavioral of ENCODE_DECODE is

component Encoder is
    Port ( DATA : in  STD_LOGIC_VECTOR (3 downto 0);
           MSG : out  STD_LOGIC_VECTOR (1 downto 0));
end component;

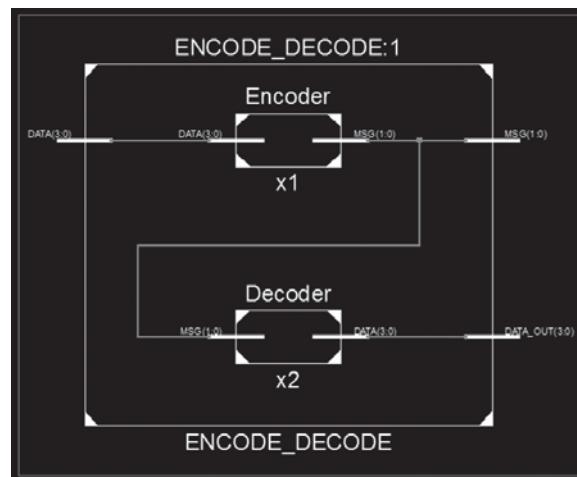
```

## 24 FUNDAMENTALS OF DIGITAL CIRCUITS

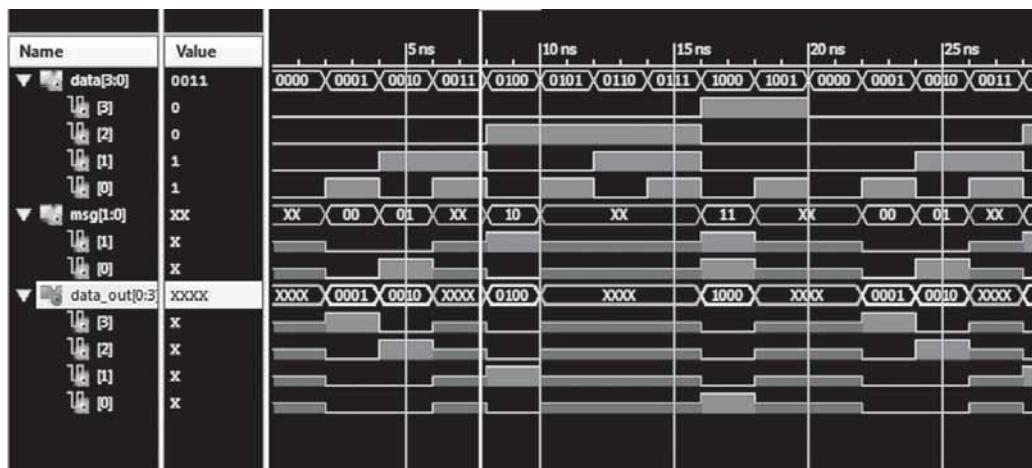
```
component Decoder is
    Port (  MSG : in  STD_LOGIC_VECTOR (1 downto 0);
            DATA : out  STD_LOGIC_VECTOR (3 downto 0));
end component;

SIGNAL N1:STD_LOGIC_VECTOR (1 downto 0);
begin
MSG  <= N1;
x1:Encoder port map(DATA,N1);
x2:Decoder port map(N1,DATA_OUT);
end Behavioral;
```

### RTL SCHEMATIC:



### SIMULATION OUTPUT:

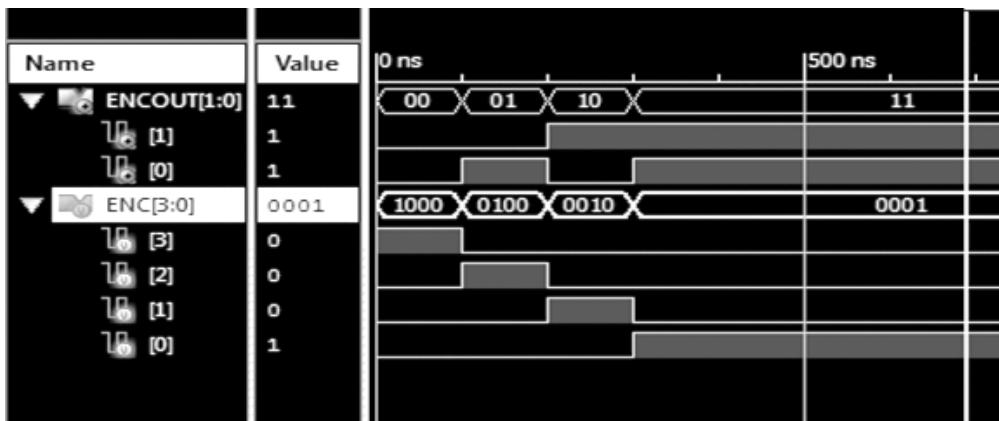


## VERILOG PROGRAMS

### 1. VERILOG PROGRAM FOR 4:2 ENCODER USING BEHAVIORAL MODELING

```
module ENCODER(input [3:0] ENC, output reg [1:0] ENCOUNT);
always@ (ENC)
case (ENC)
4'b1000:ENCOUT=2'b00;
4'b0100:ENCOUT=2'b01;
4'b0010:ENCOUT=2'b10;
4'b0001:ENCOUT=2'b11;
default:ENCOUT=2'bxx;
endcase
endmodule
```

### SIMULATION OUTPUT:



### 2. VERILOG PROGRAM FOR 2:4 DECODER USING BEHAVIORAL MODELING

```
module DECODER(input [1:0] DEC, output reg [3:0] DECOUNT);
always@ (DEC)
case (DEC)
2'b00:DECOUT=4'b0001;
2'b01:DECOUT=4'b0010;
2'b10:DECOUT=4'b0100;
2'b11:DECOUT=4'b1000;
default:DECOUT=4'bxxxx;
endcase
endmodule
```

## 26 FUNDAMENTALS OF DIGITAL CIRCUITS

### SIMULATION OUTPUT:



### 3. VERILOG PROGRAM TO ENCODE A SEQUENCE WITH 4:2 ENCODER AND DECODE WITH 2:4 DECODER USING BEHAVIORAL MODELING

#### MAIN PROGRAM:

```
module TOP(input [3:0] ENC, output [1:0] ENCOUNT, output [3:0] DECOUT);
  ENCODER E1(ENC, ENCOUNT);
  DECODER D1(ENCOUT, DECOUT);
endmodule
```

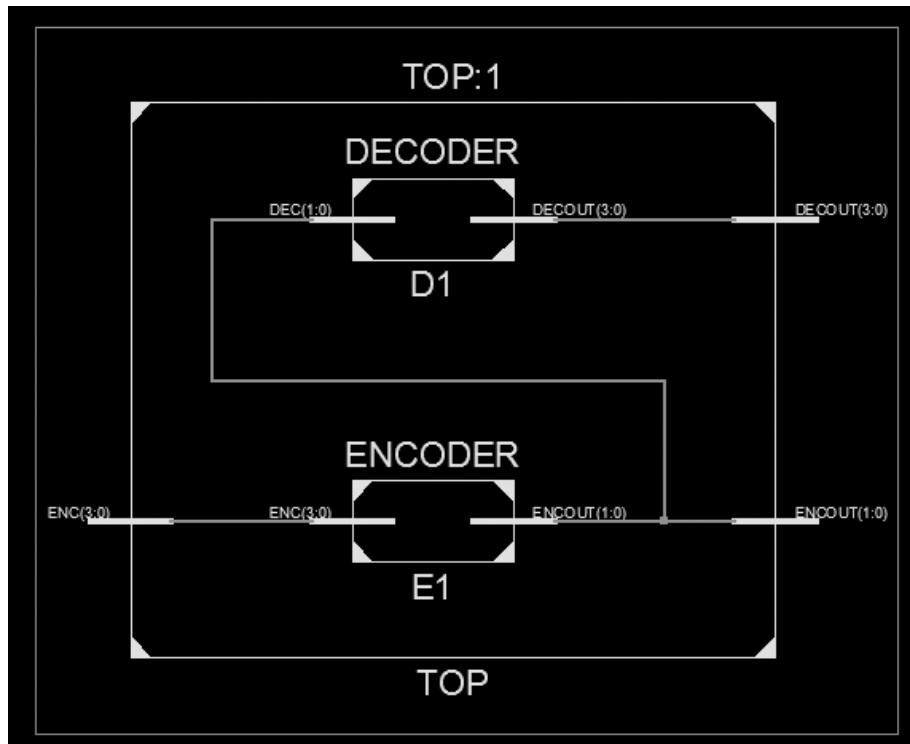
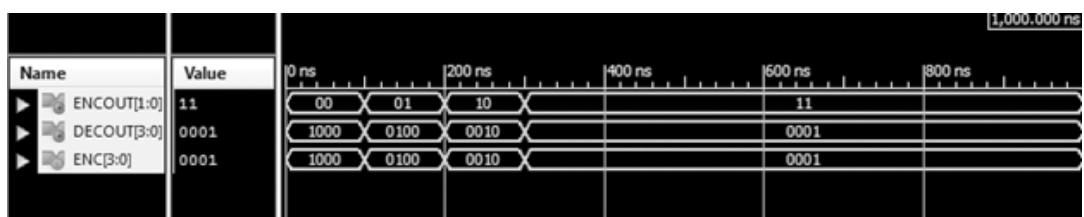
#### SUB PROGRAM:

```
module ENCODER(input [3:0] ENC, output reg [1:0] ENCOUNT);
  always@(ENC)
    case(ENC)
      4'b1000: ENCOUNT=2'b00;
      4'b0100: ENCOUNT=2'b01;
      4'b0010: ENCOUNT=2'b10;
      4'b0001: ENCOUNT=2'b11;
      default: ENCOUNT=2'bxx;
    endcase
  endmodule
```

#### SUB PROGRAM:

```
module DECODER(input [1:0] DEC, output reg [3:0] DECOUT);
  always@(DEC)
    case(DEC)
      2'b00: DECOUT=4'b1000;
      2'b01: DECOUT=4'b0100;
      2'b10: DECOUT=4'b0010;
      2'b11: DECOUT=4'b0001;
```

```
default:DECOUT=4'bxxxx;
endcase
endmodule
```

**RTL SCHEMATIC:****SIMULATION OUTPUT:**

# 2

## NUMBER SYSTEMS

### 2.1 THE DECIMAL NUMBER SYSTEM

We begin our study of the number systems with the familiar decimal number system. The decimal system contains ten unique symbols, 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Since counting in decimal involves ten symbols, we say that its *base* or *radix* is ten. There is no symbol for its base, i.e. for ten. It is a positional weighted system. It means that the value attached to a symbol depends on its location with respect to the decimal point. In this system, any number (integer, fraction, or mixed) of any magnitude can be represented by the use of these ten symbols only. Each symbol in the number is called a *digit*.

The leftmost digit in any number representation, which has the greatest positional weight out of all the digits present in that number, is called the most significant digit (MSD) and the rightmost digit, which has the least positional weight out of all the digits present in that number, is called the least significant digit (LSD). The digits on the left side of the decimal point form the integer part of a decimal number and those on the right side form the fractional part. The digits to the right of the decimal point have weights which are negative powers of 10 and the digits to the left of the decimal point have weights which are positive powers of 10.

The value of a decimal number is the sum of the products of the digits of that number with their respective column weights. The weight of each column is 10 times greater than the weight of the column to its right.

The first digit to the left of the decimal point has a weight of unity or  $10^0$ , the second digit to the left of the decimal point has a weight of 10 or  $10^1$ , the third has a weight of 100 or  $10^2$ , and so on. The first digit to the right of the decimal point has a weight of  $1/10$  or  $10^{-1}$ , the second digit to the right has a weight of  $1/100$  or  $10^{-2}$ , the third has a weight of  $1/1000$  or  $10^{-3}$ , and so on.

In general, the value of any mixed decimal number

$$d_n d_{n-1} d_{n-2} \dots d_1 d_0 \cdot d_{-1} d_{-2} d_{-3} \dots d_{-k}$$

is given by

$$(d_n \times 10^n) + (d_{n-1} \times 10^{n-1}) + \dots + (d_1 \times 10^1) + (d_0 \times 10^0) + (d_{-1} \times 10^{-1}) + (d_{-2} \times 10^{-2}) + \dots$$

Consider the decimal number 9256.26 using digits 2, 5, 6, 9. This is a mixed number. Hence,

$$9256.26 = 9 \times 1000 + 2 \times 100 + 5 \times 10 + 6 \times 1 + 2 \times (1/10) + 6 \times (1/100)$$

$$= 9 \times 10^3 + 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 6 \times 10^{-2}$$

Consider another number 6592.69, using the same digits 2, 5, 6, 9. Here,

$$6592.69 = 6 \times 10^3 + 5 \times 10^2 + 9 \times 10^1 + 2 \times 10^0 + 6 \times 10^{-1} + 9 \times 10^{-2}$$

Note the difference in position values of the same digits, when placed in different positions.

### 2.1.1 9's and 10's Complements

Subtraction of decimal numbers can be accomplished by the 9's and 10's complement methods similar to the 1's and 2's complement methods of binary. The 9's complement of a decimal number is obtained by subtracting each digit of that decimal number from 9. The 10's complement of a decimal number is obtained by adding a 1 to its 9's complement.

**EXAMPLE 2.1** Find the 9's complement of the following decimal numbers.



### *Solution*

$$\begin{array}{r}
 \text{(a)} \quad 9 \ 9 \ 9 \ 9 \\
 - 3 \ 4 \ 6 \ 5 \\
 \hline
 6 \ 5 \ 3 \ 4 \ (\text{9's complement of } 3465)
 \end{array}$$

$$\begin{array}{r}
 (b) \quad 9\ 9\ 9 \ . \ 9\ 9 \\
 - 7\ 8\ 2 \ . \ 5\ 4 \\
 \hline
 2\ 1\ 7 \ . \ 4\ 5 \text{ (9's complement of 782.54)}
 \end{array}$$

$$\begin{array}{r}
 (c) \quad 9\ 9\ 9\ 9\ .\ 9\ 9\ 9 \\
 - 4\ 5\ 2\ 6\ .\ 0\ 7\ 5 \\
 \hline
 5\ 4\ 7\ 3\ .\ 9\ 2\ 4 \text{ (9's complement of 4526.075)}
 \end{array}$$

**EXAMPLE 2.2** Find the 10's complement of the following decimal numbers.

- (a) 4069 (b) 1056.074

### *Solution*

$$\begin{array}{r}
 \text{(a) } \begin{array}{r} 9 & 9 & 9 & 9 \\ - 4 & 0 & 6 & 9 \\ \hline 5 & 9 & 3 & 0 \end{array} & \text{(9's complement of 4069)} \\
 & + 1 & \text{Add 1} \\
 \hline
 & 5 & 9 & 3 & 1 & \text{(10's complement of 4069)}
 \end{array}$$

$$\begin{array}{r}
 (b) \quad 9\ 9\ 9\ 9 \ . \ 9\ 9\ 9 \\
 - 1\ 0\ 5\ 6 \ . \ 0\ 7\ 4 \\
 \hline
 8\ 9\ 4\ 3 \ . \ 9\ 2\ 5 \quad (9\text{'s complement of } 1056.074) \\
 \qquad \qquad + 1 \qquad \qquad \text{Add 1} \\
 \hline
 8\ 9\ 4\ 3 \ . \ 9\ 2\ 6 \quad (10\text{'s complement of } 1056.074)
 \end{array}$$

### 2.1.2 9's Complement Method of Subtraction

To perform decimal subtraction using the 9's complement method, obtain the 9's complement of the subtrahend and add it to the minuend. Call this number the intermediate result. If there is a carry, it indicates that the answer is positive. Add the carry to the LSD of this result to get the answer. This is called end around carry. If there is no carry, it indicates that the answer is negative and the intermediate result is its 9's complement. Take the 9's complement of this result and place a negative sign in front to get the answer.

**EXAMPLE 2.3** Subtract the following numbers using the 9's complement method.

- (a)  $745.81 - 436.62$  (b)  $436.62 - 745.81$

**Solution**

$$\begin{array}{r}
 \begin{array}{r} 745 . 81 \\ - 436 . 62 \\ \hline 309 . 19 \end{array} & \Rightarrow & \begin{array}{r} 745 . 81 \\ + 563 . 37 \quad (\text{9's complement of } 436.62) \\ \hline \textcircled{1} 309 . 18 \quad (\text{Intermediate result}) \\ \Downarrow \qquad + 1 \quad (\text{End around carry}) \\ \hline 309 . 19 \quad (\text{Answer}) \end{array}
 \end{array}$$

The carry indicates that the answer is positive. So the answer is  $+309.19$ .

$$\begin{array}{r}
 \begin{array}{r} 436 . 62 \\ - 745 . 81 \\ \hline -309 . 19 \end{array} & \Rightarrow & \begin{array}{r} 436 . 62 \\ + 254 . 18 \quad (\text{9's complement of } 745.81) \\ \hline 690 . 80 \quad (\text{Intermediate result with no carry}) \end{array}
 \end{array}$$

There is no carry indicating that the answer is negative. So, take the 9's complement of the intermediate result and put a minus sign. The 9's complement of  $690.80$  is  $309.19$ .

Therefore, the answer is  $-309.19$ .

### 2.1.3 10's Complement Method of Subtraction

To perform decimal subtraction using the 10's complement method, obtain the 10's complement of the subtrahend and add it to the minuend. If there is a carry, ignore it. The presence of the carry indicates that the answer is positive; the result obtained is itself the answer. If there is no carry, it indicates that the answer is negative and the result obtained is its 10's complement. Obtain the 10's complement of the result and place a negative sign in front to get the answer.

**EXAMPLE 2.4** Subtract the following numbers using the 10's complement method.

- (a)  $2928.54 - 416.73$  (b)  $416.73 - 2928.54$

**Solution**

$$\begin{array}{r}
 \begin{array}{r} 2928 . 54 \\ - 0416 . 73 \\ \hline 2511 . 81 \end{array} & \Rightarrow & \begin{array}{r} 2928 . 54 \\ + 9583 . 27 \quad (10's complement of 416.73) \\ \hline \textcircled{1} 2511 . 81 \quad (\text{Ignore the carry}) \end{array}
 \end{array}$$

There is a carry indicating that the answer is positive. Ignore the carry.

The answer is  $2511.81$ .

$$(b) \begin{array}{r} 0\ 4\ 1\ 6\ .\ 7\ 3 \\ - 2\ 9\ 2\ 8\ .\ 5\ 4 \\ \hline - 2\ 5\ 1\ 1\ .\ 8\ 1 \end{array} \Rightarrow \begin{array}{r} 0\ 4\ 1\ 6\ .\ 7\ 3 \\ + 7\ 0\ 7\ 1\ .\ 4\ 6 \quad (10's \text{ complement of } 2928.54) \\ \hline 7\ 4\ 8\ 8\ .\ 1\ 9 \quad (\text{No carry}) \end{array}$$

There is no carry indicating that the answer is negative. So, take the 10's complement of the intermediate result and put a minus sign. The 10's complement of 7488.19 is 2511.81. Therefore, the answer is  $-2511.81$ .

## 2.2 THE BINARY NUMBER SYSTEM

The binary number system is a positional weighted system. The base or radix of this number system is 2. Hence, it has two independent symbols. The base itself cannot be a symbol. The symbols used are 0 and 1. A binary digit is called a *bit*. A binary number consists of a sequence of bits, each of which is either a 0 or a 1. The binary point separates the integer and fraction parts. Each digit (bit) carries a weight based on its position relative to the binary point. The weight of each bit position is one power of 2 greater than the weight of the position to its immediate right. The first bit to the left of the binary point has a weight of  $2^0$  and that column is called the *units column*. The second bit to the left has a weight of  $2^1$  and it is in the 2's column. The third bit to the left has a weight of  $2^2$  and it is in the 4's column, and so on. The first bit to the right of the binary point has a weight of  $2^{-1}$  and it is said to be in the  $1/2$ 's column, the next right bit with a weight of  $2^{-2}$  is in the  $1/4$ 's column, and so on.

The decimal value of the binary number is the sum of the products of all its bits multiplied by the weights of their respective positions. In general, a binary number with an integer part of  $(n+1)$  bits and a fraction part of  $k$  bits can be written as

$$d_n d_{n-1} d_{n-2} \dots d_1 d_0 \cdot d_{-1} d_{-2} d_{-3} \dots d_{-k}$$

Its decimal equivalent is

$$(d_n \times 2^n) + (d_{n-1} \times 2^{n-1}) + \dots + (d_1 \times 2^1) + (d_0 \times 2^0) + (d_{-1} \times 2^{-1}) + (d_{-2} \times 2^{-2}) + \dots$$

In general, the decimal equivalent of the number  $d_n d_{n-1} \dots d_1 d_0 \cdot d_{-1} d_{-2} \dots$  in any number system with base  $b$  is given by

$$(d_n \times b^n) + (d_{n-1} \times b^{n-1}) + \dots + (d_1 \times b^1) + (d_0 \times b^0) + (d_{-1} \times b^{-1}) + (d_{-2} \times b^{-2}) + \dots$$

The binary number system is used in digital computers because the switching circuits used in these computers use two-state devices such as transistors, diodes, etc. A transistor can be OFF or ON, a switch can be OPEN or CLOSED, a diode can be OFF or ON, etc. These devices have to exist in one of the two possible states. So, these two states can be represented by the symbols 0 and 1, respectively.

### 2.2.1 Counting in Binary

Counting in binary is very much similar to decimal counting as shown in Table 2.1. Start counting with 0, the next count is 1. We have now exhausted all symbols; therefore we put a 1 in the column to the left and continue to get 10, 11. Thus, 11 is the maximum we can count using two bits. So, put a 1 in the next column to the left and continue counting; we can count 100, 101, 110, 111. The

## 32 FUNDAMENTALS OF DIGITAL CIRCUITS

largest number we can count using three bits is 111. Put a 1 to the left and continue; we get, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111. The maximum count we can get using four bits is 1111. Continue counting with 5, 6, . . . bits as shown in Table 2.1.

**Table 2.1** Counting in binary

Decimal number	Binary number	Decimal number	Binary number
0	0	20	1 0 1 0 0
1	1	21	1 0 1 0 1
2	1 0	22	1 0 1 1 0
3	1 1	23	1 0 1 1 1
4	1 0 0	24	1 1 0 0 0
5	1 0 1	25	1 1 0 0 1
6	1 1 0	26	1 1 0 1 0
7	1 1 1	27	1 1 0 1 1
8	1 0 0 0	28	1 1 1 0 0
9	1 0 0 1	29	1 1 1 0 1
10	1 0 1 0	30	1 1 1 1 0
11	1 0 1 1	31	1 1 1 1 1
12	1 1 0 0	32	1 0 0 0 0 0
13	1 1 0 1	33	1 0 0 0 0 1
14	1 1 1 0	34	1 0 0 0 1 0
15	1 1 1 1	35	1 0 0 0 1 1
16	1 0 0 0 0	36	1 0 0 1 0 0
17	1 0 0 0 1	37	1 0 0 1 0 1
18	1 0 0 1 0	38	1 0 0 1 1 0
19	1 0 0 1 1	39	1 0 0 1 1 1

An easy way to remember to write a binary sequence of  $n$  bits is:

- The rightmost column in the binary number begins with a 0 and alternates between 0 and 1.
- The second column begins with  $2 (= 2^1)$  zeros and alternates between the groups of 2 zeros and 2 ones.
- The third column begins with  $4 (= 2^2)$  zeros and alternates between the groups of 4 zeros and 4 ones.
- The  $n$ th column begins with  $2^{n-1}$  zeros and alternates between the groups of  $2^{n-1}$  zeros and  $2^{n-1}$  ones.

### 2.2.2 Binary to Decimal Conversion

Binary numbers may be converted to their decimal equivalents by the positional weights method. In this method, each binary digit of the number is multiplied by its position weight and the product terms are added to obtain the decimal number.

**EXAMPLE 2.5** Convert  $10101_2$  to decimal.

**Solution**

(Positional weights)  $2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0$

$$\begin{aligned}
 (\text{Binary number}) \quad 1 & \ 0 \quad 1 \ 0 \quad 1 = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \\
 &= 16 + 0 + 4 + 0 + 1 \\
 &= 21_{10}
 \end{aligned}$$

**EXAMPLE 2.6** Convert  $11011.101_2$  to decimal.

### *Solution*

$$\begin{array}{ccccccccc}
 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} \\
 1 & 1 & 0 & 1 & 1 \cdot 1 & 0 & & \\
 & & & & & & 1 = (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\
 & & & & & & & + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) \\
 & & & & & & & = 16 + 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125 \\
 & & & & & & & = 27.625_{10}
 \end{array}$$

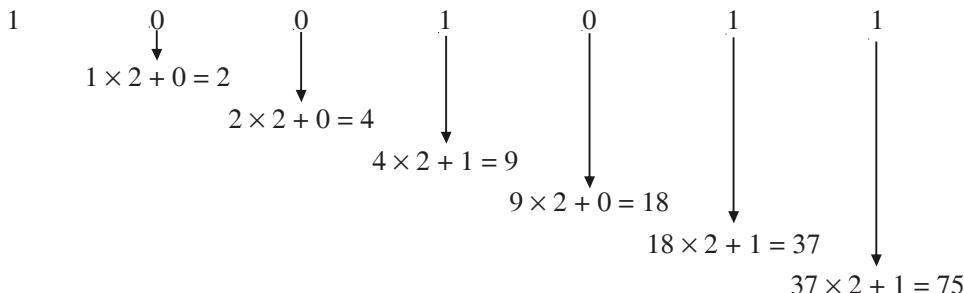
**EXAMPLE 2.7** Convert  $1001011_2$  to decimal.

### *Solution*

An integer binary number can also be converted to an integer decimal number as follows. Starting with the extreme left bit, i.e. MSB, multiply this bit by 2 and add the product to the next bit to the right.

Multiply the result obtained in the previous step by 2 and add the product to the next bit to the right.

Continue this process as shown below till all the bits in the number are exhausted.



The above steps can be written down as follows:

Copy down the extreme left bit, i.e. MSB = 1

Multiply by 2, add the next bit  $(1 \times 2) + 0 = 2$

Multiply by 2, add the next bit  $(2 \times 2) + 0 = 4$

Multiply by 2, add the next bit

Multiply by 2, add the next bit  $(9 \times 2) + 0 = 18$

Multiply by 2, add the next bit  $(18 \times 2) + 1 = 37$

Multiply by 2, add the next bit  $(37 \times 2) + 1 = 75$

The result is,  $1\ 0\ 0\ 1\ 0\ 1\ 1_2 = 75$

## Decimal to Bin

There are two methods to convert a decimal

**First method:** In this method, which is normally used only for small numbers, the values of

various powers of 2 need to be remembered. For conversion of large numbers, we should have a

## 34 FUNDAMENTALS OF DIGITAL CIRCUITS

table of powers of 2. In this method, known as the *sum-of-weights* method, the set of binary weight values whose sum is equal to the decimal number is determined. To convert a given decimal integer number to binary, first obtain the largest decimal number which is a power of 2 not exceeding the given decimal number and record it. Subtract this number from the given number and obtain the remainder. Once again obtain the largest decimal number which is a power of 2 not exceeding this remainder and record it. Subtract this number from the remainder to obtain the next remainder. Repeat the above process on the successive remainders till you get a 0 remainder. The sum of these powers of 2 expressed in binary is the binary equivalent of the original decimal number. In a similar manner, this sum-of-weights method can also be applied to convert decimal fractions to binary. To convert a decimal mixed number to binary, convert the integer and fraction parts separately into binary. In general, the sum-of-weights method can be used to convert a decimal number to a number in any other base system.

**Second method:** In this method, the decimal integer number is converted to the binary integer number by successive division by 2, and the decimal fraction is converted to binary fraction by successive multiplication by 2. This is also known as the *double-dabble* method. In the successive division-by-2 method, the given decimal integer number is successively divided by 2 till the quotient is zero. The last remainder is the MSB. The remainders read from bottom to top give the equivalent binary integer number. In the successive multiplication-by-2 method, the given decimal fraction and the subsequent decimal fractions are successively multiplied by 2, till the fraction part of the product is 0 or till the desired accuracy is obtained. The first integer obtained is the MSB. Thus, the integers read from top to bottom give the equivalent binary fraction. To convert a mixed number to binary, convert the integer and fraction parts separately to binary and then combine them.

In general, these methods can be used for converting a decimal number to an equivalent number in any other base system by just replacing 2 by the base  $b$  of that number system. That is, any decimal number can be converted to an equivalent number in any other base system by the sum-of-weights method, or by the double-dabble method—repeated division-by- $b$  for integers and repeated multiplication-by- $b$  for fractions.

**EXAMPLE 2.8** Convert  $163.875_{10}$  to binary.

**Solution**

The given decimal number is a mixed number. We, therefore, convert its integer and fraction parts separately.

*The integer part is  $163_{10}$*

The largest number, which is a power of 2, not exceeding 163 is 128.

$$128 = 2^7 = 1000000_2$$

The remainder is

$$163 - 128 = 35$$

The largest number, which a power of 2, not exceeding 35 is 32.

$$32 = 2^5 = 100000_2$$

The remainder is

$$35 - 32 = 3$$

The largest number, which is a power of 2, not exceeding 3 is 2.

$$2 = 2^1 = 10_2$$

The remainder is

$$3 - 2 = 1$$

$$1 = 2^0 = 1_2$$

Therefore,

$$163_{10} = 10000000_2 + 100000_2 + 10_2 + 1_2 = 10100011_2$$

*The fraction part is 0.875*

The largest fraction, which is a power of 2, not exceeding 0.875 is 0.5.

$$0.5 = 2^{-1} = 0.100_2$$

The remainder is

$$0.875 - 0.5 = 0.375$$

The largest fraction, which is a power of 2, not exceeding 0.375 is 0.25.

$$0.25 = 2^{-2} = 0.01_2$$

The remainder is

$$0.375 - 0.25 = 0.125$$

The largest fraction, which is a power of 2, not exceeding 0.125 is 0.125 itself.

$$0.125 = 2^{-3} = 0.001_2$$

Therefore,

$$0.875_{10} = 0.100_2 + 0.01_2 + 0.001_2 = 0.111_2$$

The final result is,

$$163.875_{10} = 10100011.111_2.$$

**EXAMPLE 2.9** Convert  $52_{10}$  to binary using the double-dabble method.

### Solution

We divide the given decimal number successively by 2 and read the remainders upwards to get the equivalent binary number.

Successive division	Remainder
2   52	
2   26	0
2   13	0
2   6	1
2   3	↑ 0
2   1	1
2   0	1

Reading the remainders from bottom to top, the result is  $52_{10} = 110100_2$ .

## 36 FUNDAMENTALS OF DIGITAL CIRCUITS

**EXAMPLE 2.10** Convert  $0.75_{10}$  to binary using the double-dabble method.

**Solution**

Multiply the given fraction by 2. Keep the integer in the product as it is and multiply the new fraction in the product by 2. Continue this process and read the integers in the products from top to bottom.

<i>Given fraction</i>	0.75
Multiply 0.75 by 2	1.50
Multiply 0.50 by 2	↓ 1.00

Reading the integers from top to bottom,  $0.75_{10} = 0.11_2$ .

**EXAMPLE 2.11** Convert  $105.15_{10}$  to binary.

**Solution**

*Conversion of integer 105*

<i>Successive division</i>	<i>Remainder</i>
2   105	
2   52	1
2   26	0
2   13	0
2   6	1
2   3	↑ 0
2   1	1
0	1

Reading the remainders from bottom to top,  $105_{10} = 1101001_2$ .

*Conversion of fraction  $0.15_{10}$*

<i>Given fraction</i>	0.15
Multiply 0.15 by 2	0.30
Multiply 0.30 by 2	0.60
Multiply 0.60 by 2	↓ 1.20
Multiply 0.20 by 2	0.40
Multiply 0.40 by 2	0.80
Multiply 0.80 by 2	1.60

This particular fraction can never be expressed exactly in binary. This process may be terminated after a few steps.

Reading the integers from top to bottom,  $0.15_{10} = 0.001001_2$ .

Therefore, the final result is,  $105.15_{10} = 1101001.001001_2$ .

### 2.2.4 Binary Addition

The rules for binary addition are the following:

$$0 + 0 = 0; \quad 0 + 1 = 1; \quad 1 + 0 = 1; \quad 1 + 1 = 10, \text{ i.e. } 0 \text{ with a carry of } 1.$$

**EXAMPLE 2.12** Add the binary numbers 1101.101 and 111.011.

*Solution*

8 4 2 1	2 <sup>-1</sup>	2 <sup>-2</sup>	2 <sup>-3</sup>	(Column numbers)
1 1 0 1	·	1	0	1
+	1 1 1	·	0	1
1 0 1 0 1 · 0 0 0				

In the 2 <sup>-3</sup> 's column	1 + 1 = 0, with a carry of 1 to the 2 <sup>-2</sup> column
In the 2 <sup>-2</sup> 's column	0 + 1 + 1 = 0, with a carry of 1 to the 2 <sup>-1</sup> column
In the 2 <sup>-1</sup> 's column	1 + 0 + 1 = 0, with a carry of 1 to the 1's column
In the 1's column	1 + 1 + 1 = 1, with a carry of 1 to the 2's column
In the 2's column	0 + 1 + 1 = 0, with a carry of 1 to the 4's column
In the 4's column	1 + 1 + 1 = 1, with a carry of 1 to the 8's column
In the 8's column	1 + 1 = 0, with a carry of 1 to the 16's column

### 2.2.5 Binary Subtraction

The binary subtraction is performed in a manner similar to that in decimal subtraction. The rules for binary subtraction are:

$$0 - 0 = 0; \quad 1 - 1 = 0; \quad 1 - 0 = 1; \quad 0 - 1 = 1, \text{ with a borrow of } 1.$$

**EXAMPLE 2.13** Subtract 111.111<sub>2</sub> from 1010.01<sub>2</sub>.

*Solution*

8 4 2 1	2 <sup>-1</sup>	2 <sup>-2</sup>	2 <sup>-3</sup>	(Column numbers)
1 0 1 0	·	0	1	0
1 1 1 · 1 1 1				
0 0 1 0 · 0 1 1				

In the 2<sup>-3</sup> column, a 1 cannot be subtracted from a 0. So, borrow a 1 from the 2<sup>-2</sup> column making the 2<sup>-2</sup> column 0. The 1 borrowed from the 2<sup>-2</sup> column becomes 10 in the 2<sup>-3</sup> column. Therefore, in the 2<sup>-3</sup> column, 10 - 1 = 1.

In the 2<sup>-2</sup> column, a 1 cannot be subtracted from a 0. So, borrow a 1 from the 2<sup>-1</sup> column, but it is also a 0. So, borrow a 1 from the 1's column. That is also a 0, so borrow a 1 from the 2's column making the 2's column 0. This 1 borrowed from the 2's column becomes 10 in the 1's column. Keep one 1 in the 1's column, bring the other 1 to the 2<sup>-1</sup> column, which becomes 10 in this column. Keep one 1 in the 2<sup>-1</sup> column and bring the other 1 to the 2<sup>-2</sup> column, which becomes 10 in this column. Therefore,

In the 2 <sup>-2</sup> 's column	10 - 1 = 1
In the 2 <sup>-1</sup> 's column	1 - 1 = 0
In the 1's column	1 - 1 = 0

## 38 FUNDAMENTALS OF DIGITAL CIRCUITS

Now, in the 2's column, a 1 cannot be subtracted from a 0; so, borrow a 1 from the 4's column. But the 4's column has a 0. So, borrow a 1 from the 8's column, making the 8's column 0, and bring it to the 4's column. It becomes 10 in the 4's column. Keep one 1 in the 4's column and bring the second 1 to the 2's column making it 10 in the 2's column. Therefore,

$$\begin{array}{ll} \text{In the 2's column} & 10 - 1 = 1 \\ \text{In the 4's column} & 1 - 1 = 0 \\ \text{In the 8's column} & 0 - 0 = 0 \end{array}$$

Hence, the result is  $0010.011_2$ .

### 2.2.6 Binary Multiplication

There are two methods of binary multiplication—the *paper* method and the *computer* method. Both the methods obey the following multiplication rules:

$$0 \times 0 = 0; \quad 1 \times 1 = 1; \quad 1 \times 0 = 0; \quad 0 \times 1 = 0$$

The paper method is similar to the multiplication of decimal numbers on paper. Multiply the multiplicand with each bit of the multiplier, and add the partial products. The partial product is the same as the multiplicand if the multiplier bit is a 1 and is zero if the multiplier bit is a 0.

**EXAMPLE 2.14** Multiply  $1101_2$  by  $110_2$ .

**Solution**

The LSB of the multiplier is a 0. So, the first partial product is a 0. The next two bits of the multiplier are 1s. So, the next two partial products are equal to the multiplicand itself. The sum of the partial products gives the answer.

$$\begin{array}{r} 1101 \\ \times 110 \\ \hline 0000 \\ 1101 \\ 1101 \\ \hline 1001110 \end{array}$$

**EXAMPLE 2.15** Multiply  $1011.101_2$  by  $101.01_2$ .

**Solution**

$$\begin{array}{r} 1011.101 \\ \times 101.01 \\ \hline 1011101 \\ 0000000 \\ 1011101 \\ 0000000 \\ \hline 1011101 \\ 1111010001 \end{array}$$

### 2.2.7 Computer Method of Multiplication

A computer can add only two numbers at a time with a carry. So, the paper method cannot be used by the digital computer. Besides, the computer has only a fixed number of bit positions available. A  $p$ -bit number multiplied by another  $p$ -bit number yields a  $2p$ -bit number. To multiply a  $p$ -bit number by another  $p$ -bit number, a  $p$ -bit multiplicand register and a  $2p$ -bit multiplier/result register are required. The multiplier is placed in the left  $p$ -bits of the multiplier/quotient (MQ) register and the right  $p$ -bits are set to zero. This permits the MQ to be used as a dual register, which holds the multiplier and the partial (then final) results. The multiplicand is placed in the M register. The MQ register is then shifted one bit to the left. If a 1 is shifted out, M is added to the MQ register. If a 0 is shifted out, a 0 is added to the MQ register. The entire process is repeated  $p$  (the number of bits in the multiplier) times. The result appears in the MQ register.

**EXAMPLE 2.16** Multiply  $1100_2$  by  $1001_2$  using the computer method.

**Solution**

The computer method of multiplication is performed as shown below.

MQ register	1 0 0 1 0 0 0 0	
Shift MQ left	1 0 0 1 0 0 0 0 0	(A 1 is shifted out. So, add M to MQ)
Add M	1 1 0 0	
Partial sum in MQ	0 0 1 0 1 1 0 0	
Shift MQ left	0 0 1 0 1 1 0 0 0	(A 0 is shifted out. So, add 0 to MQ)
Add 0	0 0 0 0	
Partial sum in MQ	0 1 0 1 1 0 0 0	
Shift MQ left	0 1 0 1 1 0 0 0 0	(A 0 is shifted out. So, add 0 to MQ)
Add 0	0 0 0 0	
Partial sum in MQ	1 0 1 1 0 0 0 0	
Shift MQ left	1 0 1 1 0 0 0 0 0	(A 1 is shifted out. So, add M to MQ)
Add M	1 1 0 0	
Final sum in MQ	0 1 1 0 1 1 0 0	

### 2.2.8 Binary Division

Like multiplication, division too can be performed by two methods—the paper method and the computer method. In the paper method, long-division procedures similar to those in decimal are used.

**EXAMPLE 2.17** Divide  $101101_2$  by  $110_2$ .

**Solution**

Divisor 110 cannot go in the first three bits of the dividend, i.e. in 101. So, consider the first 4 bits 1011 of the dividend. 110 can go in 1011, one time with a remainder of 101. Next, 110 can go in 1010, one time with a remainder of 100. Next, 110 can go in 1001, one time with a remainder of 11. Finally, 110 can go in 110 with a remainder of 0.

## 40 FUNDAMENTALS OF DIGITAL CIRCUITS

$$\begin{array}{r} 110 ) 101101 ( 111.1 \\ \underline{110} \\ 1010 \\ \underline{110} \\ 1001 \\ \underline{110} \\ 110 \\ \underline{110} \\ 000 \end{array}$$

Therefore,  $101101 \div 110 = 111.1$

**EXAMPLE 2.18** Divide  $110101.11_2$  by  $101_2$ .

*Solution*

$$\begin{array}{r} 101 ) 110101.11 ( 1010.11 \\ \underline{101} \\ 0110 \\ \underline{101} \\ 111 \\ \underline{101} \\ 101 \\ \underline{101} \\ 000 \end{array}$$

Therefore,  $110101.11 \div 101 = 1010.11$

### 2.2.9 Computer Method of Division

The computer method of division involves successive subtraction. Suppose we divide an eight-bit dividend by a four-bit divisor; the quotient will be formed in the right-half of the MQ register and the remainder in the left-half. The dividend is first placed in the MQ register, and the divisor in the D register. The divisor is then subtracted from the dividend. The result is considered negative if a borrow is required, and positive if no borrow is required. If this result is positive, the quotient will be greater than four bits; so, an error has occurred. We cannot complete the division with those registers. If the result is negative, then the quotient will be four or less bits, sufficiently small to be contained in the right-half of the MQ register.

The MQ register is next shifted left one bit. The bit shifted out, a 0 or a 1 is not lost. It is stored in a carry flag, a single flip-flop, and is used for the subtraction following the left shift. If the result of subtraction is positive, a 1 is added to the LSB of the MQ register where the quotient is accumulated; if it is negative, the divisor is added to the MQ, and MQ is shifted left one bit. This effectively puts a 0 in this bit of the quotient. The process is continued until the MQ register has been shifted left four bits (the number of bits in the D register). The remainder is then in the left-half of the MQ register and the quotient in the right-half.

**EXAMPLE 2.19** Divide 32 by 5 in binary using the computer method.

*Solution*

Dividend = 32 = 100000 = 00100000

Divisor = 5 = 101 = 0101

*Computer division*

MQ	0 0 1 0	0 0 0 0	
Subtract D	0 1 0 1		
MQ	1 1 0 1	0 0 0 0	(Borrow is required; the result is negative; division is valid)
Add D	0 1 0 1		
MQ	0 0 1 0	0 0 0 0	(Original number)
Shift MQ left	0 0 1 0 0	0 0 0 0	
Subtract D	0 1 0 1		
MQ	1 1 1 1	0 0 0 0	(Result is negative)
Add D	0 1 0 1		
MQ	0 1 0 0	0 0 0 0	
Shift MQ left	0 1 0 0 0	0 0 0 0	
Subtract D	0 1 0 1		
MQ	0 0 1 1	0 0 0 0	(Result is positive)
Add 1		1	
MQ	0 0 1 1	0 0 0 1	
Shift MQ left	0 0 1 1 0	0 0 1 0	
Subtract D	0 1 0 1		
MQ	0 0 0 1	0 0 1 0	(Result is positive)
Add 1		1	
MQ	0 0 0 1	0 0 1 1	
Shift MQ left	0 0 0 1 0	0 1 1 0	
Subtract D	0 1 0 1		
MQ	1 1 0 1	0 1 1 0	(Result is negative)
Add D	0 1 0 1		
MQ	0 0 1 0	0 1 1 0	(Final answer)

Result: Remainder = 0 0 1 0 Quotient = 0 1 1 0

### 2.3 REPRESENTATION OF SIGNED NUMBERS AND BINARY ARITHMETIC IN COMPUTERS

So far, we have considered only positive numbers. The representation of negative numbers is also equally important. There are two ways of representing signed numbers—sign-magnitude form and complement form. There are two complement forms: 1's complement form and 2's complement form.

## 42 FUNDAMENTALS OF DIGITAL CIRCUITS

Most digital computers do subtraction by the 2's complement method, but some do it by the 1's complement method. The advantage of performing subtraction by the complement method is reduction in the hardware. Instead of having separate digital circuits for addition and subtraction, only adding circuits are needed. That is, subtraction is also performed by adders only. Instead of subtracting one number from the other, the complement of the subtrahend is added to the minuend.

In sign-magnitude form, an additional bit called the *sign bit* is placed in front of the number. If the sign bit is a 0, the number is positive. If it is a 1, the number is negative. For example,

0	1	0	1	0	0	1
Sign bit		Magnitude				
1	1	0	1	0	0	1
Sign bit		Magnitude				

Under the sign-magnitude system, a great deal of manipulation is necessary to add a positive number to a negative number. Thus, though the sign-magnitude number system is possible, it is impractical.

### 2.3.1 Representation of Signed Numbers Using the 2's (or 1's) Complement Method

The 2's (or 1's) complement system for representing signed numbers works like this:

1. If the number is positive, the magnitude is represented in its true binary form and a sign bit 0 is placed in front of the MSB.
2. If the number is negative, the magnitude is represented in its 2's (or 1's) complement form and a sign bit 1 is placed in front of the MSB.

That is, to represent the numbers in sign 2's (or 1's) complement form, determine the 2's (or 1's) complement of the magnitude of the number and then attach the sign bit.

The 2's (or 1's) complement operation on a signed number will change a positive number to a negative number and vice versa. The conversion of complement to true binary is the same as the process used to convert true binary to complement. The representation of + 51 and - 51 in both 2's and 1's complement forms is shown below:

0	1	1	0	0	1	1
Sign bit		Magnitude				
1	1	1	0	0	1	1
Sign bit		Magnitude				
1	0	0	1	1	0	1
Sign bit		Magnitude				
1	0	0	1	1	0	0
Sign bit		Magnitude				

**EXAMPLE 2.20** Each of the following numbers is a signed binary number. Determine the decimal value in each case, if they are in (i) sign-magnitude form, (ii) 2's complement form, and (iii) 1's complement form.

- (a) 01101                   (b) 010111                   (c) 10111                   (d) 1101010

**Solution**

Given number	Sign-magnitude form	2's complement form	1's complement form
0 1 1 0 1	+ 13	+ 13	+ 13
0 1 0 1 1 1	+ 23	+ 23	+ 23
1 0 1 1 1	- 7	- 9	- 8
1 1 0 1 0 1 0	- 42	- 22	- 21

To subtract using the sign 2's (or 1's) complement method, represent both the subtrahend and the minuend by the same number of bits. Take the 2's (or 1's) complement of the subtrahend including the sign bit. Keep the minuend in its original form and add the 2's (or 1's) complement of the subtrahend to it.

The choice of 0 for positive sign, and 1 for negative sign is not arbitrary. In fact, this choice makes it possible to add the sign bits in binary addition just as other bits are added. When the sign bit is a 0, the remaining bits represent magnitude, and when the sign bit is a 1, the remaining bits represent 2's or 1's complement of the number. The polarity of the signed number can be changed simply by performing the complement on the complete number.

Table 2.2 lists all possible 4-bit signed binary numbers in the three representations. The equivalent decimal number is also shown for reference. Note that the positive numbers in all three representations are identical and have 0 in the leftmost position. The signed 2's complement system has only one representation for 0, which is always positive. The other two systems have either a positive or a negative 0 which is something not encountered in ordinary arithmetic. Note that all negative numbers have a 1 in the leftmost bit position: This is the way we distinguish them from the positive numbers. With four bits, we can represent 16 binary numbers. In the signed magnitude and the 1's complement representations, there are eight positive numbers and eight negative numbers including two zeros. In the 2's complement representation, there are eight positive numbers including one zero and eight negative numbers.

**Special case in 2's complement representation:** Whenever a signed number has a 1 in the sign bit and all 0s for the magnitude bits, the decimal equivalent is  $-2^n$ , where  $n$  is the number of bits in the magnitude. For example, 1000 = -8 and 10000 = -16.

**Characteristics of the 2's complement numbers:** The 2's complement numbers have the following properties:

1. There is one unique zero.
2. The 2's complement of 0 is 0.
3. The leftmost bit cannot be used to express a quantity. It is a sign bit. If it is a 1, the number is negative and if it is a 0, the number is positive.

## 44 FUNDAMENTALS OF DIGITAL CIRCUITS

4. For an  $n$ -bit word which includes the sign bit, there are  $(2^{n-1} - 1)$  positive integers,  $2^{n-1}$  negative integers and one 0, for a total of  $2^n$  unique states.
5. Significant information is contained in the 1s of the positive numbers and 0s of the negative numbers.
6. A negative number may be converted into a positive number by finding its 2's complement.

**Table 2.2** Signed binary numbers

Decimal	Sign 2's complement form	Sign 1's complement form	Sign magnitude form
+7	0 1 1 1	0 1 1 1	0 1 1 1
+6	0 1 1 0	0 1 1 0	0 1 1 0
+5	0 1 0 1	0 1 0 1	0 1 0 1
+4	0 1 0 0	0 1 0 0	0 1 0 0
+3	0 0 1 1	0 0 1 1	0 0 1 1
+2	0 0 1 0	0 0 1 0	0 0 1 0
+1	0 0 0 1	0 0 0 1	0 0 0 1
+0	0 0 0 0	0 0 0 0	0 0 0 0
-0	—	1 1 1 1	1 0 0 0
-1	1 1 1 1	1 1 1 0	1 0 0 1
-2	1 1 1 0	1 1 0 1	1 0 1 0
-3	1 1 0 1	1 1 0 0	1 0 1 1
-4	1 1 0 0	1 0 1 1	1 1 0 0
-5	1 0 1 1	1 0 1 0	1 1 0 1
-6	1 0 1 0	1 0 0 1	1 1 1 0
-7	1 0 0 1	1 0 0 0	1 1 1 1
-8	1 0 0 0	—	—

**Methods of obtaining the 2's complement of a number:** The 2's complement of a number can be obtained in three ways as given below.

1. By obtaining the 1's complement of the given number (by changing all 0s to 1s and 1s to 0s) and then adding 1.
2. By subtracting the given  $n$ -bit number  $N$  from  $2^n$ .
3. Starting at the LSB, copying down each bit up to and including the first 1 bit encountered, and complementing the remaining bits.

**EXAMPLE 2.21** Express  $-45$  in 8-bit 2's complement form.

**Solution**

$+45$  in 8-bit form is 00101101.

*First method*

Obtain the 1's complement of 00101101 and then add 1.

Positive expression of the given number	0 0 1 0 1 1 0 1
1's complement of it	1 1 0 1 0 0 1 0
Add 1	<u>+ 1</u>
Thus, the 2's complement form of - 45 is	1 1 0 1 0 0 1 1

*Second method*

Subtract the given number  $N$  from  $2^n$

$$\begin{array}{rcl} 2^n & = & 1 0 0 0 0 0 0 0 0 \\ \text{Subtract } 45 & = & - \underline{0 0 1 0 1 1 0 1} \\ & & 1 1 0 1 0 0 1 1 \end{array}$$

Thus, the 2's complement form of - 45 is

*Third method*

Copy down the bits starting from LSB up to and including the first 1, and then complement the remaining bits.

Original number	0 0 1 0 1 1 0 1
Copy up to the first 1 bit	1
Complement the remaining bits	<u>1 1 0 1 0 0 1</u>
Thus, the 2's complement form of - 45 is	1 1 0 1 0 0 1 1

**EXAMPLE 2.22** Express -73.75 in 12-bit 2's complement form.

*Solution*

$$+ 73.75 = N = 01001001.1100$$

*First method*

Positive expression of the given number	0 1 0 0 1 0 0 1 . 1 1 0 0
1's complement of it	1 0 1 1 0 1 1 0 . 0 0 1 1
Add 1	<u>+ 1</u>
Thus, the 2's complement of -73.75 is	1 0 1 1 0 1 1 0 . 0 1 0 0

*Second method*

$$\begin{array}{rcl} 2^8 & = & 1 0 0 0 0 0 0 0 0 . 0 0 0 0 \\ \text{Subtract } 73.75 & = & - \underline{0 1 0 0 1 0 0 1 . 1 1 0 0} \\ & & 1 0 1 1 0 1 1 0 . 0 1 0 0 \end{array}$$

Thus, the 2's complement of -73.75 is

*Third method*

Original number	0 1 0 0 1 0 0 1 . 1 1 0 0
Copy up to the first 1 bit	1 0 0
Complement the remaining bits	<u>1 0 1 1 0 1 1 0 . 0</u>
Thus, -73.75 in 2's complement form is	1 0 1 1 0 1 1 0 . 0 1 0 0

### 2.3.2 2's Complement Arithmetic

The 2's complement system is used to represent negative numbers using modulus arithmetic. The word length of a computer is fixed. That means, if a 4-bit number is added to another 4-bit number,

## 46 FUNDAMENTALS OF DIGITAL CIRCUITS

the result will be only of 4 bits. Carry, if any, from the fourth bit will overflow. This is called the *modulus arithmetic*. For example:  $1100 + 1111 = 1011$ .

In the 2's complement subtraction, add the 2's complement of the subtrahend to the minuend. If there is a carry out, ignore it. Look at the sign bit, i.e. MSB of the sum term. If the MSB is a 0, the result is positive and is in true binary form. If the MSB is a 1 (whether there is a carry or no carry at all) the result is negative and is in its 2's complement form. Take its 2's complement to find its magnitude in binary.

**EXAMPLE 2.23** Subtract 14 from 46 using the 8-bit 2's complement arithmetic.

**Solution**

$$\begin{array}{rcl} +14 & = & 000001110 \\ -14 & = & 111100100 \quad (\text{In 2's complement form}) \end{array}$$

$$\begin{array}{rcl} +46 & = & 00101110 \\ -14 & \Rightarrow & +11110010 \quad (\text{2's complement form of } -14) \\ \hline +32 & = & \textcircled{1}00100000 \quad (\text{Ignore the carry}) \end{array}$$

There is a carry, ignore it. The MSB is 0. So, the result is positive and is in normal binary form. Therefore, the result is  $+00100000 = +32$ .

**EXAMPLE 2.24** Add  $-75$  to  $+26$  using the 8-bit 2's complement arithmetic.

**Solution**

$$\begin{array}{rcl} +75 & = & 01001011 \\ -75 & = & 10110101 \quad (\text{In 2's complement form}) \end{array}$$

$$\begin{array}{rcl} +26 & = & 00011010 \\ -75 & \Rightarrow & +10110101 \quad (\text{2's complement form of } -75) \\ \hline -49 & = & 11001111 \quad (\text{No carry}) \end{array}$$

There is no carry, the MSB is a 1. So, the result is negative and is in 2's complement form. The magnitude is 2's complement of 11001111, that is,  $00110001 = 49$ . Therefore, the result is  $-49$ .

**EXAMPLE 2.25** Add  $-45.75$  to  $+87.5$  using the 12-bit 2's complement arithmetic.

**Solution**

$$\begin{array}{rcl} +87.5 & = & 01010111.1000 \\ -45.75 & \Rightarrow & +11010010.0100 \quad (-45.75 \text{ in 2's complement form}) \\ +41.75 & = & \textcircled{1}00101001.1100 \quad (\text{Ignore the carry}) \end{array}$$

There is a carry, ignore it. The MSB is 0. So, the result is positive and is in normal binary form. Therefore, the result is  $+41.75$ .

**EXAMPLE 2.26** Add  $27.125$  to  $-79.625$  using the 12-bit 2's complement arithmetic.

**Solution**

$$\begin{array}{rcl} +27.125 & = & 00011011.0010 \\ -79.625 & \Rightarrow & +10110000.0110 \quad (-79.625 \text{ in 2's complement form}) \\ -52.500 & = & 11001011.1000 \quad (\text{No carry}) \end{array}$$

There is no carry. The MSB is a 1 indicating that the result is negative and is in its 2's complement form. The 2's complement of 11001011.1000 is 00110100.1000. Therefore, the result is - 52.5.

**EXAMPLE 2.27** Add - 31.5 to - 93.125 using the 12-bit 2's complement arithmetic.

**Solution**

$$\begin{array}{r} -93.125 \\ -31.500 \\ \hline -124.625 \end{array} \Rightarrow \begin{array}{r} 10100010.1110 \\ +11100000.1000 \\ \hline 10000011.0110 \end{array} \begin{array}{l} (-93.125 \text{ in 2's complement form}) \\ (-31.5 \text{ in 2's complement form}) \\ (\text{Ignore the carry}) \end{array}$$

There is a carry, ignore it. The MSB is a 1. So, the result is negative and is in its 2's complement form. The 2's complement of 1000 0011.0110 is 0111 1100.1010. Therefore, the result is - 124.625.

**EXAMPLE 2.28** Add 47.25 to 55.75 using the 2's complement method.

**Solution**

$$\begin{array}{r} 47.25 \\ 55.75 \\ \hline 103.00 \end{array} \Rightarrow \begin{array}{r} 00101111.0100 \\ +00110111.1100 \\ \hline 01100111.0000 \end{array} \begin{array}{l} (\text{No carry}) \end{array}$$

There is no carry. The MSB is a 0. Therefore, the result is positive and is in its true binary form. Hence, it is equal to + 103.0.

**EXAMPLE 2.29** Add + 40.75 to - 40.75 using the 12-bit 2's complement arithmetic.

**Solution**

$$\begin{array}{r} +40.75 \\ -40.75 \\ \hline 00.00 \end{array} \Rightarrow \begin{array}{r} 00101000.1100 \\ +11010111.0100 \\ \hline 00000000.0000 \end{array} \begin{array}{l} (-40.75 \text{ in 2's complement form}) \\ (\text{Ignore the carry}) \end{array}$$

There is a carry, ignore it. The result is 0.

**Method of obtaining the 1's complement of a number:** The 1's complement of a number is obtained by simply complementing each bit of the number, that is, by changing all the 0s to 1s and all the 1s to 0s. We can also say that the 1's complement of a number is obtained by subtracting each bit of the number from 1. This complemented value represents the negative of the original number. This system is very easy to implement in hardware by simply feeding all bits through inverters. One of the difficulties of using 1's complement is its representation of zero. Both 00000000 and its 1's complement 11111111 represent zero. The 00000000 is called *positive zero* and the 11111111 is called *negative zero*.

**EXAMPLE 2.30** Represent - 99 and - 77.25 in 8-bit 1's complement form.

**Solution**

We first write the positive representation of the given number in binary form and then complement each of its bits to represent the negative of the number.

- (a)  $+99 = 01100011$   
 $-99 = 10011100$  (In 1's complement form)
- (b)  $+77.25 = 01001101.0100$   
 $-77.25 = 10110010.1011$  (In 1's complement form)

### 2.3.3 1's Complement Arithmetic

In 1's complement subtraction, add the 1's complement of the subtrahend to the minuend. If there is a carry out, bring the carry around and add it to the LSB. This is called the *end around carry*. Look at the sign bit (MSB). If this is a 0, the result is positive and is in true binary. If the MSB is a 1 (whether there is a carry or no carry at all), the result is negative and is in its 1's complement form. Take its 1's complement to get the magnitude in binary.

**EXAMPLE 2.31** Subtract 14 from 25 using the 8-bit 1's complement arithmetic.

**Solution**

$$\begin{array}{r}
 25 \\
 -14 \\
 \hline
 +11
 \end{array} \Rightarrow \begin{array}{r}
 00011001 \\
 +11110001 \\
 \hline
 \textcircled{1}00001010
 \end{array} \quad \begin{array}{l} \text{(In 1's complement form)} \\ \text{(End around carry)} \end{array}$$

+ 1

The MSB is a 0. So, the result is positive and is in pure binary. Therefore, the result is,  $00001011 = +11_{10}$ .

**EXAMPLE 2.32** Add  $-25$  to  $+14$  using the 8-bit 1's complement method.

**Solution**

$$\begin{array}{r}
 +14 \\
 -25 \\
 \hline
 -11
 \end{array} \Rightarrow \begin{array}{r}
 00001110 \\
 +11100110 \\
 \hline
 11110100
 \end{array} \quad \begin{array}{l} \text{(In 1's complement form)} \\ \text{(No carry)} \end{array}$$

There is no carry. The MSB is a 1. So, the result is negative and is in its 1's complement form. The 1's complement of  $11110100$  is  $00001011$ . The result is, therefore,  $-11_{10}$ .

**EXAMPLE 2.33** Add  $-25$  to  $-14$  using the 8-bit 1's complement method.

**Solution**

$$\begin{array}{r}
 -25 \\
 -14 \\
 \hline
 -39
 \end{array} \Rightarrow \begin{array}{r}
 11100110 \\
 +11110001 \\
 \hline
 \textcircled{1}11010111
 \end{array} \quad \begin{array}{l} \text{(In 1's complement form)} \\ \text{(In 1's complement form)} \\ \text{(End around carry)} \end{array}$$

+ 1

The MSB is a 1. So, the result is negative and is in its 1's complement form. The 1's complement of  $11101011$  is  $00100111$ . Therefore, the result is  $-39$ .

**EXAMPLE 2.34** Add  $+25$  to  $+14$  using the 8-bit 1's complement arithmetic.

**Solution**

$$\begin{array}{r}
 +25 \\
 +14 \\
 \hline
 +39
 \end{array} \Rightarrow \begin{array}{r}
 00011001 \\
 +00001110 \\
 \hline
 00100111
 \end{array} \quad \begin{array}{l} \text{(In 1's complement form)} \\ \text{(In 1's complement form)} \end{array}$$

There is no carry. The MSB is a 0. So, the result is positive and is in pure binary. Therefore, the result is,  $00100111 = +39$ .

**EXAMPLE 2.35** Add + 25 to – 25 using the 8-bit 1's complement method.

**Solution**

$$\begin{array}{r}
 + 25 \\
 - 25 \\
 \hline
 00011001 \\
 \Rightarrow + 11100110 \quad (\text{In 1's complement form}) \\
 \hline
 11111111
 \end{array}$$

There is no carry. The MSB is a 1. So, the result is negative and is in its 1's complement form. The 1's complement of 11111111 is 00000000. Therefore, the result is – 0.

**EXAMPLE 2.36** Subtract 27.50 from 68.75 using the 12 bit 1's complement arithmetic.

**Solution**

$$\begin{array}{r}
 + 68.75 \\
 - 27.50 \\
 \hline
 + 11100100.0111 \\
 \Rightarrow \begin{array}{r}
 01000100.1100 \\
 \textcircled{1} 00101001.0011 \\
 \swarrow \qquad \qquad \qquad + 1 \\
 \hline
 00101001.0100
 \end{array} \quad (\text{End around carry})
 \end{array}$$

The MSB is a 0. So, the result is positive and is in its normal binary form. Therefore, the result is + 41.25.

**EXAMPLE 2.37** Add – 89.75 to + 43.25 using the 12-bit 1's complement method.

**Solution**

$$\begin{array}{r}
 + 43.25 \\
 - 89.75 \\
 \hline
 - 10100110.0011 \\
 \Rightarrow \begin{array}{r}
 00101011.0100 \\
 + 10100110.0011 \\
 \hline
 11010001.0111
 \end{array} \quad (\text{In 1's complement form})
 \end{array}$$

There is no carry. The MSB is a 1. So, the result is negative and is in its 1's complement form. The 1's complement of 11010001.0111 is 00101110.1000. Therefore, the result is – 46.50.

**EXAMPLE 2.38** Write the following binary numbers in sign magnitude form, in sign 1's complement form, and in sign 2's complement form using 16-bit registers.

- (a) + 1001010
- (b) – 11110000
- (c) – 11001100.1
- (d) + 100000011.111

**Solution**

The representation of positive numbers in sign magnitude form, in sign 1's complement form, and in sign 2's complement form is exactly the same. The first bit 0 indicates the positive sign and the remaining bits indicate the magnitude.

To write the negative number in 16-bit sign magnitude form, put a 1 in the MSB position and write the magnitude in 15-bit form.

To write the negative number in sign 1's complement form, put a 1 in the MSB position and write the 1's complement of the 15-bit magnitude in the other 15-bit positions.

## 50 FUNDAMENTALS OF DIGITAL CIRCUITS

To write the negative number in sign 2's complement form, put a 1 in the MSB position and write the 15-bit 2's complement form of the magnitude in the other 15-bit positions.

(a) The given number + 1 0 0 1 0 1 0 in 15-bit form is + 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0

In 16-bit sign magnitude form it is 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0

In 16-bit sign 1's complement form it is 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0

In 16-bit sign 2's complement form it is 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0

(b) The given number - 1 1 1 1 0 0 0 0 in 15-bit form is - 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0

In 16-bit sign magnitude form it is 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0

In 16-bit sign 1's complement form it is 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1

In 16-bit sign 2's complement form it is 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0

(c) The given number - 1 1 0 0 1 1 0 0 . 1 in 15-bit form is - 0 0 0 1 1 0 0 1 1 0 0 . 1 0 0 0

In 16-bit sign magnitude form it is 1 0 0 0 1 1 0 0 1 1 0 0 . 1 0 0 0

In 16-bit sign 1's complement form it is 1 1 1 1 0 0 1 1 0 0 1 1 . 0 1 1 1

In 16-bit sign 2's complement form it is 1 1 1 1 0 0 1 1 0 0 1 1 . 1 0 0 0

(d) The given number + 1 0 0 0 0 0 0 1 1 . 1 1 1 in 15-bit form is 0 0 1 0 0 0 0 0 1 1 . 1 1 1 0

In 16-bit sign magnitude form it is 0 0 0 1 0 0 0 0 0 0 1 1 . 1 1 1 0

In 16-bit sign 1's complement form it is 0 0 0 1 0 0 0 0 0 0 1 1 . 1 1 1 0

In 16-bit sign 2's complement form it is 0 0 0 1 0 0 0 0 0 0 1 1 . 1 1 1 0

**EXAMPLE 2.39** Perform N1 + N2, N1 + (-N2) for the following 8-bit numbers expressed in 2's complement representation. Verify your answers by using decimal addition and subtraction

$$(a) \quad N1 = 0 0 1 1 0 0 1 0, \quad N2 = 1 1 1 1 1 1 0 1$$

$$(b) \quad N1 = 1 0 0 0 1 1 1 0, \quad N2 = 0 0 0 0 1 1 0 1$$

### Solution

Since the given 8-bit numbers are in 2's complement form, if the MSB is a 0, the number is positive and the remaining bits give the magnitude of the positive number. If the MSB is a 1, the number is negative and 2's complement of the remaining bits give the magnitude of the negative number. So, perform the addition and subtraction using 2's complement method. The arithmetic operations and the verification of the answers using decimal addition and subtraction are performed as given below.

$$\begin{array}{rcl} (a) \text{ Given} & N1 = 0 0 1 1 0 0 1 0_2 & = + 0 1 1 0 0 1 0 = + 5 0_{10}, \quad \text{and} \\ & N2 = 1 1 1 1 1 1 0 1_2 & = - 0 0 0 0 0 1 1 = - 3_{10} \\ & N1 + N2 & = \end{array}$$

0 0 1 1 0 0 1 0	+ 5 0 +
+ 1 1 1 1 1 1 0 1	- 3
<hr/>	
1 0 0 1 0 1 1 1	= + 4 7
+ 4 7	

There is a carry. Ignore it. The MSB is a 0. So, answer is positive and the remaining bits indicate the magnitude in normal binary. It is + 47.

$$\begin{array}{rcl} N1 + (-N2) & = & 0 0 1 1 0 0 1 0 \quad + 5 0 + \\ & & + 0 0 0 0 0 0 1 1 \quad + 3 \\ & \hline & 0 0 1 1 0 1 0 1 \quad = + 5 3 \quad + 5 3 \end{array}$$

There is no carry. The MSB is a 0. So, the answer is positive and is in true binary form. It is + 53.

(b) Given       $N1 = 10001110 = -1110010 = -114_{10}$   
 $N2 = 00001101 = +0001101 = +13_{10}$   
 $N1 + N2 = \begin{array}{r} 10001110 \\ + 00001101 \\ \hline 10011011 \end{array} = -101_{10}$

There is no carry. The MSB is a 1. So, the answer is negative. Take its 2's complement and put minus sign. So it is  $-1100101_2 = -101_2$ .

$$\begin{array}{r} N1 + (-N2) = 10001110 \\ + 11110011 \\ \hline \textcircled{1} 10000001 = -127_{10} \end{array} \quad \begin{array}{r} -114+ \\ -13 \\ \hline -127 \end{array}$$

There is a carry. Ignore it. The MSB is a 1. So, the answer is negative. Take its 2's complement and put a minus sign. It is  $-0111111_2 = -127_{10}$ .

**EXAMPLE 2.40** (a) The binary numbers listed have a sign bit in the left most position and if negative, numbers are in 2's complement form. Perform the arithmetic operations indicated and verify the answers. (b) Repeat if the numbers are in 1's complement form.

- (i)  $101011 + 111000$
- (ii)  $001110 + 110010$
- (iii)  $111001 - 001010$
- (iv)  $101011 - 100110$

### Solution

(a) Given that the binary numbers listed have a sign bit in the MSB position and if negative, numbers are in 2's complement form. So, if MSB = 0, the number is positive and the remaining bits give its magnitude. If the MSB = 1, the number is negative and the remaining bits give the 2's complement of the magnitude. Therefore,

$$\begin{array}{ll} 101011 = -10101 = -21 & 111000 = -01000 = -8 \\ 001110 = +01110 = +14 & 110010 = -01110 = -14 \\ 111001 = -00111 = -7 & 001010 = +01010 = +10 \\ 101011 = -10101 = -21 & 100110 = -11010 = -26 \end{array}$$

The arithmetic operations in 2's complement form and their verifications are given below.

(i)  $\begin{array}{r} 101011 = -10101 = -21+ \\ + 111000 = -01000 = -8 \\ \hline \textcircled{1} 100011 = -29 \end{array}$

Both the augend and addend are negative and are in 2's complement form. In the sum term there is a carry. Ignore it. The sign bit is a 1. So, the answer is negative. Take its 2's complement and put a minus sign. Answer  $= -011101_2 = -29$ . The decimal addition is shown above.

(ii)  $\begin{array}{r} 001110 = +01110 = +14+ \\ + 110010 = -01110 = -14 \\ \hline \textcircled{1} 000000 = 0 \end{array}$

## 52 FUNDAMENTALS OF DIGITAL CIRCUITS

The augend is positive and the addend is negative and is in its 2's complement form. In the sum term there is a carry. Ignore it. The answer is all 0s. It is 0. The decimal addition is shown above.

$$(iii) \quad \begin{array}{r} 111001 \\ -001010 \\ \hline \textcircled{1} 101111 \end{array} \Rightarrow \begin{array}{r} 111001 \\ +110110 \\ \hline 101111 \end{array} = -00111 = -7 \\ = -01010 = \underline{-10} \\ = -17 = \underline{-17}$$

The minuend is negative and the subtrahend is positive. Take the 2's complement of the subtrahend and add it to the minuend. In the sum term there is a carry. Ignore it. The sign bit is a 1. So, the answer is negative. Take its 2's complement and put a minus sign. It is -17. The decimal subtraction is shown above.

$$(iv) \quad \begin{array}{r} 101011 \\ -100110 \\ \hline \textcircled{1} 000101 \end{array} \Rightarrow \begin{array}{r} 101011 \\ +011010 \\ \hline 000101 \end{array} = -10101 = \underline{-21} \\ = -11010 = \underline{-26} \\ = +5 = \underline{+5}$$

Both the minuend and the subtrahend are negative and are in 2's complement form. Since the subtrahend is to be subtracted from the minuend, take the 2's complement of the subtrahend and add it to the minuend. In the sum term there is a carry. Ignore it. The sign bit is a 0. So, the answer is positive and is in true binary form. It is +5. The decimal subtraction is shown above.

(b) Given that the binary numbers listed have a sign bit and if negative, numbers are in 1's complement form. So, if MSB = 0, the number is positive and the remaining bits give its magnitude. If the MSB = 1, the number is negative and the remaining bits give the 1's complement of the magnitude. Therefore,

$$\begin{array}{ll} 101011 = -10100 = -20 & 111000 = -00111 = -7 \\ 001110 = +01110 = +14 & 110010 = -01101 = -13 \\ 111001 = -00110 = -6 & 001010 = +01010 = +10 \\ 101011 = -10100 = -20 & 100110 = -11001 = -25 \end{array}$$

The arithmetic operations in 1's complement form and their verifications are given below.

$$(i) \quad \begin{array}{r} 101011 = -10100 = -20 + \\ +111000 = -00111 = \underline{-7} \\ \hline \textcircled{1} 100011 \\ \Downarrow \\ \hline 100100 = -27 \end{array}$$

Both the augend and addend are negative and are in 1's complement form. In the sum term there is a carry. Add it to the LSB. The sign bit is a 1. So, the answer is negative. Take its 1's complement and put a minus sign. Answer =  $-11011_2 = -27$ . The decimal addition is also shown above.

$$(ii) \quad \begin{array}{r} 001110 = +01110 = +14 + \\ +110010 = -01101 = \underline{-13} \\ \hline \textcircled{1} 000000 \\ \Downarrow \\ \hline 000001 = +1 \end{array}$$

The augend is positive and the addend is negative and is in its 1's complement form. In the sum term there is a carry. Add it to the LSB. The sign bit is a 0. So, the answer is positive and is in true binary form. It is +1. The decimal addition is also shown above.

$$\begin{array}{rcl}
 \text{(iii)} & \begin{array}{r} 1\ 1\ 1\ 0\ 0\ 1 \\ - 0\ 0\ 1\ 0\ 1\ 0 \end{array} & \Rightarrow \begin{array}{rcl} 1\ 1\ 1\ 0\ 0\ 1 & = & -0\ 0\ 1\ 1\ 0 \\ + 1\ 1\ 0\ 1\ 0\ 1 & = & -0\ 1\ 0\ 1\ 0 \\ \hline \textcircled{1}\ 1\ 0\ 1\ 1\ 1\ 0 & & -10 \\ \swarrow & 1 & \\ \hline 1\ 0\ 1\ 1\ 1\ 1 & = & -16 \end{array}
 \end{array}$$

The minuend is negative and is in 1's complement form. The subtrahend is positive. Take the 1's complement of the subtrahend and add it to the minuend. In the sum term there is a carry. Add it to the LSB. The sign bit is a 1. So, the answer is negative. Take its 1's complement and put a minus sign. It is -16. The decimal subtraction is also shown above.

$$\begin{array}{rcl}
 \text{(iv)} & \begin{array}{r} 1\ 0\ 1\ 0\ 1\ 1 \\ - 1\ 0\ 0\ 1\ 1\ 0 \end{array} & \Rightarrow \begin{array}{rcl} 1\ 0\ 1\ 0\ 1\ 1 & = & -1\ 0\ 1\ 0\ 0 \\ + 0\ 1\ 1\ 0\ 0\ 1 & = & -1\ 1\ 0\ 0\ 1 \\ \hline \textcircled{1}\ 0\ 0\ 0\ 1\ 0\ 0 & & -25 \\ \swarrow & 1 & \\ \hline 0\ 0\ 0\ 1\ 0\ 1 & = & +5 \end{array}
 \end{array}$$

Both the minuend and the subtrahend are negative and are in 1's complement form. Take the 1's complement of the subtrahend and add it to the minuend. The sum term has a carry. Add it to the LSB. The sign bit is a 0. So, the answer is positive and is in normal binary form. It is +5. The decimal subtraction is also shown above.

**EXAMPLE 2.41** Perform the subtraction with the following unsigned binary numbers  
(a) by taking the 2's complement of the subtrahend and (b) by taking the 1's complement of the subtrahend.

- (i)  $11010 - 10000$
- (ii)  $11010 - 1101$
- (iii)  $100 - 11000$
- (iv)  $1010100 - 1010100$

#### Solution

(a) Subtraction of the given unsigned binary numbers using 2's complement method

$$\begin{array}{rcl}
 \text{(i)} & \begin{array}{r} 1\ 1\ 0\ 1\ 0 \\ - 1\ 0\ 0\ 0\ 0 \end{array} & \Rightarrow \begin{array}{rcl} 1\ 1\ 0\ 1\ 0 & = & 26 \\ + 1\ 0\ 0\ 0\ 0 & = & -16 \\ \hline \textcircled{1}\ 0\ 1\ 0\ 1\ 0 & & +10 \\ = & +0\ 1\ 0\ 1\ 0 = +10 & \text{(Result is positive = +10)} \end{array}
 \end{array}$$

There is a carry. Ignore it. The MSB is 0. Hence, the answer is positive and is in true binary form.

So it is  $+0\ 1\ 0\ 1\ 0 = +10$

## 54 FUNDAMENTALS OF DIGITAL CIRCUITS

$$\begin{array}{r}
 \text{(ii)} \quad 1\ 1\ 0\ 1\ 0 \Rightarrow 1\ 1\ 0\ 1\ 0 = 26 \\
 - 1\ 1\ 0\ 1 \quad + 1\ 0\ 0\ 1\ 1 = -13 \\
 \hline
 \text{Ignore the carry, result is positive} = +13 \quad +13 \\
 = +0\ 1\ 1\ 0\ 1 = +13
 \end{array}$$

There is a carry. Ignore it. The MSB is 0. Hence, the answer is positive and is in true binary form.

So it is  $+0\ 1\ 1\ 0\ 1 = +13$

$$\begin{array}{r}
 \text{(iii)} \quad 1\ 0\ 0 \Rightarrow 0\ 0\ 0\ 0\ 1\ 0\ 0 \Rightarrow 0\ 0\ 0\ 0\ 1\ 0\ 0 = 4 \\
 - 1\ 1\ 0\ 0\ 0\ 0 \quad + 0\ 1\ 1\ 0\ 0\ 0\ 0 = -48 \\
 \hline
 \text{Result is negative} = -44 \\
 = -0\ 1\ 0\ 1\ 1\ 0\ 0 = -44
 \end{array}$$

There is no carry. The MSB is 1. Hence, the answer is negative. Take its 2's complement and put a minus sign. So, it is  $-1\ 0\ 1\ 1\ 0\ 0 = -44$

$$\begin{array}{r}
 \text{(iv)} \quad 1\ 0\ 1\ 0\ 1\ 0\ 0 \Rightarrow 1\ 0\ 1\ 0\ 1\ 0\ 0 = 84 \\
 - 1\ 0\ 1\ 0\ 1\ 0\ 0 \quad + 1\ 0\ 1\ 0\ 1\ 0\ 0 = -84 \\
 \hline
 \text{Ignore the carry} = 0 \quad (\text{Result} = 0)
 \end{array}$$

There is a carry. Ignore it. The MSB is 0. Hence, the answer is positive and is in true binary form.

So it is  $+0\ 0\ 0\ 0\ 0\ 0 = 0$ .

(b) *Subtraction of the given unsigned binary numbers using 1's complement method*

$$\begin{array}{r}
 \text{(i)} \quad 1\ 1\ 0\ 1\ 0 \Rightarrow 1\ 1\ 0\ 1\ 0 = 26 \\
 - 1\ 0\ 0\ 0\ 0 \quad + 0\ 1\ 1\ 1\ 1 = -16 \\
 \hline
 \text{Ignore the carry} = +10 \quad (\text{End around carry}) \\
 0\ 1\ 0\ 1\ 0 = +10 \quad (\text{Result is positive} = +10)
 \end{array}$$

There is a carry. Add it to the LSB. The MSB is 0. Hence the answer is positive and is in true binary form.

So it is  $+1\ 0\ 1\ 0 = +10$

$$\begin{array}{r}
 \text{(ii)} \quad 1\ 1\ 0\ 1\ 0 \Rightarrow 1\ 1\ 0\ 1\ 0 = 26 \\
 - 1\ 1\ 0\ 1 \quad + 1\ 0\ 0\ 1\ 0 = -13 \\
 \hline
 \text{Add carry} = 1 \quad +13 \\
 0\ 1\ 1\ 0\ 1 \quad (\text{Result is positive} = +13)
 \end{array}$$

There is a carry. Add it to the LSB. The MSB is 0. Hence, the answer is positive and is in true binary form.

So it is  $+1\ 1\ 0\ 1 = +13$

$$\begin{array}{r}
 \text{(iii)} \quad \begin{array}{r} 1\ 0\ 0 \\ -1\ 1\ 0\ 0\ 0\ 0 \end{array} \Rightarrow \begin{array}{r} 0\ 0\ 0\ 0\ 1\ 0\ 0 \\ +1\ 0\ 0\ 1\ 1\ 1\ 1 \end{array} = 4 \\
 \hline
 \begin{array}{r} 1\ 0\ 1\ 0\ 0\ 1\ 1 \\ \text{(Result is negative)} \end{array} = -44 \\
 = -1\ 0\ 1\ 1\ 0\ 0 \quad (= -44)
 \end{array}$$

There is no carry. The MSB is 1. Hence, the answer is negative. Take the 1's complement of the remaining bits and put a minus sign. So, it is  $-101100 = -44$ .

$$\begin{array}{r}
 \text{(iv)} \quad \begin{array}{r} 1\ 0\ 1\ 0\ 1\ 0\ 0 \\ -1\ 0\ 1\ 0\ 1\ 0\ 0 \end{array} \Rightarrow \begin{array}{r} 1\ 0\ 1\ 0\ 1\ 0\ 0 \\ +0\ 1\ 0\ 1\ 0\ 1\ 1 \end{array} = 84 \\
 \hline
 \begin{array}{r} 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ \text{(Result is negative} = -0) \end{array} = 0
 \end{array}$$

There is no carry. The MSB is 1. Hence, the answer is negative. Take the 1's complement of the remaining bits and put a minus sign. So, it is  $-000000 = 0$ .

### 2.3.4 Double Precision Numbers

For any computer the word length is fixed. In a 16-bit computer, that is, in a computer with a 16-bit word length, only numbers from  $+2^{16-1} - 1$  ( $+32,767$ ) to  $-2^{16-1}$  ( $-32,768$ ) can be expressed in each register. If numbers greater than this are to be expressed, two storage locations need to be used, that is, each such number has to be stored in two registers. This is called *double precision*. Leaving the MSB which is the sign bit, this allows a 31-bit number length with two 16-bit registers. If still larger numbers are to be expressed, three registers are used to store each number. This is called *triple precision*.

### 2.3.5 Floating Point Numbers

In the decimal system, very large and very small numbers are expressed in scientific notation, by stating a number (mantissa) and an exponent of 10. Examples are  $6.53 \times 10^{-27}$  and  $1.58 \times 10^{21}$ . Binary numbers can also be expressed in the same notation by stating a number and an exponent of 2. However, the format for a computer may be as shown below.

Mantissa	Exponent
0110000000	100101

In this machine, the 16-bit word consists of two parts, a 10-bit mantissa and a 6-bit exponent. The mantissa is in 2's complement form. So, the MSB can be thought of as the sign bit. The binary point is assumed to be to the right of this sign bit. The 6 bits of the exponent could represent 0 through 63. However, to express negative exponents, the number  $32_{10}$  ( $100000_2$ ) has been added to the desired exponent. The actual exponent of the number, therefore, is equal to the exponent given by the 6 bits minus 32. This is a common system used in floating point formats. It is called the excess-32 notation. According to these definitions, the floating-point number shown above is

The mantissa portion	$= +0 . 110000000$
The exponent portion	$= 100101$
The actual exponent	$= 100101 - 100000 = 000101$
The entire number	$= N = +0.1100_2 \times 2^5 = 11000_2 = 24_{10}$

## 56 FUNDAMENTALS OF DIGITAL CIRCUITS

There are many formats of floating point numbers, each computer having its own. Some use two words for the mantissa, and one for the exponent; others use one and half words for the mantissa, and a half word for the exponent. On some machines the programmer can select from several formats, depending on the accuracy desired. Some use excess- $n$  notation for the exponent, some use 2's complement, some even use sign-magnitude for both the mantissa and the exponent.

## 2.4 THE OCTAL NUMBER SYSTEM

The octal number system was extensively used by early minicomputers. It is also a positional weighted system. Its base or radix is 8. It has 8 independent symbols 0, 1, 2, 3, 4, 5, 6, and 7. Since its base  $8 = 2^3$ , every 3-bit group of binary can be represented by an octal digit. An octal number is, thus, 1/3rd the length of the corresponding binary number.

### 2.4.1 Usefulness of the Octal System

The ease with which conversions can be made between octal and binary makes the octal system more attractive as a *shorthand* means of expressing large binary numbers. In computer work, binary numbers with up to 64-bits are not uncommon. These binary numbers do not always represent a numerical quantity; they often represent some type of code, which conveys non-numerical information. In computers, binary numbers might represent (a) the actual numerical data, (b) the numbers corresponding to a location (address) in memory, (c) an instruction code, (d) a code expressing alphabetic and other non-numerical characters, or (e) a group of bits representing the status of devices internal or external to the computer.

When dealing with large binary numbers of many bits, it is convenient and more efficient for us to write the numbers in octal rather than binary. However, the digital circuits and systems work strictly in binary; we use octal only for the convenience of the operators of the system. Table 2.3 shows octal counting.

Table 2.3 Octal counting

Decimal number	Octal number						
0	0	10	12	20	24	30	36
1	1	11	13	21	25	31	37
2	2	12	14	22	26	32	40
3	3	13	15	23	27	33	41
4	4	14	16	24	30	34	42
5	5	15	17	25	31	35	43
6	6	16	20	26	32	36	44
7	7	17	21	27	33	37	45
8	10	18	22	28	34	38	46
9	11	19	23	29	35	39	47

### 2.4.2 Octal to Binary Conversion

To convert a given octal number to a binary, just replace each octal digit by its 3-bit binary equivalent.

**EXAMPLE 2.42** Convert  $367.52_8$  to binary.

**Solution**

Given octal number is

Convert each octal digit to binary

The result is

$$\begin{array}{cccccc} 3 & 6 & 7 & . & 5 & 2 \\ 011 & 110 & 111 & . & 101 & 010 \\ & & & & 01110111 & . 101010_2 \end{array}$$

### 2.4.3 Binary to Octal Conversion

To convert a binary number to an octal number, starting from the binary point make groups of 3 bits each, on either side of the binary point, and replace each 3-bit binary group by the equivalent octal digit.

**EXAMPLE 2.43** Convert  $110101.101010_2$  to octal.

**Solution**

Groups of three bits are

Convert each group to octal

The result is

$$\begin{array}{ccccc} 110 & 101 & . & 101 & 010 \\ 6 & 5 & . & 5 & 2 \\ & & & 65.52_8 \end{array}$$

**EXAMPLE 2.44** Convert  $10101111001.0111_2$  to octal.

**Solution**

Groups of three bits are

$$\begin{array}{cccccc} 10 & 101 & 111 & 001 & . & 011 & 1 \\ = 010 & 101 & 111 & 001 & . & 011 & 100 \end{array}$$

Convert each group into octal

The result is

$$\begin{array}{ccccc} 2 & 5 & 7 & 1 & . & 3 & 4 \\ & & & & & 2571.34_8 \end{array}$$

### 2.4.4 Octal to Decimal Conversion

To convert an octal number to a decimal number, multiply each digit in the octal number by the weight of its position and add all the product terms.

The decimal value of the octal number  $d_n d_{n-1} d_{n-2} \dots d_0 d_{-1} d_{-2} \dots d_{-k}$  is

$$(d_n \times 8^n) + (d_{n-1} \times 8^{n-1}) + \dots + (d_1 \times 8^1) + (d_0 \times 8^0) + (d_{-1} \times 8^{-1}) + (d_{-2} \times 8^{-2}) + \dots$$

**EXAMPLE 2.45** Convert  $4057.06_8$  to decimal.

**Solution**

$$\begin{aligned} 4057.06_8 &= 4 \times 8^3 + 0 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 + 0 \times 8^{-1} + 6 \times 8^{-2} \\ &= 2048 + 0 + 40 + 7 + 0 + 0.0937 \\ &= 2095.0937_{10} \end{aligned}$$

### 2.4.5 Decimal to Octal Conversion

To convert a mixed decimal number to a mixed octal number, convert the integer and fraction parts separately.

To convert the given decimal integer number to octal, successively divide the given number by 8 till the quotient is 0. The last remainder is the MSD. The remainders read upwards give the equivalent octal integer number.

## 58 FUNDAMENTALS OF DIGITAL CIRCUITS

To convert the given decimal fraction to octal, successively multiply the decimal fraction and the subsequent decimal fractions by 8 till the product is 0 or till the required accuracy is obtained. The first integer from the top is the MSD. The integers to the left of the octal point read downwards give the octal fraction.

**EXAMPLE 2.46** Convert  $378.93_{10}$  to octal.

**Solution**

*Conversion of  $378_{10}$  to octal*

<i>Successive division</i>	<i>Remainders</i>
8   378	
8   47	↑ 2
8   5	7
0	5

Read the remainders from bottom to top. Therefore,  $378_{10} = 572_8$ .

*Conversion of  $0.93_{10}$  to octal*

$0.93 \times 8$	7.44
$0.44 \times 8$	3.52
$0.52 \times 8$	↓ 4.16
$0.16 \times 8$	1.28

Read the integers to the left of the octal point downwards.

Therefore,  $0.93_{10} = 0.7341_8$ . Hence  $378.93_{10} = 572.7341_8$ .

Conversion of large decimal numbers to binary and large binary numbers to decimal can be conveniently and quickly performed via octal as shown below.

**EXAMPLE 2.47** Convert  $5497_{10}$  to binary.

**Solution**

Since the given decimal number is large, we first convert this number to octal and then convert the octal number to binary.

<i>Successive division</i>	<i>Remainders</i>
8   5497	
8   687	↑ 1
8   85	7
8   10	5
8   1	2
0	1

Therefore,  $5497_{10} = 12571_8 = 001010101111001_2$ .

**EXAMPLE 2.48** Convert  $101111010001_2$  to decimal.

**Solution**

Since the given binary number is large, we first convert this number to octal and then convert the octal number to decimal.

$$101111010001_2 = 5721_8 = 5 \times 8^3 + 7 \times 8^2 + 2 \times 8^1 + 1 \times 8^0 = 2560 + 448 + 16 + 1 = 3025_{10}.$$

#### 2.4.6 Octal Arithmetic

Octal arithmetic rules are similar to the decimal or binary arithmetic. Normally we are not interested in performing octal arithmetic operations using octal representation of numbers. This number system is normally used to enter long strings of binary data in a digital system like a microcomputer. This makes the task of entering binary data in a microcomputer easier. Arithmetic operations can be performed by converting the octal numbers to binary numbers and then using the rules of binary arithmetic. Octal subtraction can be performed using 1's complement method or 2's complement method. However octal subtraction can also be performed directly by 7's and 8's complement methods similar to the 9's and 10's complement methods of decimal system.

**EXAMPLE 2.49** Add  $(27.5)_8$  and  $(74.4)_8$ .

**Solution**

$$\begin{array}{rcl} 27.5_8 & = & 0\ 1\ 0\ \quad 1\ 1\ 1\ \cdot\ 1\ 0\ 1_2 \\ + 74.4_8 & = & + 1\ 1\ 1\ \quad 1\ 0\ 0\ \cdot\ 1\ 0\ 0_2 \\ \hline 124.1_8 & = & 1\ 0\ 1\ 0\ \quad 1\ 0\ 0\ \cdot\ 0\ 0\ 1 \end{array}$$

**EXAMPLE 2.50** Subtract (a)  $45_8$  from  $66_8$  and (b)  $73_8$  from  $25_8$ .

**Solution**

Using 8-bit representation and 2's complement method

$$\begin{array}{l} \text{(a)} \quad \begin{array}{rcl} 66_8 & = & 0\ 0\ \quad 1\ 1\ 0\ \quad 1\ 1\ 0_2 \\ - 45_8 & = & + 1\ 1\ \quad 0\ 1\ 1\ \quad 0\ 1\ 1_2 \end{array} \quad (\text{2's complement of } -45_8) \\ \hline 21_8 & = & \textcircled{1}\ 0\ 0\ \quad 0\ 1\ 0\ \quad 0\ 0\ 1_2 \quad (\text{Ignore the carry. Answer is positive}) \end{array}$$
  

$$\begin{array}{l} \text{(b)} \quad \begin{array}{rcl} 25_8 & = & 0\ 0\ \quad 0\ 1\ 0\ \quad 1\ 0\ 1_2 \\ - 73_8 & = & + 1\ 1\ \quad 0\ 0\ 0\ \quad 1\ 0\ 1_2 \end{array} \quad (\text{2's complement of } -73_8) \\ \hline - 48_8 & = & 1\ 1\ \quad 0\ 1\ 1\ \quad 0\ 1\ 0_2 \quad (\text{Answer is negative}) \end{array}$$

2's complement of  $1\ 1\ 0\ 1\ 1\ 0\ 1\ 0_2 = -0\ 0\ 1\ 0\ 0\ 1\ 1\ 0 = -48_8$

Multiplication and division can also be performed using the binary representation of octal numbers and then making use of multiplication and division rules of binary numbers.

#### 2.5 THE HEXADECIMAL NUMBER SYSTEM

Binary numbers are long. These numbers are fine for machines but are too lengthy to be handled by human beings. So, there is a need to represent the binary numbers concisely. One number system developed with this objective is the hexadecimal number system (or Hex). Although it is

## 60 FUNDAMENTALS OF DIGITAL CIRCUITS

somewhat more difficult to interpret than the octal number system, it has become the most popular means of direct data entry and retrieval in digital systems. The hexadecimal number system is a positional-weighted system. The base or radix of this number system is 16, that means, it has 16 independent symbols. The symbols used are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. Since its base is  $16 = 2^4$ , every 4 binary digit combination can be represented by one hexadecimal digit. So, a hexadecimal number is 1/4th the length of the corresponding binary number, yet it provides the same information as the binary number. A 4-bit group is called a *nibble*. Since computer words come in 8 bits, 16 bits, 32 bits and so on, that is, multiples of 4 bits, they can be easily represented in hexadecimal. The hexadecimal system is particularly useful for human communications with computers. By far, this is the most commonly used number system in computer literature. It is used both in large and small computers.

### 2.5.1 Hexadecimal Counting Sequence

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
100	101	102	103	104	105	106	107	108	109	10A	10B	10C	10D	10E	10F
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1F0	1F1	1F2	1F3	1F4	1F5	1F6	1F7	1F8	1F9	1FA	1FB	1FC	1FD	1FE	1FF

### 2.5.2 Binary to Hexadecimal Conversion

To convert a binary number to a hexadecimal number, starting from the binary point, make groups of 4 bits each, on either side of the binary point and replace each 4-bit group by the equivalent hexadecimal digit as shown in Table 2.4.

Table 2.4 Binary to hexadecimal conversion

Hexadecimal	Binary	Hexadecimal	Binary
0	0 0 0 0	8	1 0 0 0
1	0 0 0 1	9	1 0 0 1
2	0 0 1 0	A	1 0 1 0
3	0 0 1 1	B	1 0 1 1
4	0 1 0 0	C	1 1 0 0
5	0 1 0 1	D	1 1 0 1
6	0 1 1 0	E	1 1 1 0
7	0 1 1 1	F	1 1 1 1

**EXAMPLE 2.51** Convert  $1011011011_2$  to hexadecimal.

**Solution**

Make groups of 4 bits, and replace each 4-bit group by a hex digit.

Given binary number is                    1011011011  
 Groups of four bits are                0010     1101     1011  
 Convert each group to hex            2              D              B  
 The result is                          2DB<sub>16</sub>

**EXAMPLE 2.52** Convert 0101111011.011111<sub>2</sub> to hexadecimal.

**Solution**

Given binary number is                    0101111011.011111  
 Groups of four bits are                0010     1111     1011     .     0111     1100  
 Convert each group to hex            2              F              B              .     7              C  
 The result is                          2FB.7C<sub>16</sub>

### 2.5.3 Hexadecimal to Binary Conversion

To convert a hexadecimal number to binary, replace each hex digit by its 4-bit binary group.

**EXAMPLE 2.53** Convert 4BAC<sub>16</sub> to binary.

**Solution**

Given hex number is                    4              B              A              C  
 Convert each hex digit to 4-bit binary        0100     1011     1010     1100  
 The result is                          0100101110101100<sub>2</sub>

**EXAMPLE 2.54** Convert 3A9E.B0D<sub>16</sub> to binary.

**Solution**

Given hex number is                    3              A              9              E     .     B              0              D  
 Convert each hex digit to 4-bit binary    0011     1010     1001     1110     .     1011     0000     1101  
 The result is                          0011101010011110.101100001101<sub>2</sub>

### 2.5.4 Hexadecimal to Decimal Conversion

To convert a hexadecimal number to decimal, multiply each digit in the hex number by its position weight and add all those product terms.

If the hex number is  $d_nd_{n-1}\dots d_1d_0 \cdot d_{-1}d_{-2}\dots d_{-k}$ , its decimal equivalent is

$$(d_n \times 16^n) + (d_{n-1} \times 16^{n-1}) + \dots + (d_1 \times 16^1) + (d_0 \times 16^0) + (d_{-1} \times 16^{-1}) + (d_{-2} \times 16^{-2}) + \dots$$

**EXAMPLE 2.55** Convert 5C7<sub>16</sub> to decimal.

**Solution**

Multiply each digit of 5C7 by its position weight and add the product terms.

$$\begin{aligned} 5C7_{16} &= (5 \times 16^2) + (12 \times 16^1) + (7 \times 16^0) \\ &= 1280 + 192 + 7 \\ &= 1479_{10} \end{aligned}$$

**EXAMPLE 2.56** Convert A0F9.0EB<sub>16</sub> to decimal.

**Solution**

$$\begin{aligned} A0F9.0EB_{16} &= (10 \times 16^3) + (0 \times 16^2) + (15 \times 16^1) \\ &\quad + (9 \times 16^0) + (0 \times 16^{-1}) + (14 \times 16^{-2}) + (11 \times 16^{-3}) \end{aligned}$$

$$\begin{aligned}
 &= 40960 + 0 + 240 + 9 + 0 + 0.0546 + 0.0026 \\
 &= 41209.0572_{10}
 \end{aligned}$$

### 2.5.5 Decimal to Hexadecimal Conversion

To convert a decimal integer number to hexadecimal, successively divide the given decimal number by 16 till the quotient is zero. The last remainder is the MSB. The remainders read from bottom to top give the equivalent hexadecimal integer.

To convert a decimal fraction to hexadecimal, successively multiply the given decimal fraction and subsequent decimal fractions by 16, till the product is zero or till the required accuracy is obtained, and collect all the integers to the left of the decimal point. The first integer is the MSB and the integers read from top to bottom give the hexadecimal fraction. This is known as the *hex dabble method*.

**EXAMPLE 2.57** Convert  $2598.675_{10}$  to hex.

**Solution**

The given decimal number is a mixed number. Convert the integer and the fraction parts separately to hex.

*Conversion of  $2598_{10}$*

Successive division		Remainder	
		Decimal	Hex
16	2598		
16	162	6	↑ 6
16	10	2	2
	0	10	A

Reading the remainders upwards,  $2598_{10} = A26_{16}$ .

*Conversion of  $0.675_{10}$*

Given fraction is 0.675

$0.675 \times 16$		10.8
$0.800 \times 16$		12.8
$0.800 \times 16$		12.8
$0.800 \times 16$	↓	12.8

Reading the integers to the left of hexadecimal point downwards,  $0.675_{10} = 0.ACCC_{16}$ .

Therefore,  $2598.675_{10} = A26.ACCC_{16}$ .

Conversion of very large decimal numbers to binary and very large binary numbers to decimal is very much simplified if it is done via the hex route.

**EXAMPLE 2.58** Convert  $49056_{10}$  to binary.

**Solution**

The given decimal number is very large. It is tedious to convert this number to binary directly. So, convert this to hex first, and then convert the hex to binary.

Successive division	Remainder		
	Decimal	Hex	Binary group
16   49056			
16   3066	0	↑ 0	0 0 0 0
16   191	10	A	1 0 1 0
16   11	15	F	1 1 1 1
0	11	B	1 0 1 1

Therefore,  $49056_{10} = BFA0_{16} = 1011,1111,1010,0000_2$ .

**EXAMPLE 2.59** Convert  $1011011101101110_2$  to decimal.

**Solution**

The given binary number is very large. So, perform the binary to decimal conversion via the hex route.

$$\begin{aligned}1011, 0111, 0110, 1110_2 &= B\ 7\ 6\ E_{16} \\B76E_{16} &= (11 \times 16^3) + (7 \times 16^2) + (6 \times 16^1) + (14 \times 16^0) \\&= 45056 + 1792 + 96 + 14 \\&= 46958_{10}\end{aligned}$$

### 2.5.6 Octal to Hexadecimal Conversion

To convert an octal number to hexadecimal, the simplest way is to first convert the given octal number to binary and then the binary number to hexadecimal.

**EXAMPLE 2.60** Convert  $756.603_8$  to hex.

**Solution**

Given octal number is	7	5	6	.	6	0	3
Convert each octal digit to binary	111	101	110	.	110	000	011
Groups of four bits are	0001	1110	1110	.	1100	0001	1000
Convert each four-bit group to hex	1	E	E	.	C	1	8
The result is					1EE.C18	$_{16}$	

### 2.5.7 Hexadecimal to Octal Conversion

To convert a hexadecimal number to octal, the simplest way is to first convert the given hexadecimal number to binary and then the binary number to octal.

**EXAMPLE 2.61** Convert  $B9F.AE_{16}$  to octal.

**Solution**

Given hex number is	B	9	F	.	A	E		
Convert each hex digit to binary	1011	1001	1111	.	1010	1110		
Groups of three bits are	101	110	011	111	.	101	011	100
Convert each three-bit group to octal	5	6	3	7	.	5	3	4
The result is						5637.534		

### 2.5.8 Hexadecimal Arithmetic

The rules for arithmetic operations with hexadecimal numbers are similar to the rules for decimal, octal and binary systems. Arithmetic operations are not done directly in hex. The hex numbers are first converted into binary and arithmetic operations are done in binary. Hexadecimal subtraction can be performed using 1's complement method or 2's complement method. However, hexadecimal subtraction can also be performed directly by 15's and 16's complement methods similar to the 9's and 10's complement methods of decimal system.

**EXAMPLE 2.62** (a) Add  $6E_{16}$  and  $C5_{16}$ , (b) subtract  $7B_{16}$  from  $C4_{16}$  and (c) subtract  $5D_{16}$  from  $3A_{16}$ .

**Solution**

$$\begin{array}{rcl} \text{(a)} & \begin{array}{rcl} 6E_{16} & = & 0110 \\ + C5_{16} & = & +1100 \\ \hline 133_{16} & = & 10011 \end{array} & \begin{array}{rcl} 1110_2 & & 0101_2 \\ 0101_2 & & \\ \hline 0011_2 & & \end{array} \end{array}$$

$$\begin{array}{rcl} \text{(b)} & \begin{array}{rcl} C4_{16} & = & 1100 \\ - 7B_{16} & = & +1000 \\ \hline 49_{16} & = & 0100 \end{array} & \begin{array}{l} 0100_2 \\ 0101_2 \\ \hline 1001_2 \end{array} \end{array}$$

(2's complement form of  $-7B_{16}$ )

$$\begin{array}{rcl} \text{(c)} & \begin{array}{rcl} 3A_{16} & = & 0011 \\ - 5D_{16} & = & +1010 \\ \hline -23_{16} & = & 1101 \end{array} & \begin{array}{l} 1010_2 \\ 0011_2 \\ \hline 1001_2 \end{array} \end{array}$$

(2's complement of  $-5D_{16}$ )

$$2\text{'s complement of } 11011101_2 = -00100011 = -23_{16}$$

Multiplication and division can also be performed using the binary representation of hexadecimal numbers and then making use of multiplication and division rules of binary numbers.

**EXAMPLE 2.63** Convert the following numbers with the given radix to decimal and then to binary.

$$\text{(a) } 4433_5 \quad \text{(b) } 1199_{12} \quad \text{(c) } 5654_7 \quad \text{(d) } 1221_3$$

**Solution**

To convert the given number in any number system into decimal use sum of weights method. The obtained decimal number can be converted into binary by sum of weights method or by successive division by 2 method.

$$\begin{array}{l} \text{(a) } 4433_5 = 4 \times 5^3 + 4 \times 5^2 + 3 \times 5^1 + 3 \times 5^0 \\ = 4 \times 125 + 4 \times 25 + 3 \times 5 + 3 \times 1 = 618_{10} \end{array}$$

$$618_{10} = 512 + 0 + 0 + 64 + 32 + 0 + 8 + 0 + 2 + 0 = 1001101010_2$$

$$\begin{array}{l} \text{(b) } 1199_{12} = 1 \times 12^3 + 1 \times 12^2 + 9 \times 12^1 + 9 \times 12^0 \\ = 1 \times 1728 + 1 \times 144 + 9 \times 12 + 9 \times 1 = 1989_{10} \end{array}$$

$$\begin{array}{l} 1989_{10} = 1024 + 512 + 256 + 128 + 64 + 0 + 0 + 0 + 4 + 0 + 1 \\ = 11111000101_2 \end{array}$$

- (c)  $5654_7 = 5 \times 7^3 + 6 \times 7^2 + 5 \times 7^1 + 4 \times 7^0$   
 $= 5 \times 343 + 6 \times 49 + 5 \times 7 + 4 \times 1 = 2048_{10}$   
 $2048_{10} = 2048 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0$   
 $= 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0_2$
- (d)  $1221_3 = 1 \times 3^3 + 2 \times 3^2 + 2 \times 3^1 + 1 \times 3^0$   
 $= 1 \times 27 + 2 \times 9 + 2 \times 3 + 1 \times 1 = 52_{10}$   
 $52_{10} = 32 + 16 + 0 + 4 + 0 + 0 = 1\ 1\ 0\ 1\ 0\ 0_2$

**EXAMPLE 2.64** List the first 20 numbers in base 3, base 5, base 7 and base 12 systems.

**Solution**

The first 20 numbers in different bases are given in Table 2.5.

**Table 2.5** First 20 numbers in different bases

<b>Decimal</b>	<b>Base</b>				<b>Decimal</b>	<b>Base</b>			
	<b>3</b>	<b>5</b>	<b>7</b>	<b>12</b>		<b>3</b>	<b>5</b>	<b>7</b>	<b>12</b>
0	0	0	0	0	10	101	20	13	A
1	1	1	1	1	11	102	21	14	B
2	2	2	2	2	12	110	22	15	10
3	10	3	3	3	13	111	23	16	11
4	11	4	4	4	14	112	24	20	12
5	12	10	5	5	15	120	30	21	13
6	20	11	6	6	16	121	31	22	14
7	21	12	10	7	17	122	32	23	15
8	22	13	11	8	18	200	33	24	16
9	100	14	12	9	19	201	34	25	17

**EXAMPLE 2.65** (a) A person on Saturn possessing 18 fingers has a property worth  $(1,00,000)_{18}$ . He has three daughters and two sons. He wants to distribute half the money equally to his sons and the remaining half to his daughters equally. How much will his each son and each daughter get in Indian currency.

(b) An Indian started on an expedition to Saturn with rupees 1,00,000. The expenditure on Saturn will be in the ratio of 1 : 2 : 7 for food, clothing and travelling. How much will he be spending on each item in the currency of Saturn.

**Solution**

(a)  $(1,00,000)_{18} = 1 \times 18^5 = (1,889,568)_{10}$

Half this money =  $1,889,568/2 = \text{Rs. } 9,44,784$

Each son will get  $9,44,784/2 = \text{Rs. } 4,72,392$

Each daughter will get  $9,44,784/3 = \text{Rs. } 3,14,928$

(b) Amount of money available in Indian currency Rs.  $(1,00,000)_{10}$

The amount spent on food, clothing and travelling is in the ratio of 1 : 2 : 7. So the amount spent on food = Rs.  $10,000 = 1\text{CFA}_{18}$

## 66 FUNDAMENTALS OF DIGITAL CIRCUITS

the amount spent on clothing = Rs. 20,000 =  $3702_{18}$

the amount spent on travelling = Rs. 70,000 =  $C00G_{18}$

**EXAMPLE 2.66** Given that  $16_{10} = 100_b$ , find the value of  $b$ .

**Solution**

Given  $16_{10} = 100_b$ .

Convert  $100_b$  to decimal.

$$\text{Therefore, } 16 = 1 \times b^2 + 0 \times b^1 + 0 \times b^0 = b^2$$

$$\text{or } b = 4.$$

**EXAMPLE 2.67** Given that  $292_{10} = 1204$  in some number system, find the base of that system.

**Solution**

Let the base be  $b$ . Then

$$292_{10} = 1204_b = 1 \times b^3 + 2 \times b^2 + 0 \times b^1 + 4 \times b^0$$

or

$$292_{10} = b^3 + 2b^2 + 4$$

Since 4 is the largest digit in the given number,  $b \geq 5$ . By trial and error,  $b = 6$ .

**EXAMPLE 2.68** In the following series, the same integer is expressed in different number systems. Determine the missing number of the series: 10000, 121, 100, ?, 24, 22, 20.

**Solution**

If we take the first number 10000 to be in binary, its equivalent is  $16_{10}$ . The next number is 121; it is equal to  $16_{10}$  in base 3. Similarly, 100 is equal to  $16_{10}$  in base 4, and 24 is equal to  $16_{10}$  in base 6. So, the missing number is 31 in base 5.

**EXAMPLE 2.69** Each of the following arithmetic operations is correct in at least one number system. Determine the possible bases in each operation.

$$(a) 1234 + 5432 = 6666$$

$$(b) \frac{41}{3} = 13$$

$$(c) \frac{33}{3} = 11$$

$$(d) 23 + 44 + 14 + 32 = 223$$

$$(e) \frac{302}{20} = 12.1$$

$$(f) \sqrt{41} = 5$$

**Solution**

$$(a) \begin{array}{r} 1 & 2 & 3 & 4 \\ + & 5 & 4 & 3 & 2 \\ \hline 6 & 6 & 6 & 6 \end{array}$$

It is valid in any number system with base  $\geq 7$  (since the largest digit used is 6).

(b) Let the base be  $b$ . Express both sides in decimal.

$$\frac{4b+1}{3} = b+3$$

or

$$4b+1 = 3b+9$$

or  $b = 8$

Therefore, the above equation is valid in base 8 system.

$$(c) \frac{33}{3} = 11$$

Let the base be  $b$ . Converting to decimal,

$$3b + 3 = 3(b + 1) = 3b + 3.$$

It is valid for any value of  $b$ . Since the largest digit in the number is 3. The base can be anything greater than 3.

(d) Let the base be  $b$ . Expressing in decimal,

$$2b + 3 + 4b + 4 + b + 4 + 3b + 2 = 2b^2 + 2b + 3$$

or  $b^2 - 4b - 5 = 0$

or  $b = 5$

The relation is therefore valid in base 5 system.

(e) Let the base be  $b$ . Expressing in decimal,

$$\frac{3b^2 + 2}{2b} = b + 2 + \frac{1}{b}$$

Solving for  $b$ , we get  $b = 4$ .

(f) Let the base be  $b$ . Expressing in decimal,  $\sqrt{4b + 1} = 5$ .

Squaring both sides,

$$4b + 1 = 25$$

or  $b = 6$

Therefore, the base is 6.

### SHORT QUESTIONS AND ANSWERS

**1.** What do you mean by a positional-weighted system?

**A.** A positional-weighted system is one in which the values attached to the symbols depend on their location with respect to the radix point.

**2.** Name some positional-weighted systems.

**A.** Binary, Octal, Decimal, and Hexadecimal are the commonly used positional-weighted number systems.

**3.** What does the base or radix of a number system indicate?

**A.** The base or radix of a number system indicates the number of unique symbols used in that system. The radix point separates the integer and fraction parts.

**4.** What do the extreme right and left digits in a number indicate?

**A.** In a number, the extreme left digit is the MSD and the extreme right digit is the LSD.

**5.** Why is the binary number system used in digital systems?

**A.** The Binary number system is used in digital systems because the devices used in digital systems operate in two states (ON and OFF) and the signals have two levels which are conveniently represented using binary number system.

## 68 FUNDAMENTALS OF DIGITAL CIRCUITS

6. How do you convert a decimal number into a number in any other system with base  $b$ ?
  - A. A decimal integer part can be converted into any other system by using the sum of weights method or by repeated division by  $b$ . In repeated division by  $b$ , the remainders are read from bottom to top. A decimal fraction part can be converted into any other system by using the sum of weights method or by repeated multiplication by  $b$ . In repeated multiplication by  $b$ , the integers to the left of the radix point are read from top to bottom.
7. What do you mean by a bit?
  - A. A binary digit is called a bit.
8. What do you mean by a nibble?
  - A. Each 4-bit binary group is called a nibble.
9. What do you mean by a byte?
  - A. Each 8-bit binary group is called a byte.
10. Define word.
  - A. A group of bits processed by a digital system at a time is called a word.
11. What do you mean by word length?
  - A. The number of bits used to make a word is called word length.
12. What is the easiest way to convert large decimal numbers into binary and vice versa?
  - A. The easiest way to convert large decimal numbers into binary and vice versa is via the hexadecimal system.
13. What is the easiest way to convert octal numbers to hexadecimal and vice versa?
  - A. The easiest way to convert octal numbers to hexadecimal and vice versa is via the binary system.
14. How is subtraction performed by complement method?
  - A. Subtraction in any number system with base  $b$  can be performed by using  $b$ 's complement method or  $(b - 1)$ 's complement method.
15. How are negative numbers represented?
  - A. Negative numbers can be represented in (a) sign-magnitude form or (b) 1's  $((b - 1)$ 's) complement form or (c) 2's  $(b$ 's) complement form.
16. Explain the sign magnitude representation of numbers.
  - A. In the sign magnitude representation of numbers, the MSB is used to represent sign (0 for positive and 1 for negative) and the remaining bits represent the magnitude in straight binary form.
17. What is 1's complement representation method?
  - A. In 1's complement representation, the positive numbers are represented exactly in the same form as straight binary representation whereas the negative numbers are represented by subtracting equivalent positive number from  $(2^n - 1)$ , where  $n$  is the number of bits used, or by complementing each bit of its positive equivalent.
18. What is 2's complement representation method?
  - A. In 2's complement representation, the positive numbers are represented exactly in the same way as in straight binary representation, but the negative numbers are represented by subtracting the equivalent positive number from  $2^n$  where  $n$  is the number of bits used or by adding 1 to its 1's complement form.
19. How do you obtain the 1's  $((b - 1)$ 's) complement of a number?
  - A. The 1's  $((b - 1)$ 's) complement of a number can be obtained by subtracting each bit (digit) of the number from  $1(b - 1)$ . The 1's complement of a binary number can be simply obtained by complementing each bit, i.e. by changing the 0s to 1s and 1s to 0s.

- 20.** How do you obtain the 2's ( $b$ 's) complement of a number?
- A. The 2's ( $b$ 's) complement of a number can be obtained by adding 1 to the 1's ( $((b - 1)$ 's) complement of the number.
- 21.** What do you mean by the 'sign-magnitude' form of representation?
- A. The sign-magnitude form of representation is one in which an additional bit called the sign bit is placed in front of the number. If the sign bit is a 0, the number is positive. If it is a 1, the number is negative.
- 22.** What are the characteristics of the 2's complement numbers?
- A. The 2's complement numbers have the following characteristics.
1. There is one unique zero.
  2. The 2's complement of 0 is 0.
  3. The leftmost bit cannot be used to express a quantity. It is a sign bit. If it is a 1, the number is negative and if it is a 0, the number is positive.
  4. For an  $n$ -bit word which includes the sign bit, there are  $2^{n-1} - 1$  positive integers,  $2^{n-1}$  negative integers and one 0, for a total of  $2^n$  unique states.
  5. Significant information is contained in the 1s of the positive numbers and the 0s of the negative numbers.
  6. A negative number may be converted into a positive number by finding its 2's complement.
- 23.** What are the three methods of obtaining the 2's complement of a given binary number?
- A. The 2's complement of a binary number can be obtained
- (a) By finding its 1's complement and adding 1 to it, or
  - (b) By subtracting the given  $N$ -bit binary number from  $2^N$ , or
  - (c) By copying down, starting from the LSB all bits up to and including the first 1 and then complementing the remaining bits.
- 24.** How do you perform subtraction using the 2's complement method?
- A. In 2's complement subtraction, add the 2's complement of the subtrahend to the minuend. If there is a carry out, ignore it. Look at the sign bit, i.e. the MSB of the sum term. If the MSB is a 0, the result is positive and is in true binary form. If the MSB is a 1 (whether there is a carry or no carry at all), the result is negative and is in its complement form. Take its 2's complement to get the magnitude in binary.
- 25.** How do you perform subtraction using the 1's complement method?
- A. In 1's complement subtraction, add the 1's complement of the subtrahend to the minuend. If there is a carry out, bring the carry around and add it to the LSB. This is called the end around carry. Look at the sign bit (MSB). If the MSB is a 1 (whether there is a carry or no carry at all), the result is negative and is in its 1's complement form. Take its 1's complement to get the magnitude in binary.
- 26.** What do you mean by end around carry? When does it come into picture?
- A. The process of adding the carry bit to the LSB is called end around carry. It comes into picture in 1's ( $((b - 1)$ 's) complement method of subtraction.
- 27.** How are large binary numbers represented?
- A. Large binary numbers can be represented using double precision or triple precision. Double precision requires two storage locations (registers) to represent each binary number and triple precision requires three.

## 70 FUNDAMENTALS OF DIGITAL CIRCUITS

28. What is the advantage of floating point notation?
  - A. The advantage of floating point notation is—very large and very small binary numbers can be represented very conveniently using this.
29. What is the main advantage of octal and hexadecimal systems?
  - A. The main advantage of octal and hexadecimal systems is their ease of conversion to and from binary. Also an octal number is 1/3rd of the length of the corresponding binary number and a hex number is 1/4th of the length of the corresponding binary. So, they are short hand representations of binary numbers.
30. Why is hexadecimal code widely used in digital systems?
  - A. Hexadecimal code is widely used in digital systems, because it is very convenient to enter binary data in a digital system using hex code.
31. How do you convert an octal number into binary?
  - A. To convert an octal number into binary, replace each octal digit by its 3-bit binary equivalent.
32. How do you convert a binary number into octal?
  - A. To convert a binary number into octal, starting from the binary point make groups of 3-bits on either side of the binary point and replace each 3-bit group by an octal digit. The required number of 0s can be added to the left of the integer part and to the right of the fraction part.
33. How do you convert a binary number into hex?
  - A. To convert a binary number into hex, starting from the binary point make groups of 4-bits on either side of the binary point and replace each 4-bit group by a hex digit.
34. How do you convert a hex number into binary?
  - A. To convert a hex number into binary, replace each hex digit by its 4-bit binary equivalent.

### REVIEW QUESTIONS

1. Write short notes on binary number system.
2. Discuss 1's and 2's complement methods of subtraction.
3. Discuss octal number system.
4. Discuss hexadecimal number system.
5. Discuss (a) double precision numbers and (b) floating point system.

### FILL IN THE BLANKS

1. The \_\_\_\_\_ or \_\_\_\_\_ of a number system indicates the number of unique symbols used in that system.
2. In a \_\_\_\_\_, the values attached to the symbols depend on their location with respect to the radix point.
3. The \_\_\_\_\_ separates the integer and fraction parts.
4. The extreme right digit in any number is the \_\_\_\_\_ and the extreme left digit is the \_\_\_\_\_.
5. A binary digit is called a \_\_\_\_\_.

6. Each 4-bit binary group is called a \_\_\_\_\_.
7. Binary word lengths are multiples of \_\_\_\_\_ bits.
8. Conversion of large decimal numbers into binary and vice versa is simpler via the \_\_\_\_\_.
9. The weights assigned to LSB and MSB of an 8-bit number are \_\_\_\_\_.
10. The MSB of a binary number has a weight of 512, the number consists of \_\_\_\_\_ bits.
11. The MSB of a signed binary number indicates its \_\_\_\_\_.
12. Conversion of octal to hexadecimal and vice versa is simpler via the \_\_\_\_\_.
13. Subtraction in any number system with base  $b$  can be done by the \_\_\_\_\_ method or \_\_\_\_\_ method.
14. In  $b$ 's complement method, the carry is \_\_\_\_\_ and in  $(b - 1)$ 's complement method the carry is \_\_\_\_\_.
15. The complement of the complement gives the \_\_\_\_\_.
16. The 2's complement system has a \_\_\_\_\_ zero but the 1's complement system has \_\_\_\_\_ zeros.
17. Large numbers can be represented using \_\_\_\_\_ or \_\_\_\_\_.
18. Double precision notation requires \_\_\_\_\_ storage locations (registers) to represent each binary number.
19. In the \_\_\_\_\_ notation, both very large and very small binary numbers can be represented very conveniently.
20. The main advantage of octal and hexadecimal systems is their \_\_\_\_\_.
21. An octal number is \_\_\_\_\_ the length of the corresponding binary number and a hex number is \_\_\_\_\_ the length of the corresponding binary.
22. To convert an octal number into binary, replace each octal digit by its \_\_\_\_\_.
23. To convert a hex number into binary, replace each hex digit by its \_\_\_\_\_.

### OBJECTIVE TYPE QUESTIONS

1. Knowledge of binary number system is required for the designers of computers and other digital systems because
  - (a) it is easy to learn binary number system
  - (b) it is easy to learn Boolean algebra
  - (c) it is easy to use binary codes
  - (d) the devices used in these systems operate in binary
2. A binary number with  $n$  bits all of which are 1s has the value
 

(a) $n^2 - 1$	(b) $2^n$	(c) $2^{(n-1)}$	(d) $2^n - 1$
---------------	-----------	-----------------	---------------
3. If  $\sqrt{41} = 5$ , the base (radix) of the number system is
 

(a) 5	(b) 6	(c) 7	(d) 8
-------	-------	-------	-------
4. The number of bits required to assign binary roll numbers to a class of 60 students is
 

(a) 5	(b) 6	(c) 7	(d) 8
-------	-------	-------	-------
5. The radix of the number system if  $302/20 = 12.1$  is
 

(a) 3	(b) 4	(c) 5	(d) 6
-------	-------	-------	-------

72 FUNDAMENTALS OF DIGITAL CIRCUITS

## PROBLEMS

- 2.1** Subtract the following decimal numbers by the 9's and 10's complement methods.  
(a)  $274 - 86$       (b)  $93 - 615$       (c)  $574.6 - 279.7$       (d)  $376.3 - 765.6$

**2.2** Convert the following binary numbers to decimal.  
(a) 1011      (b) 1101101      (c) 1101.11      (d) 1101110.011

**2.3** Convert the following decimal numbers to binary.  
(a) 37      (b) 28      (c) 197.56      (d) 205.05

**2.4** Add the following binary numbers.  
(a)  $11011 + 1101$       (b)  $1011 + 1101 + 1001 + 1111$   
(c)  $10111.101 + 110111.01$       (d)  $1010.11 + 1101.10 + 1001.11 + 1111.11$

**2.5** Subtract the following binary numbers.  
(a)  $1011 - 101$       (b)  $10110 - 1011$       (c)  $1100.10 - 111.01$       (d)  $10001.01 - 1111.11$

**2.6** Multiply the following binary numbers.  
(a)  $1101 \times 101$       (b)  $11001 \times 10$       (c)  $1101.11 \times 101.1$       (d)  $10110 \times 10.1$

## 74 FUNDAMENTALS OF DIGITAL CIRCUITS

**2.7** Divide the following binary numbers.

- (a) 1010 by 11      (b) 11110 by 101      (c) 11011 by 10.1      (d) 110111.1 by 101

**2.8** Multiply the following binary numbers by the computer method.

- (a) 1011 × 11      (b) 1001 × 101      (c) 1100 × 110      (d) 1111 × 100

**2.9** Divide the following binary numbers by the computer method.

- (a) 1011 by 10      (b) 110010 by 101      (c) 1100100 by 1101      (d) 100010 by 1010

**2.10** Find the 12-bit 2's complement form of the following decimal numbers.

- (a) -37      (b) -173      (c) -65.5      (d) -197.5

**2.11** Find the 12-bit 1's complement form of the following decimal numbers.

- (a) -97      (b) -224      (c) -205.75      (d) -29.375

**2.12** Subtract the following decimal numbers using the 12-bit 2's complement arithmetic.

- (a) 46 - 19      (b) 27 - 75      (c) 125.25 - 46.75      (d) 36.75 - 89.5

**2.13** Subtract the following decimal numbers using the 12-bit 1's complement arithmetic.

- (a) 52 - 17      (b) 46 - 84      (c) 63.75 - 17.5      (d) 73.5 - 112.75

**2.14** Represent the following decimal numbers in 8-bit: (i) sign magnitude form, (ii) sign 1's complement form, and (iii) sign 2's complement form.

- (a) +14      (b) +27      (c) +45      (d) -17  
(e) -37      (f) -76

**2.15** Convert the following octal numbers to hexadecimal.

- (a) 256      (b) 2035      (c) 1762.46      (d) 6054.263

**2.16** Convert the following hexadecimal numbers to octal.

- (a) 2AB      (b) 42FD      (c) 4F7.A8      (d) BC70.0E

**2.17** Convert the following octal numbers to decimal.

- (a) 463      (b) 2056      (c) 2057.64      (d) 6534.04

**2.18** Convert the following decimal numbers to octal.

- (a) 287      (b) 3956      (c) 420.6      (d) 8476.47

**2.19** Add the following octal numbers.

- (a) 173 + 265      (b) 1247 + 2053      (c) 25.76 + 16.57      (d) 273.56 + 425.07

**2.20** Subtract the following octal numbers.

- (a) 64 - 37      (b) 462 - 175      (c) 175.6 - 47.7      (d) 3006.05 - 2657.16

**2.21** Multiply the following octal numbers.

- (a) 46 × 4      (b) 267 × 5      (c) 26.5 × 2.5      (d) 647.2 × 5.4

**2.22** Divide the following octal numbers.

- (a) 420 by 5      (b) 2567 by 6      (c) 153.6 by 7      (d) 4634.62 by 12

**2.23** Subtract the following octal numbers by the 7's and 8's complement methods.

- (a) 76 - 25      (b) 256 - 643      (c) 173.5 - 66.6      (d) 243.6 - 705.64

**2.24** Convert the following hexadecimal numbers to binary.

- (a) C20      (b) F297      (c) AF9.B0D      (d) E79A.6A4

**2.25** Convert the following binary numbers to hexadecimal.

- (a) 10110      (b) 1011011011  
(c) 110110111.01111      (d) 1101101101101.101101

**2.26** Convert the following hexadecimal numbers to decimal.

- (a) AB6      (b) 2EB7      (c) A08F.EA      (d) 8E47.AB



## VHDL PROGRAMS

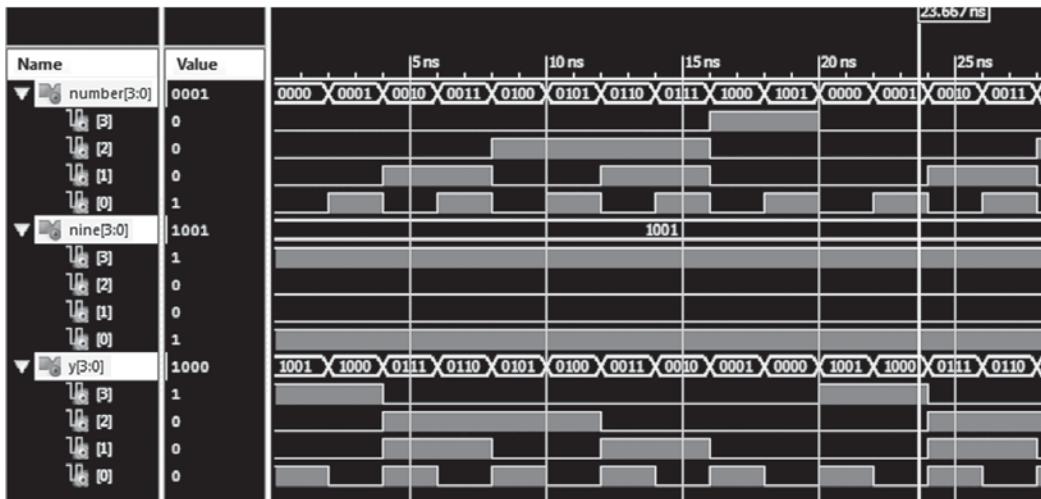
### 1. VHDL PROGRAM FOR NINE'S COMPLEMENT USING DATA FLOW MODELING

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity NINES_COMPLEMENT is
    Port ( Number, Nine : in STD_LOGIC_VECTOR (3 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end NINES_COMPLEMENT;
architecture Behavioral of NINES_COMPLEMENT is
begin
Y <= (Nine - Number);
end Behavioral;

```

### SIMULATION OUTPUT:



### SIMULATION OUTPUT:

Name	Value	999,996 ps	999,997 ps	999,998 ps	999,999 ps	1,
number[15:0]	3465		3465			
nine[15:0]	9999		9999			
y[15:0]	6534		6534			

### 2. VHDL PROGRAM FOR TEN'S COMPLEMENT USING DATA FLOW MODELING

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

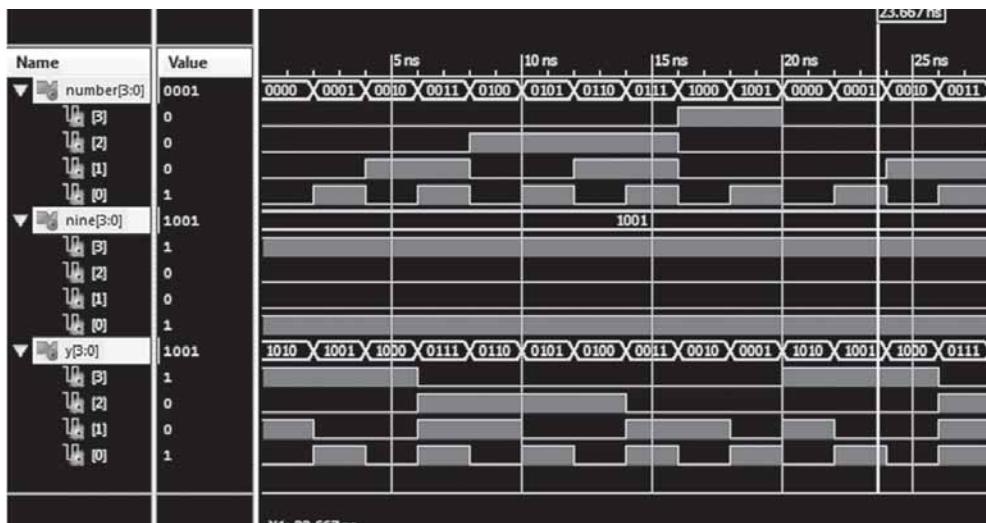
```

```

use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity TENS_COMPLEMENT is
Port ( Number ,Nine : in STD_LOGIC_VECTOR (3 downto 0);
Y : out STD_LOGIC_VECTOR (3 downto 0));
end TENS_COMPLEMENT;
architecture Behavioral of TENS_COMPLEMENT is
SIGNAL M : STD_LOGIC_VECTOR (3 downto 0);
begin
M <= (Nine - Number);
Y <= (M+1);
end Behavioral;

```

### SIMULATION OUTPUT:



### 3. VHDL PROGRAM FOR NINE'S COMPLEMENT SUBTRACTION USING DATA FLOW MODELING

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity NINES_COMPLEMENT_SUBTRACTION is
Port ( Number1,Number2 ,Nine : in STD_LOGIC_VECTOR (15 downto 0);
       Y : out STD_LOGIC_VECTOR (15 downto 0));
end NINES_COMPLEMENT_SUBTRACTION;
architecture Behavioral of NINES_COMPLEMENT_SUBTRACTION is
SIGNAL M : STD_LOGIC_VECTOR (15 downto 0);
begin
M<= (Nine - Number2);

```

## 78 FUNDAMENTALS OF DIGITAL CIRCUITS

```
Y <= (M+Number1);
end Behavioral;
```

### SIMULATION OUTPUT:

Name	Value	999,996 ps	999,997 ps	999,998 ps	999,999 ps
► number1[15:0]	5324		5324		
► number2[15:0]	6875		6875		
► nine[15:0]	9999		9999		
► y[15:0]	8448		8448		

## 4. VHDL PROGRAM FOR TEN'S COMPLEMENT SUBTRACTION USING DATA FLOW MODELING

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity TENS_COMPLEMENT_SUBTRACTION is
    Port ( Number1,Number2 ,Nine : in STD_LOGIC_VECTOR (15 downto 0);
           Y : out STD_LOGIC_VECTOR (15 downto 0));
end TENS_COMPLEMENT_SUBTRACTION;
architecture Behavioral of TENS_COMPLEMENT_SUBTRACTION is
SIGNAL M1 : STD_LOGIC_VECTOR (15 downto 0);
SIGNAL M2 : STD_LOGIC_VECTOR (15 downto 0);
begin
M1 <= (Nine - Number2);
M2 <= (M1 + 1);
Y <= (Number1 + M2);
end Behavioral;
```

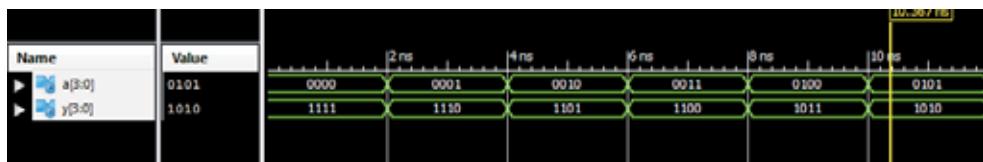
### SIMULATION OUTPUT:

Name	Value	999,996 ps	999,997 ps	999,998 ps	999,999 ps
► number1[15:0]	5324		5324		
► number2[15:0]	6875		6875		
► nine[15:0]	9999		9999		
► y[15:0]	8449		8449		

## 5. VHDL PROGRAM FOR ONE'S COMPLEMENT USING DATA FLOW MODELING

```
library ieee;
use ieee.std_logic_1164.all;
entity ones_complement is
port ( a : in std_logic_vector (3 downto 0);
       y : out std_logic_vector (3 downto 0));
end ones_complement;
architecture behavioral of ones_complement is
begin
y <= (not (a));
end behavioral;
```

### SIMULATION OUTPUT:



## 6. VHDL PROGRAM FOR TWO'S COMPLEMENT USING DATA FLOW MODELING

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity ones_complement is
port ( a : in std_logic_vector (3 downto 0);
y : out std_logic_vector (3 downto 0));
end ones_complement;
architecture behavioral of ones_complement is
begin
y <= ((not (a)) + "1");
end behavioral;
```

### SIMULATION OUTPUT:



## 7. VHDL PROGRAM FOR BINARY ADDITION USING DATA FLOW MODELING

```
library ieee;
use ieee.std_logic_1164.all;
```

## 80 FUNDAMENTALS OF DIGITAL CIRCUITS

```
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity binary_addition is
port ( a : in std_logic_vector (3 downto 0);
       b : in std_logic_vector (3 downto 0);
       y : out std_logic_vector (3 downto 0));
end binary_addition;
architecture behavioral of binary_addition is
begin
y <= (a + b) ;
end behavioral;
```

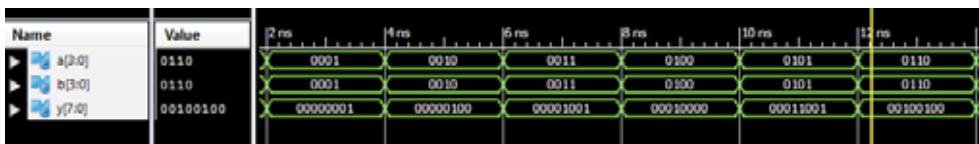
### SIMULATION OUTPUT:



## 8. VHDL PROGRAM FOR BINARY MULTIPLICATION USING DATA FLOW MODELING

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity binary_addition is
port ( a : in std_logic_vector (3 downto 0);
       b : in std_logic_vector (3 downto 0);
       y : out std_logic_vector (7 downto 0));
end binary_addition;
architecture behavioral of binary_addition is
begin
y <= (a * b) ;
end behavioral;
```

### SIMULATION OUTPUT:



## VERILOG PROGRAMS

### 1. VERILOG PROGRAM FOR NINE'S COMPLEMENT USING DATA FLOW MODELING

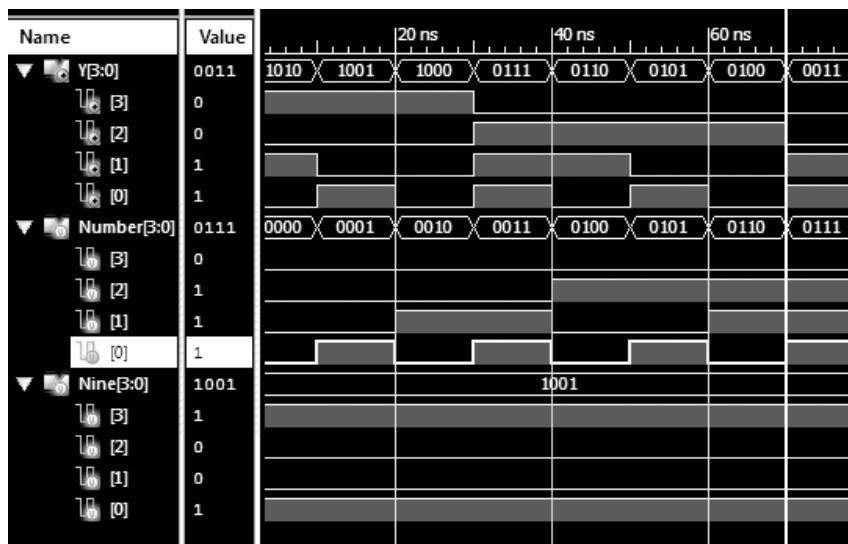
```
module NINES_COMPLEMENT (Number ,Nine , Y );
input [3:0] Number ,Nine ;
output [3:0] Y;
assign Y = Nine - Number;
endmodule
```

#### SIMULATION OUTPUT:

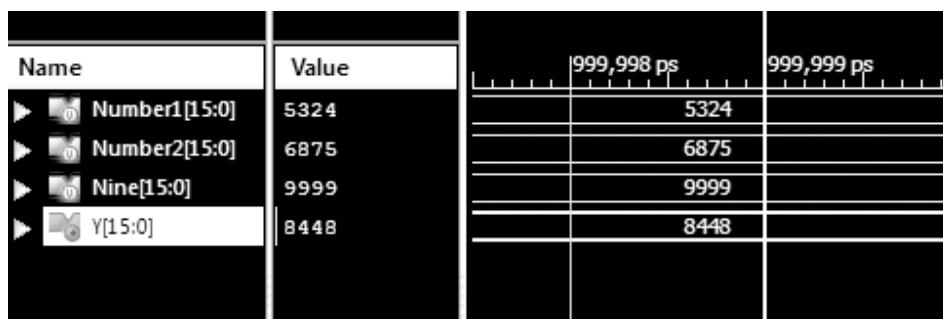


### 2. VERILOG PROGRAM FOR TEN'S COMPLEMENT USING DATA FLOW MODELING

```
module TENS_COMPLEMENT (Number ,Nine , Y );
input [3:0] Number ,Nine ;
output [3:0] Y;
wire [3:0] M;
assign M = Nine - Number;
assign Y = (M+1);
endmodule
```

**SIMULATION OUTPUT:****3. VERILOG PROGRAM FOR NINE'S COMPLEMENT SUBTRACTION USING DATA FLOW MODELING**

```
module NINES_COMPLEMENT_SUBTRACTION(Number1,Number2, Nine,Y);
input [15:0] Number1,Number2,Nine;
output [15:0] Y;
wire [15:0] M;
assign M = Nine - Number2;
assign Y = (M+Number1);
endmodule
```

**SIMULATION OUTPUT:****4. VERILOG PROGRAM FOR TEN'S COMPLEMENT SUBTRACTION USING DATA FLOW MODELING**

```
module TENS_COMPLEMENT_SUBTRACTION (Number1,Number2 ,Nine , Y );
input [15:0] Number1,Number2 ,Nine ;

```

```

output [15:0] Y;
wire [15:0] M1,M2;
assign M1 = Nine - Number2;
assign M2 = M1+1;
assign Y = (Number1+M2);
endmodule

```

#### SIMULATION OUTPUT:

Name	Value	999,997 ps	999,998 ps	999,999 ps
► Number1[15:0]	5324		5324	
► Number2[15:0]	6875		6875	
► Nine[15:0]	9999		9999	
► Y[15:0]	8449		8449	

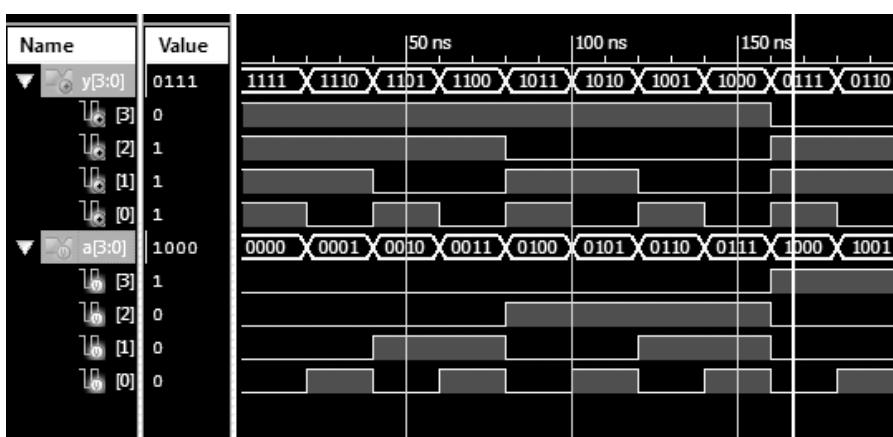
#### 5. VERILOG PROGRAM FOR ONE'S COMPLEMENT USING DATA FLOW MODELING

```

module ones_complement(a,y);
input [3:0] a;
output [3:0] y;
assign y = ~a;
endmodule

```

#### SIMULATION OUTPUT:



#### 6. VERILOG PROGRAM FOR TWO'S COMPLEMENT USING DATA FLOW MODELING

```

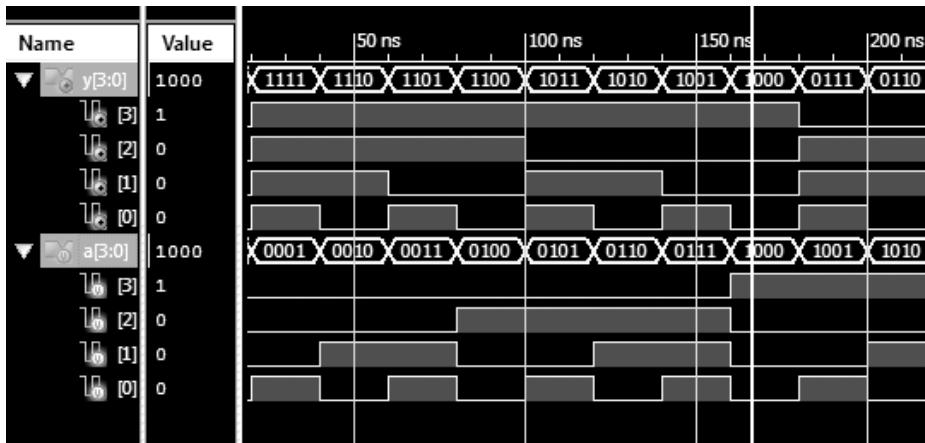
module twos_complement (a,y);
input [3:0] a;

```

## 84 FUNDAMENTALS OF DIGITAL CIRCUITS

```
output [3:0] y;  
assign y = ((~a)+1);  
endmodule
```

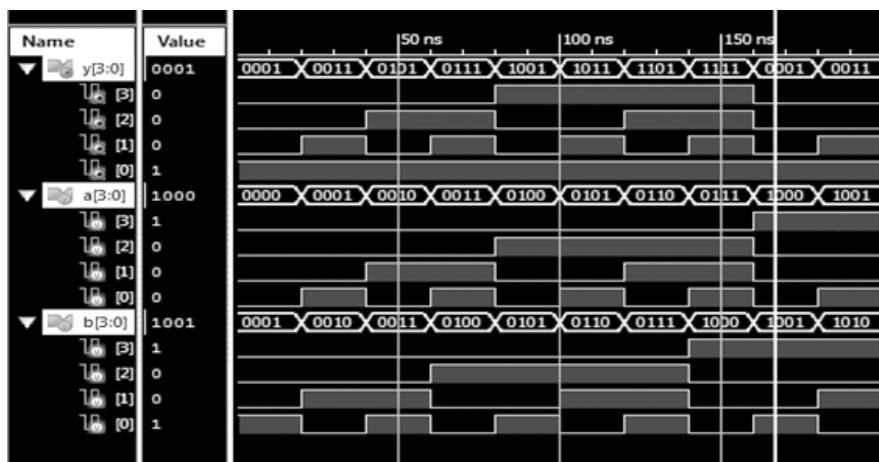
### SIMULATION OUTPUT:



## 7. VERILOG PROGRAM FOR BINARY ADDITION USING DATA FLOW MODELING

```
module binary_addition (a,b,y);  
input [3:0] a,b;  
output [3:0] y;  
assign y = a+b;  
endmodule
```

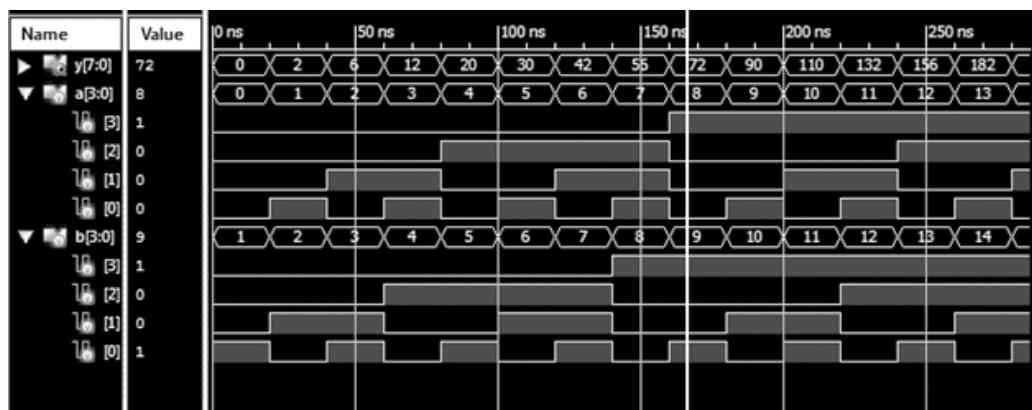
### SIMULATION OUTPUT:



## 8. VERILOG PROGRAM FOR BINARY MULTIPLICATION USING DATA FLOW MODELING

```
module binary_multiplication (a,b,y);
input [3:0] a,b;
output [7:0] y;
assign y = a*b;
endmodule
```

### SIMULATION OUTPUT:



# 3

## BINARY CODES

### 3.1 CLASSIFICATION OF BINARY CODES

#### 3.1.1 Numeric and Alphanumeric Codes

Binary codes can be classified as *numeric codes* and *alphanumeric codes*. Numeric codes are codes which represent numeric information, i.e. only numbers as a series of 0s and 1s. 8421, XS-3, Gray code are numeric codes. Numeric codes used to represent the decimal digits are called Binary Coded Decimal (BCD) codes. 8421, 2421, 5211 are BCD codes. Alphanumeric codes are codes which represent alphanumeric information, i.e. letters of the alphabet and decimal numbers as a sequence of 0s and 1s. EBCDIC code and ASCII code are alphanumeric codes.

We are very comfortable with the decimal number system, but digital systems force us to use the binary system. Although the binary number system has many practical advantages and is widely used in digital computers, in many cases it is very convenient to work with decimal numbers, especially when communication between man and machine is extensive. Since most of the numerical data generated by man are in decimal numbers, to simplify the communication process between man and machine, several systems of numeric codes have been devised to represent decimal numbers as a series of BCD codes.

A BCD code is one, in which the digits of a decimal number are encoded—one at a time—into groups of four binary digits. These codes combine the features of decimal and binary numbers. There are a large number of BCD codes. In order to represent decimal digits 0, 1, 2,...,9, it is necessary to use a sequence of at least four binary digits. Such a sequence of binary digits which represents a decimal digit is called a *code word*. All BCD codes have at least six unassigned code words.

#### 3.1.2 Weighted and Non-weighted Codes

The BCD codes may be *weighted codes* or *non-weighted codes*. The weighted codes are those which obey the position-weighting principle. Each position of the number represents a specific

weight. For each group of four bits, the sum of the weights of those positions where the binary digit is 1 is equal to the decimal digit which the group represents. 8421, 2421, 84-2-1 are weighted codes. Non-weighted codes are codes which are not assigned with any weight to each digit position, i.e. each digit position within the number is not assigned fixed value. Excess-3(XS-3) code and Gray code are non-weighted codes. There are several weighted codes.

### 3.1.3 Positively-weighted and Negatively-weighted Codes

The weighted codes may be either *positively-weighted* codes or *negatively-weighted* codes. Positively-weighted codes are those in which all the weights assigned to the binary digits are positive. There are only 17 positively-weighted codes. In every positively-weighted code, the first weight must be 1, the second weight must be either 1 or 2, and the sum of all the weights must be equal to or greater than 9. The codes 8421, 2421, 5211, 3321 and 4311 are some of the positively-weighted codes available. Negatively-weighted codes are those in which some of the weights assigned to the binary digits are negative. The codes 642-3, 631-1, 84-2-1 and 74-2-1 are some of the negatively-weighted codes.

Table 3.1 shows some of the positively-weighted, negatively-weighted, and non-weighted codes.

**Table 3.1** Binary coded decimal codes

Decimal digit	8 4 2 1	2 4 2 1	5 2 1 1	5 4 2 1	6 4 2 -3	8 4 -2 -1	XS-3
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1	0 1 0 1	0 1 1 1	0 1 0 0
2	0 0 1 0	0 0 1 0	0 0 1 1	0 0 1 0	0 0 1 0	0 1 1 0	0 1 0 1
3	0 0 1 1	0 0 1 1	0 1 0 1	0 0 1 1	1 0 0 1	0 1 0 1	0 1 1 0
4	0 1 0 0	0 1 0 0	0 1 1 1	0 1 0 0	0 1 0 0	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 1 1	1 0 0 0	1 0 0 0	1 0 1 1	1 0 1 1	1 0 0 0
6	0 1 1 0	1 1 0 0	1 0 1 0	1 0 0 1	0 1 1 0	1 0 1 0	1 0 0 1
7	0 1 1 1	1 1 0 1	1 1 0 0	1 0 1 0	1 1 0 1	1 0 0 1	1 0 1 0
8	1 0 0 0	1 1 1 0	1 1 1 0	1 0 1 1	1 0 1 0	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 1 1	1 1 1 1	1 1 0 0	1 1 1 1	1 1 1 1	1 1 0 0
<hr/>							
<b>Unused bit combinations</b>	1 0 1 0	0 1 0 1	0 0 1 0	0 1 0 1	0 0 0 1	0 0 0 1	0 0 0 0
	1 0 1 1	0 1 1 0	0 1 0 0	0 1 1 0	0 0 1 1	0 0 1 0	0 0 0 1
	1 1 0 0	0 1 1 1	0 1 1 0	0 1 1 1	0 1 1 1	0 0 1 1	0 0 1 0
	1 1 0 1	1 0 0 0	1 0 0 1	1 1 0 1	1 0 0 0	1 1 0 0	1 1 0 1
	1 1 1 0	1 0 0 1	1 0 1 1	1 1 1 0	1 1 0 0	1 1 0 1	1 1 1 0
	1 1 1 1	1 0 1 0	1 1 0 1	1 1 1 1	1 1 1 0	1 1 1 0	1 1 1 1

### 3.1.4 Error Detecting and Error Correcting Codes

Binary codes may also be error detecting codes or error correcting codes. Codes which allow only error detection are called error detecting codes. Shift counter code, 2-out-of-5, 63210 codes are error detecting codes. Codes which allow error detection as well as error correction are called error correcting codes. The Hamming code is a error correcting code.

### 3.1.5 Sequential Codes

A sequential code is one, in which each succeeding code word is one binary number greater than its preceding code word. Such a code facilitates mathematical manipulation of data. The 8421 and XS-3 codes are sequential. The codes 5211, 2421 and 642-3 are not sequential.

### 3.1.6 Self-complementing Codes

A code is said to be self-complementing, if the code word of the 9's complement of  $N$ , i.e. of  $9 - N$  can be obtained from the code word of  $N$  by interchanging all the 0s and 1s. Therefore, in a self-complementing code, the code for 9 is the complement of the code for 0, the code for 8 is the complement of the code for 1 and so on. The 2421, 5211, 642-3, 84-2-1 and XS-3 are self-complementing codes. The 8421 and 5421 codes are not self-complementing. The self-complementing property is desirable in a code when the 9's complement must be found such as in 9's complement subtraction. Self complementing codes have an advantage that their logical complement is the same as the arithmetic complement. For a code to be self-complementing, the sum of all its weights must be 9. This is because whatever may be the weights, 0 is to be represented by 0000 and since in a self-complementing code, the code for 9 is the complement of the code for 0, 9 has to be represented by 1111. There are only four (2421, 5211, 3321, 4311) positively-weighted self-complementing codes. There are 13 negatively-weighted self-complementing codes.

### 3.1.7 Cyclic Codes

Cyclic codes are those in which each successive code word differs from the preceding one in only one bit position. They are also called *unit distance codes*. The unit distance codes have special advantages in that they minimize transitional errors or flashing. The Gray code is a cyclic code. It is often used for translating an analog quantity such as shaft position into a digital form.

### 3.1.8 Reflective Codes

A reflective code is a binary code in which the  $n$  least significant bits for code words  $2^n$  through  $2^{n+1} - 1$  are the mirror images of those for 0 through  $2^n - 1$ . The Gray code is a reflective code.

### 3.1.9 Straight Binary Code

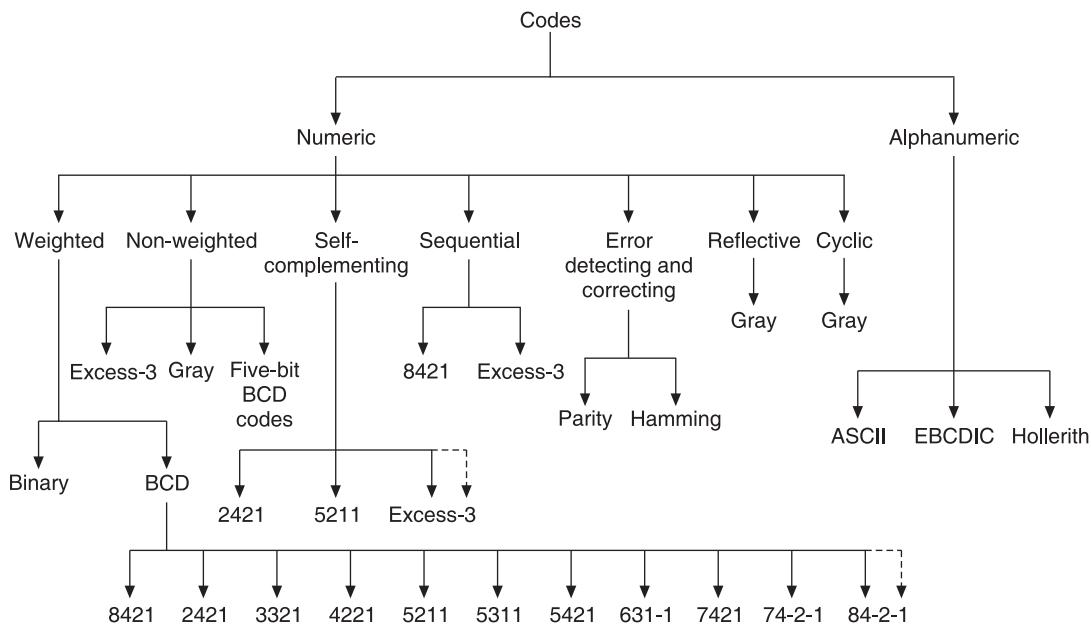
A straight binary code is used to represent numbers using natural binary form as discussed earlier. Various arithmetic operations can be performed in this form. It is a weighted code since a weight is attached to every position.

The classification of codes is shown in Figure 3.1.

## 3.2 THE 8421 BCD CODE (NATURAL BCD CODE)

The 8421 BCD code is so widely used that it is a common practice to refer to it simply as BCD code. In this code, each decimal digit, 0 through 9, is coded by a 4-bit binary number. It is also called the *natural binary code* because of the 8, 4, 2 and 1 weights attached to it. It is a weighted code and is also sequential. Therefore, it is useful for mathematical operations. The main advantage of this code is its ease of conversion to and from decimal. It is less efficient than the

pure binary, in the sense that it requires more bits. For example, the decimal number 14 can be represented as 1110 in pure binary but as 0001 0100 in 8421 code. Another disadvantage of the BCD code is that, arithmetic operations are more complex than they are in pure binary. There are six illegal combinations 1010, 1011, 1100, 1101, 1110 and 1111 in this code, i.e. they are not part of the 8421 BCD code system. A disadvantage of the 8421 code is that, the rules of binary addition and subtraction do not apply to the entire 8421 number but only to the individual 4-bit groups.



**Figure 3.1** Classification of codes.

### 3.2.1 BCD Addition

The BCD addition is performed by individually adding the corresponding digits of the decimal numbers expressed in 4-bit binary groups starting from the LSD. If there is no carry and the sum term is not an illegal code, no correction is needed. If there is a carry out of one group to the next group, or if the sum term is an illegal code, then  $6_{10}$  (0110) is added to the sum term of that group and the resulting carry is added to the next group. (This is done to skip the six illegal states).

**EXAMPLE 3.1** Perform the following decimal additions in the 8421 code.

### *Solution*

$$(a) \quad \begin{array}{r} 25 \\ +13 \\ \hline 38 \end{array} \quad \Rightarrow \quad \begin{array}{r} 0010 & 0101 \\ +0001 & 0011 \\ \hline 0011 & 1000 \end{array} \quad \begin{array}{l} (25 \text{ in BCD}) \\ (13 \text{ in BCD}) \\ \text{(No carry, no illegal code. So, this is the correct sum.)} \end{array}$$

## 90 FUNDAMENTALS OF DIGITAL CIRCUITS

$$\begin{array}{r}
 \text{(b)} \quad \begin{array}{r} 679.6 \\ + 536.8 \end{array} \Rightarrow \begin{array}{rrrrr} 0110 & 0111 & 1001 & .0110 & (679.6 \text{ in BCD}) \\ +0101 & 0011 & 0110 & .1000 & (536.8 \text{ in BCD}) \end{array} \\
 \hline
 \begin{array}{r} 1216.4 \\ 1011 & 1010 & 1111 & .1110 \\ +0110 & +0110 & +0110 & +.0110 \end{array} \quad \begin{array}{l} (\text{All are illegal codes}) \\ (\text{Add } 0110 \text{ to each}) \end{array} \\
 \hline
 \begin{array}{rrrrr} \textcircled{1}0001 & \textcircled{1}0000 & \textcircled{1}0101 & \textcircled{1}.0100 & (\text{Propagate carry}) \\ +1 \swarrow & +1 \swarrow & +1 \swarrow & +1 \swarrow & \end{array} \\
 \hline
 \begin{array}{rrrrr} 0001 & 0010 & 0001 & 0110 & .0100 \\ 1 & 2 & 1 & 6 & .4 \end{array} \quad (\text{Corrected sum} = 1216.4)
 \end{array}$$

### 3.2.2 BCD Subtraction

The BCD subtraction is performed by subtracting the digits of each 4-bit group of the subtrahend from the corresponding 4-bit group of the minuend in binary starting from the LSD. If there is no borrow from the next higher group then no correction is required. If there is a borrow from the next group, then  $6_{10}$  (0110) is subtracted from the difference term of this group. (This is done to skip the six illegal states.)

**EXAMPLE 3.2** Perform the following decimal subtractions in the 8421 BCD code.

$$\begin{array}{ll}
 \text{(a)} \quad 38 - 15 & \text{(b)} \quad 206.7 - 147.8
 \end{array}$$

**Solution**

$$\begin{array}{r}
 \text{(a)} \quad \begin{array}{r} 38 \\ -15 \end{array} \Rightarrow \begin{array}{rr} 0011 & 1000 \\ -0001 & 0101 \end{array} \quad \begin{array}{l} (38 \text{ in BCD}) \\ (15 \text{ in BCD}) \end{array} \\
 \hline
 \begin{array}{r} 23 \\ 0010 & 0011 \end{array} \quad (\text{No borrow. So, this is the correct difference.})
 \end{array}$$
  

$$\begin{array}{r}
 \text{(b)} \quad \begin{array}{r} 206.7 \\ -147.8 \end{array} \Rightarrow \begin{array}{rrrrr} 0010 & 0000 & 0110 & .0111 & (206.7 \text{ in BCD}) \\ -0001 & 0100 & 0111 & .1000 & (147.8 \text{ in BCD}) \end{array} \\
 \hline
 \begin{array}{rrrrr} 58.9 & 0000 & 1011 & 1110 & .1111 \\ & -0110 & -0110 & -0110 & -.0110 \end{array} \quad (\text{Borrows are present, subtract } 0110) \\
 \hline
 \begin{array}{rrrrr} & 0101 & 1000 & .1001 & ( \text{Corrected difference} = 58.9_{10} ) \end{array}
 \end{array}$$

### 3.2.3 BCD Subtraction Using 9's and 10's Complement Methods

In practice, subtraction is performed by the complement method. Since we are subtracting decimal digits, we must form the 9's or 10's complement of the decimal subtrahend and encode that number in the 8421 code. The resulting BCD numbers are then added.

**EXAMPLE 3.3** Perform the following decimal subtractions in BCD by the 9's complement method.

$$\begin{array}{ll}
 \text{(a)} \quad 305.5 - 168.8 & \text{(b)} \quad 679.6 - 885.9
 \end{array}$$

**Solution**

$$\begin{array}{r}
 \text{(a)} \quad \begin{array}{r} 305.5 \\ -168.8 \end{array} \Rightarrow \begin{array}{r} 305.5 \\ +831.1 \end{array} \quad (\text{9's complement of } 168.8) \\
 \hline
 \begin{array}{r} 136.7 \\ \textcircled{1}136.6 \\ \swarrow + 1 \end{array} \quad (\text{End around carry}) \\
 \hline
 \begin{array}{r} 136.7 \quad (\text{Corrected difference}) \end{array}
 \end{array}$$

$305.5_{10}$	$+831.1_{10}$	$\Rightarrow$	$0011\ 0000\ 0101\ .0101$	(305.5 in BCD)
			$+1000\ 0011\ 0001\ .0001$	(9's complement of 168.8 in BCD)
			$\hline$	
			$+1011\ 0011\ 0110\ .0110$	(1011 is an illegal code, add 0110)
			$+0110$	
			$\hline$	
			<b>①</b> 0001 0011 0110 .0110	
			$\Downarrow$	
			$+1$	(End around carry)
			$\hline$	
			0001 0011 0110 .0111	(Corrected difference = 136.7)

$$(b) \quad \begin{array}{r} 679.6 \\ -885.9 \\ \hline -206.3 \end{array} \Rightarrow \begin{array}{r} 679.6 \\ +114.0 \\ \hline 793.6 \end{array} \quad \begin{array}{l} (9\text{'s complement of } 885.9) \\ (\text{No carry}) \end{array}$$

There is no carry indicating that the result is negative and is in its 9's complement form. The 9's complement of 793.6 is 206.3. So, the result is -206.3.

$679.6_{10}$	$+114.0_{10}$	$\Rightarrow$	0110 +0001 <hr/> 0111 +0110 <hr/> 0111 +0110 <hr/> 0111 +1 ↗ <hr/> 0111	.0111 0001 0100 .0000 <hr/> 1101 .0110 <hr/> 0011 .0110 <hr/> 0011 .0110	(679.6 in BCD) (9's complement of 885.9 in BCD) (1101 is an illegal code, add 0110)  (Propagate carry)  (No carry; 793.6 in BCD)
--------------	---------------	---------------	---	---	--

There is no carry and, therefore, the result is negative and is in its 9's complement form. The 9's complement of 793.6 is 206.3. So the result is -206.3.

**EXAMPLE 3.4** Perform the following subtractions in 8421 code using the 10's complement method.

(a) 342.7 – 108.9

(b) 206.4 – 507.6

### *Solution*

$$(a) \quad \begin{array}{r} 342.7 \\ -108.9 \\ \hline 233.8 \end{array} \Rightarrow \begin{array}{r} 342.7 \\ +891.1 \\ \hline 1233.8 \end{array} \quad \begin{array}{l} \text{(10's complement of 108.9)} \\ \text{(Ignore carry)} \end{array}$$

$  \begin{array}{r}  342.7 \\  +891.1  \end{array}  $	$  \begin{array}{r}  0011 & 0100 & 0010 & .0111 \\  +1000 & 1001 & 0001 & .0001 \\  \hline  1011 & 1101 & 0011 & .1000  \end{array}  $	<p>(342.7 in BCD)          (10's complement of 108.9 in BCD)</p>
	$  \begin{array}{r}  +0110 & +0110 \\  \hline  \textcircled{1}0001 & \textcircled{1}0011 & 0011 & .1000  \end{array}  $	<p>(1011 and 1101 are illegal codes,          add 0110 to each.)</p>
	$  \begin{array}{r}  +1 \leftarrow & +1 \leftarrow \\  \textcircled{1} &  \end{array}  $	<p>(Propagate carry)</p>
	$  \begin{array}{r}  0010 & 0011 & 0011 & .1000 \\  0010 & 0011 & 0011 & .1000  \end{array}  $	<p>(Ignore carry)</p>

$$(b) \begin{array}{r} 206.4 \\ -507.6 \\ \hline -301.2 \end{array} \Rightarrow \begin{array}{r} 206.4 \\ +492.4 \\ \hline 698.8 \end{array} \begin{array}{l} (\text{10's complement of } 507.6) \\ (\text{No carry}) \end{array}$$

No carry. So, the answer is negative and is in 10's complement form. The 10's complement of 698.8 is  $-301.2$ .

$$\begin{array}{r} 206.4 \\ +492.4 \\ \hline \end{array} \Rightarrow \begin{array}{r} 0010\ 0000\ 0110\ .0100 \\ +0100\ 1001\ 0010\ .0100 \\ \hline 0110\ 1001\ 1000\ .1000 \end{array} \begin{array}{l} (\text{206.4 in BCD}) \\ (\text{10's complement of } 507.6 \text{ in BCD}) \\ (\text{No illegal codes, no carry}) \end{array}$$

As there is no carry, the result is negative and is in its 10's complement form. The 10's complement of 698.8 is 301.2. So, the corrected difference is  $-301.2$ .

### 3.3 THE EXCESS THREE (XS-3) CODE

The Excess-3 code, also called XS-3, is a non-weighted BCD code. This code derives its name from the fact that each binary code word is the corresponding 8421 code word plus 0011(3). It is a sequential code and, therefore, can be used for arithmetic operations. It is a self-complementing code. Therefore, subtraction by the method of complement addition is more direct in XS-3 code than that in 8421 code. The XS-3 code has six invalid states 0000, 0001, 0010, 1101, 1110 and 1111. The XS-3 code has some very interesting properties when used in addition and subtraction.

#### 3.3.1 XS-3 Addition

To add in XS-3, add the XS-3 numbers by adding the 4-bit groups in each column starting from the LSD. If there is no carry out from the addition of any of the 4-bit groups, subtract 0011 from the sum term of those groups (because when two decimal digits are added in XS-3 and there is no carry, the result is in XS-6). If there is a carry out, add 0011 to the sum term of those groups (because when there is a carry, the invalid states are skipped and the result is in normal binary).

**EXAMPLE 3.5** Perform the following additions in XS-3 code.

$$(a) 37 + 28 \qquad \qquad \qquad (b) 247.6 + 359.4$$

**Solution**

$$(a) \begin{array}{r} 37 \\ +28 \\ \hline 65 \end{array} \Rightarrow \begin{array}{r} 0110\ 1010 \\ +0101\ 1011 \\ \hline 1011\ \textcircled{1}0101 \\ \quad +1 \swarrow \\ 1100\ 0101 \\ -0011\ +0011 \\ \hline 1001\ 1000 \end{array} \begin{array}{l} (37 \text{ in XS-3}) \\ (28 \text{ in XS-3}) \\ (\text{Carry generated}) \\ (\text{Propagate carry}) \\ (\text{Add 0011 to correct 0101 and} \\ \text{subtract 0011 to correct 1100}) \\ (\text{Corrected sum in XS-3} = 65_{10}) \end{array}$$

$$\begin{array}{r}
 \text{(b) } 247.6 \Rightarrow \begin{array}{cccc} 0101 & 0111 & 1010 & .1001 \end{array} \quad (\text{247.6 in XS-3}) \\
 +359.4 \quad \begin{array}{cccc} +0110 & 1000 & 1100 & .0111 \end{array} \quad (\text{359.4 in XS-3}) \\
 \hline
 607.0 \quad \begin{array}{cccc} 1011 & 1111 & \textcircled{1}0110 & \textcircled{1}.0000 \\ & +1 \swarrow & +1 \swarrow & \\ \hline & 1011 & \textcircled{1}0000 & 0111 & .0000 \\ & +1 \swarrow & & & \\ \hline & 1100 & 0000 & 0111 & .0000 \\ -0011 & +0011 & +0011 & +.0011 & \\ \hline & 1001 & 0011 & 1010 & .0011 \end{array} \quad (\text{Add 0011 to 0000, 0111, 0000} \\
 \quad \quad \quad \text{and subtract 0011 from 1100}) \\
 \quad \quad \quad (\text{Corrected sum in XS-3} = 607.0_{10})
 \end{array}$$

### 3.3.2 XS-3 Subtraction

To subtract in XS-3, subtract the XS-3 numbers by subtracting each 4-bit group of the subtrahend from the corresponding 4-bit group of the minuend starting from the LSD. If there is no borrow from the next 4-bit group, add 0011 to the difference term of such groups (because when decimal digits are subtracted in XS-3 and there is no borrow, the result is in normal binary). If there is a borrow, subtract 0011 from the difference term (because taking a borrow is equivalent to adding six invalid states, so the result is in XS-6).

**EXAMPLE 3.6** Perform the following subtractions in XS-3 code.

$$\text{(a) } 267 - 175 \quad \text{(b) } 57.6 - 27.8$$

**Solution**

$$\begin{array}{r}
 \text{(a) } \begin{array}{r} 267 \Rightarrow \begin{array}{ccc} 0101 & 1001 & 1010 \end{array} \quad (\text{267 in XS-3}) \\ -175 \quad \begin{array}{ccc} -0100 & 1010 & 1000 \end{array} \quad (\text{175 in XS-3}) \\ \hline 092 \quad \begin{array}{ccc} 0000 & 1111 & 0010 \\ +0011 & -0011 & +0011 \\ \hline 0011 & 1100 & 0101 \end{array} \quad (\text{Correct 0010 and 0000 by adding 0011 and} \\ \quad \quad \quad \text{correct 1111 by subtracting 0011}) \\ \quad \quad \quad (\text{Corrected difference in XS-3} = 92_{10}) \end{array} \\
 \text{(b) } \begin{array}{r} 57.6 \Rightarrow \begin{array}{ccc} 1000 & 1010 & .1001 \end{array} \quad (\text{57.6 in XS-3}) \\ -27.8 \quad \begin{array}{ccc} -0101 & 1010 & .1011 \end{array} \quad (\text{27.8 in XS-3}) \\ \hline 29.8 \quad \begin{array}{ccc} 0010 & 1111 & .1110 \\ +0011 & -0011 & -.0011 \\ \hline 0101 & 1110 & .1011 \end{array} \quad (\text{Correct 0010 by adding 0011 and correct} \\ \quad \quad \quad 1110 and 1111 by subtracting 0011) \\ \quad \quad \quad (\text{Corrected difference in XS-3} = 29.8_{10}) \end{array} \end{array}$$

### 3.3.3 XS-3 Subtraction Using 9's and 10's Complement Methods

In practice, subtraction is performed by the 9's complement or the 10's complement method.

**EXAMPLE 3.7** Perform the following subtractions in XS-3 code using the 9's complement method.

$$\text{(a) } 687 - 348 \quad \text{(b) } 246 - 592$$

**Solution**

(a) The subtrahend (348) in XS-3 code and its complements are:

9's complement of 348 = 651

XS-3 code of 348 = 0110 0111 1011

1's complement of 348 in XS-3 = 1001 1000 0100

XS-3 code of 687 = 1001 1011 1010

$$\begin{array}{r}
 687 \\
 -348 \\
 \hline
 339
 \end{array} \Rightarrow \begin{array}{r}
 687 \\
 +651 \\
 \hline
 \textcircled{1}338
 \end{array} \begin{array}{l}
 \text{(9's complement of 348)} \\
 \text{+1} \curvearrowleft \text{(End around carry)} \\
 \hline
 339 \quad \text{(Corrected difference in decimal)}
 \end{array}$$

$$\begin{array}{rccccc}
 1001 & 1011 & 1010 & \text{(687 in XS-3)} \\
 +1001 & 1000 & 0100 & \text{(1's complement of 348 in XS-3)} \\
 \hline
 \textcircled{1}0010 \textcircled{1}0011 & 1110 & \text{(Carry generated)} \\
 +1 \curvearrowleft & +1 \curvearrowleft & \text{(Propagate carry)} \\
 \hline
 \textcircled{1} & 0011 & 0011 & 1110 \\
 \curvearrowleft & & & +1 \\
 \hline
 0011 & 0011 & 1111 & \text{(Correct 1111 by subtracting 0011 and} \\
 +0011 & +0011 & -0011 & \text{correct both groups of 0011 by adding 0011)} \\
 \hline
 0110 & 0110 & 1100 & \text{(Corrected difference in XS-3 = } 339_{10})
 \end{array}$$

(b) The subtrahend (592) in XS-3 and its complements are:

9's complement of 592 = 407

XS-3 code of 592 = 1000 1100 0101

1's complement of 592 in XS-3 = 0111 0011 1010

XS-3 code of 246 = 0101 0111 1001

$$\begin{array}{r}
 246 \\
 -592 \\
 \hline
 -346
 \end{array} \Rightarrow \begin{array}{r}
 246 \\
 +407 \\
 \hline
 653
 \end{array} \begin{array}{l}
 \text{(9's complement of 592)} \\
 \text{(No carry)}
 \end{array}$$

As there is no carry, the result is negative and is in its 9's complement form. The 9's complement of 653 is 346. So, the result is -346.

$$\begin{array}{rccccc}
 0101 & 0111 & 1001 & \text{(246 in XS-3)} \\
 +0111 & 0011 & 1010 & \text{(1's complement of 592 in XS-3)} \\
 \hline
 1100 & 1010 & \textcircled{1}0011 & \text{(Propagate carry)} \\
 +1 \curvearrowleft & & & \\
 \hline
 1100 & 1011 & 0011 & \text{(Correct 0011 by adding 0011 and correct} \\
 -0011 & -0011 & +0011 & \text{1011 and 1100 by subtracting 0011)} \\
 \hline
 1001 & 1000 & 0110 & \text{(No carry)}
 \end{array}$$

The sum is the XS-3 form of 653. There is no carry. So, the result is negative and is in 1's complement form. The 1's complement of the sum is 0110 0111 1001 in XS-3 code ( $346_{10}$ ). So, the answer is -346.

**EXAMPLE 3.8** Perform the following subtractions in XS-3 code using the 10's complement method.

$$(a) 597 - 239 \quad (b) 354 - 672$$

**Solution**

$$(a) 10's complement of 239 = 761$$

$$\text{XS-3 code of } 239 = 0101\ 0110\ 1100$$

$$2's \text{ complement of } 239 \text{ in XS-3 code} = 1010\ 1001\ 0100$$

$$\text{XS-3 code of } 597 = 1000\ 1100\ 1010$$

$$\begin{array}{r} 597 \\ -239 \\ \hline 358 \end{array} \Rightarrow \begin{array}{r} 597 \\ +761 \\ \hline 1358 \end{array} \begin{array}{l} (10's \text{ complement of } 239) \\ (\text{Ignore carry}) \\ (358 \text{ (Corrected difference in decimal)}) \end{array}$$

$$\begin{array}{r} 1000 & 1100 & 1010 \\ +1010 & 1001 & 0100 \\ \hline 10010 & 10101 & 1110 \end{array} \begin{array}{l} (597 \text{ in XS-3}) \\ (2's \text{ complement of } 239 \text{ in XS-3}) \\ (\text{Propagate carry}) \end{array}$$

$$\begin{array}{r} +1 \swarrow +1 \swarrow \\ \hline 1 & 0011 & 0101 & 1110 \end{array} \begin{array}{l} (\text{Ignore carry}) \\ (\text{Correct } 1110 \text{ by subtracting } 0011 \text{ and correct } 0101 \text{ and } 0011 \text{ by adding } 0011) \\ (\text{Corrected difference in XS-3 code} = 358) \end{array}$$

$$(b) 10's complement of 672 = 328$$

$$\text{XS-3 code of } 672 = 1001\ 1010\ 0101$$

$$2's \text{ complement of } 672 \text{ in XS-3 code} = 0110\ 0101\ 1011$$

$$\text{XS-3 code of } 354 = 0110\ 1000\ 0111$$

$$\begin{array}{r} 354 \\ -672 \\ \hline -318 \end{array} \Rightarrow \begin{array}{r} 354 \\ +328 \\ \hline 682 \end{array} \begin{array}{l} (10's \text{ complement of } 672) \\ (\text{No carry}) \end{array}$$

No carry. So, the result is negative and is in its 10's complement form. The 10's complement of 682 is 318. So, the answer is  $-318$ .

$$\begin{array}{r} 0110 & 1000 & 0111 \\ +0110 & 0101 & 1011 \\ \hline 1100 & 1101 & 10010 \end{array} \begin{array}{l} (354 \text{ in XS-3}) \\ (2's \text{ complement of } 672 \text{ in XS-3}) \\ (\text{Propagate carry}) \end{array}$$

$$\begin{array}{r} +1 \swarrow \\ \hline 1100 & 1110 & 0010 \\ -0011 & -0011 & +0011 \\ \hline 1001 & 1011 & 0101 \end{array} \begin{array}{l} (\text{Correct } 0010 \text{ by adding } 0011 \text{ and correct } 1110 \text{ and } 1100 \text{ by subtracting } 0011) \\ (\text{No carry}) \end{array}$$

The sum is the XS-3 form of 682. There is no carry. So, the result is negative and is in the 2's complement form. The 2's complement of the sum is  $0110\ 0100\ 1011$  in XS-3 code ( $318_{10}$ ). So, the answer is  $-318$ .

**EXAMPLE 3.9** Encode the decimal digits 0, 1, 2, . . . , 9 by means of weighted codes 3321, 4221, 731–2, 631–1, 5311, 74–2–1 and 7421.

**Solution**

The encoding is shown in Table 3.2.

**Table 3.2** Encoded decimal digits

Decimal	3 3 2 1	4 2 2 1	7 3 1 –2	6 3 1 –1	5 3 1 1	7 4 –2 –1	7 4 2 1
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1	0 0 1 0	0 0 1 0	0 0 0 1	0 1 1 1	0 0 0 1
2	0 0 1 0	0 0 1 0	0 1 1 1	0 1 0 1	0 0 1 1	0 1 1 0	0 0 1 0
3	0 0 1 1	0 0 1 1	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 1	0 0 1 1
4	0 1 0 1	0 1 1 0	0 1 1 0	0 1 1 0	0 1 0 1	0 1 0 0	0 1 0 0
5	1 0 1 0	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 0	1 0 1 0	0 1 0 1
6	1 1 0 0	1 1 0 0	1 0 1 1	1 0 1 1	1 0 0 1	1 0 0 1	0 1 1 0
7	1 1 0 1	1 1 0 1	1 0 0 0	1 0 1 0	1 0 1 1	1 0 0 0	1 0 0 0
8	1 1 1 0	1 1 1 0	1 1 0 1	1 1 0 1	1 1 0 0	1 1 1 1	1 0 0 1
9	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 0 1	1 1 1 0	1 0 1 0

### 3.4 THE GRAY CODE (REFLECTIVE-CODE)

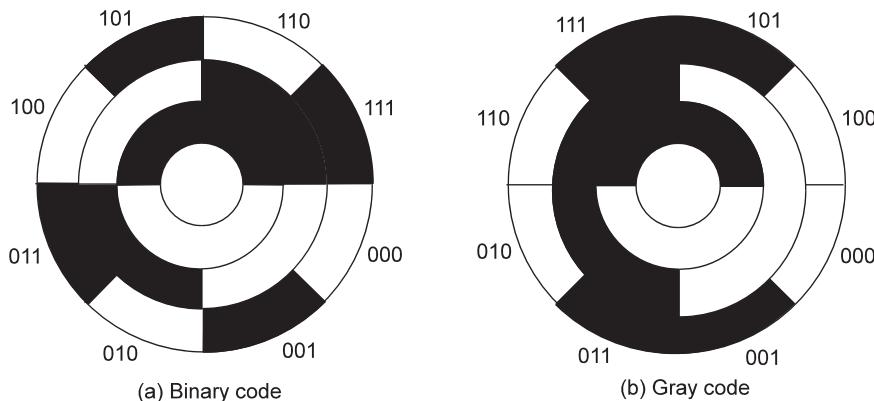
The Gray code is a non-weighted code, and is not suitable for arithmetic operations. It is not a BCD code. It is a cyclic code because successive code words in this code differ in one bit position only, i.e. it is a unit distance code. It is the most popular of the unit distance codes. It is also a reflective code, i.e. it is both reflective and unit distance. The  $n$  least significant bits for  $2^n$  through  $2^{n+1} - 1$  are the mirror images of those for 0 through  $2^n - 1$ . An  $N$ -bit Gray code can be obtained by reflecting an  $N - 1$  bit code about an axis at the end of the code, and putting the MSB of 0 above the axis and the MSB of 1 below the axis. Reflection of Gray codes is shown in Table 3.3. Another property of the Gray code is that the Gray-coded number corresponding to the decimal number  $2^n - 1$  for any  $n$  differs from Gray-coded 0 in one bit position only. This property places the Gray code for the largest  $N$ -bit binary number at unit distance from 0. It is easier to determine the pattern assignment corresponding to an arbitrary number, or decode an arbitrary number Gray code pattern using the conversions to and from binary. In fact, one reason for the popularity of the Gray code is its ease of conversion to and from binary. Gray codes are used in instrumentation and data acquisition systems where linear or angular displacement is measured. They are also used in shaft encoders, I/O devices, A/D converters and other peripheral equipment.

Consider a rotating disk that provides an output of its position in 3-bit binary (Figure 3.2). When the brushes are on the black part, they output a 1, and when they are on a white sector they output a 0. Suppose the disk is coded in binary; consider now what happens when the brushes are on the 111 sector and almost ready to enter the 000 sector. If one brush were slightly ahead of the other, say the 4's brush, the position would be indicated by a 011 instead of a 111 or 000. Therefore, a  $180^\circ$  error in disk position would result. Since it is physically impossible to have all the brushes precisely aligned, some error would always be present at the edges of the sectors. If the disks were coded in gray, a similar error would make the disk to be read with a very small error. For example

if the brushes are on the sector 010 and almost ready to enter the 110 sector and if the 4's brush is slightly ahead, the position would be indicated by 110 instead of 010 resulting in a very small error. Figure 3.2 illustrates this operation.

**Table 3.3** Reflection of Gray codes

Gray Code				Decimal	4-bit binary
1-bit	2-bit	3-bit	4-bit		
0	00	000	0000	0	0000
1	01	001	0001	1	0001
	11	011	0011	2	0010
	10	010	0010	3	0011
		110	0110	4	0100
		111	0111	5	0101
		101	0101	6	0110
		100	0100	7	0111
			1100	8	1000
			1101	9	1001
			1111	10	1010
			1110	11	1011
			1010	12	1100
			1011	13	1101
			1001	14	1110
			1000	15	1111



**Figure 3.2** Position indicating system.

### 3.4.1 Binary-to-Gray Conversion

If an  $n$ -bit binary number is represented by  $B_n B_{n-1} \dots B_1$  and its Gray code equivalent by  $G_n G_{n-1} \dots G_1$ , where  $B_n$  and  $G_n$  are the MSBs, then the Gray code bits are obtained from the binary code as follows.

$G_n = B_n$	$G_{n-1} = B_n \oplus B_{n-1}$	$G_{n-2} = B_{n-1} \oplus B_{n-2} \dots$	$G_1 = B_2 \oplus B_1$
-------------	--------------------------------	--	------------------------

where the symbol  $\oplus$  stands for the Exclusive OR (X-OR) operation explained below.

The conversion procedure is as follows:

1. Record the MSB of the binary as the MSB of the Gray code.
2. Add the MSB of the binary to the next bit in binary, recording the sum and ignoring the carry, if any, i.e. X-OR the bits. This sum is the next bit of the Gray code.
3. Add the 2nd bit of the binary to the 3rd bit of the binary, the 3rd bit to the 4th bit, and so on.
4. Record the successive sums as the successive bits of the Gray code until all the bits of the binary number are exhausted.

Another way to convert a binary number to its Gray code is to Exclusive OR (i.e. to take the modulo sum of) the bits of the binary number with those of the binary number shifted one position to the right. The LSB of the shifted number is discarded and the MSB of the Gray code number is the same as the MSB of the original binary number.

**EXAMPLE 3.10** Convert the binary 1001 to the Gray code.

**Solution**

(a)	Binary	1	$\xrightarrow{-\oplus}$	0	$\xrightarrow{-\oplus}$	0	$\xrightarrow{-\oplus}$	1
	Gray	1		1		0		1
(b)	Binary			1 0 0 1				
	Shifted binary				1 0 0	1		
	Gray			1 1 0 1				

**Method I.** As shown in (a), record the 8's bit '1' (MSB) of the binary as the 8's bit of the Gray code. Then add the 8's bit of the binary to the 4's bit of the binary ( $1 + 0 = 1$ ). Record the sum as the 4's bit of the Gray code. Add the 4's bit of the binary to the 2's bit of the binary ( $0 + 0 = 0$ ). Record the sum as the 2's bit of the Gray code. Add the 2's bit of the binary to the 1's bit of the binary ( $0 + 1 = 1$ ). Record the sum as the 1's bit of the Gray code. The resultant Gray code is 1101.

**Method II.** As shown in (b), write the given binary number and add it to the same number shifted one place to the right. Record the MSB of the binary, i.e. 1 as the MSB of the Gray code. Add the subsequent columns ( $0 + 1 = 1$ ;  $0 + 0 = 0$ ;  $1 + 0 = 1$ ) and record the corresponding sums as the subsequent significant bits of the Gray code. Ignore the bit shifted out. The resultant Gray code is 1101.

### 3.4.2 Gray-to-Binary Conversion

If an  $n$ -bit Gray number is represented by  $G_n G_{n-1} \dots G_1$  and its binary equivalent by  $B_n B_{n-1} \dots B_1$ , then binary bits are obtained from Gray bits as follows:

$B_n = G_n$	$B_{n-1} = B_n \oplus G_{n-1}$	$B_{n-2} = B_{n-1} \oplus G_{n-2} \dots$	$B_1 = B_2 \oplus G_1$
-------------	--------------------------------	--	------------------------

The conversion procedure is:

1. The MSB of the binary number is the same as the MSB of the Gray code number; record it.
2. Add the MSB of the binary to the next significant bit of the Gray code, i.e. X-OR them; record the sum and ignore the carry.
3. Add the 2nd bit of the binary to the 3rd bit of the Gray; the 3rd bit of the binary to the 4th bit of the Gray code, and so on, each time recording the sum and ignoring the carry.
4. Continue this till all the Gray bits are exhausted. The sequence of bits that has been written down is the binary equivalent of the Gray code number.

**EXAMPLE 3.11** Convert the Gray code 1101 to binary.

**Solution**

The conversion is done as shown below:

Gray	1	1	0	1			
		⊗		⊗		⊗	
Binary	1	0	0	1			

Record the 8's bit (MSB) of the Gray code as the 8's bit of the binary. Add the 8's bit of the binary to the 4's bit of the Gray code ( $1 + 1 = 10$ ) and record the sum (0) as the 4's bit of the binary (ignore the carry bit). Add the 4's bit of the binary to the 2's bit of the Gray code ( $0 + 0 = 0$ ) and record the sum as the 2's bit of the binary. Add the 2's bit of the binary to the 1's bit of the Gray code ( $0 + 1 = 1$ ) and record the sum as the 1's bit of the binary. The resultant binary number is 1001.

**EXAMPLE 3.12**

(a) Convert the following into the Gray number.

$$(i) 3A7_{16} \quad (ii) 527_8 \quad (iii) 652_{10}$$

(b) Convert the Gray number 1 0 1 1 0 0 1 0 into

$$(i) \text{hex} \quad (ii) \text{octal} \quad (iii) \text{decimal}$$

**Solution**

(a) To convert the given number in any system into Gray number first convert it into binary and then convert that binary number into the Gray code using the normal procedure.

$$(i) 3A7_{16} = 0011, 1010, 0111_2 = 1001110100 \text{ (Gray)}$$

$$(ii) 527_8 = 101, 011, 011_2 = 11110110 \text{ (Gray)}$$

$$(iii) 652_{10} = 1010001100_2 = 1111001010 \text{ (Gray)}$$

(b) To convert the given Gray number into any other number system, first convert the given Gray number into a binary number and then convert that binary number into the required number system.

$$10110010 \text{ (Gray)} = 1101100_2 = DC_{16} = 334_8 = 220_{10}$$

### 3.4.3 The XS-3 Gray Code

In a normal Gray code, the bit patterns for 0 (0000) and 9 (1101) do not have a unit distance between them. That is, they differ in more than one position. In XS-3 Gray code, each decimal

digit is encoded with the Gray code pattern of the decimal digit that is greater by 3. It has a unit distance between the patterns for 0 and 9. Table 3.4 shows the XS-3 Gray code for decimal digits 0 through 9.

**Table 3.4** The XS-3 Gray code for decimal digits 0–9

Decimal digit	XS-3 Gray code	Decimal digit	XS-3 Gray code
0	0 0 1 0	5	1 1 0 0
1	0 1 1 0	6	1 1 0 1
2	0 1 1 1	7	1 1 1 1
3	0 1 0 1	8	1 1 1 0
4	0 1 0 0	9	1 0 1 0

### 3.5 ERROR-DETECTING CODES

When binary data is transmitted and processed, it is susceptible to noise that can alter or distort its contents. The 1s may get changed to 0s and 0s to 1s. Because digital systems must be accurate to the digit, errors can pose a serious problem. Several schemes have been devised to detect the occurrence of a single-bit error in a binary word, so that whenever such an error occurs the concerned binary word can be corrected and retransmitted.

#### 3.5.1 Parity

The simplest technique for detecting errors is that of adding an extra bit, known as the *parity* bit, to each word being transmitted. There are two types of parity—odd parity and even parity. For odd parity, the parity bit is set to a 0 or a 1 at the transmitter such that the total number of 1 bits in the word including the parity bit is an odd number. For even parity, the parity bit is set to a 0 or a 1 at the transmitter such that the total number of 1 bits in the word including the parity bit is an even number. Table 3.5 shows the parity bits to be added to transmit decimal digits 0 through 9 in the 8421 code.

**Table 3.5** Odd and even parity in the 8421 BCD code

Decimal	8421 BCD	Odd parity	Even parity
0	0 0 0 0	1	0
1	0 0 0 1	0	1
2	0 0 1 0	0	1
3	0 0 1 1	1	0
4	0 1 0 0	0	1
5	0 1 0 1	1	0
6	0 1 1 0	1	0
7	0 1 1 1	0	1
8	1 0 0 0	0	1
9	1 0 0 1	1	0

When the digital data is received, a parity checking circuit generates an error signal if the total number of 1s is even in an odd-parity system or odd in an even-parity system. This parity check can always detect a single-bit error but cannot detect two or more errors within the same word. In any practical system, there is always a finite probability of the occurrence of single error. The probability that two or more errors will occur simultaneously, although non-zero, is substantially smaller. Odd parity is used more often than even parity because even parity does not detect the situation where all 0s are created by a short-circuit or some other fault condition.

If the code possesses the property by which the occurrence of any single-bit error transforms a valid code word into an invalid one, it is said to be an error-detecting (single bit) code. In general, to obtain an  $N$  bit error-detecting code, no more than half of the possible  $2^N$  combinations of digits can be used. Thus, to obtain an error-detecting code for 10 decimal digits, at least 5 binary digits are to be used. The code words are chosen in such a manner that in order to change one valid code word into another valid code word, at least two digits must be complemented. A code is an error-detecting code, if and only if its minimum distance is two or more. The *distance* between two words is defined as the number of digits that must change in a word so that the other word results. For example, the distance between 0011 and 1010 is 2, and the distance between 0111 and 1000 is 4.

**EXAMPLE 3.13** In an even-parity scheme, which of the following words contain an error?

- (a) 10101010      (b) 11110110      (c) 10111001

**Solution**

- (a) The number of 1s in the word is even (4). Therefore, there is no error.
- (b) The number of 1s in the word is even (6). Therefore, there is no error.
- (c) The number of 1s in the word is odd (5). So, this word has an error.

**EXAMPLE 3.14** In an odd-parity scheme, which of the following words contain an error?

- (a) 10110111      (b) 10011010      (c) 11101010

**Solution**

- (a) The number of 1s in the word is even (6). So, this word has an error.
- (b) The number of 1s in the word is even (4). So, this word has an error.
- (c) The number of 1s in the word is odd (5). Therefore, there is no error.

### 3.5.2 Check Sums

Simple parity cannot detect two errors within the same word. One way of overcoming this difficulty is to use a sort of two-dimensional parity. As each word is transmitted, it is added to the sum of the previously transmitted words, and the sum retained at the transmitter end. At the end of transmission, the sum (called the *check sum*) up to that time is sent to the receiver. The receiver can check its sum with the transmitted sum. If the two sums are the same, then no errors were detected at the receiver end. If there is an error, the receiving location can ask for retransmission of the entire data. This is the type of transmission used in teleprocessing systems.

### 3.5.3 Block Parity

When several binary words are transmitted or stored in succession, the resulting collection of bits can be regarded as a block of data, having rows and columns. Parity bits can then be assigned to

both rows and columns. This scheme makes it possible to *correct* any single error occurring in a data word and to *detect* any two errors in a word. The parity row is often called a parity word. Such a block parity technique, also called word parity, is widely used for data stored on magnetic tapes.

For example, six 8-bit words in succession can be formed into a  $6 \times 8$  block for transmission. Parity bits are added so that odd parity is maintained both row-wise and column-wise and the block is transmitted as a  $7 \times 9$  block as shown in Table A. At the receiving end, parity is checked both row-wise and column-wise and suppose errors are detected as shown in Table B. These single-bit errors detected can be corrected by complementing the error bit. In Table B, parity errors in the 3rd row and 5th column mean that the 5th bit in the 3rd row is in error. It can be corrected by complementing it.

Two errors as shown in Table C can only be detected but not corrected. In Table C, parity errors are observed in both columns 2 and 4. It indicates that in one row there are two errors.

**Table A**

0 1 0 1 1 0 1 1'0								
1 0 0 1 0 1 0 1'1								
0 1 1 0 1 1 1 0'0								
1 1 0 1 0 0 1 1'0								
1 0 0 0 1 1 0 1'1								
0 1 1 1 0 1 1 1'1								
Parity row → 0 1 1 1 0 1 1 0 0								
							↑	Parity column

**Table B**

0 1 0 1 1 0 1 1'0								
1 0 0 1 0 1 0 1'1								
0 1 1 0 0 1 1 0'0							← Parity error in 3rd row	
1 1 0 1 0 0 1 1'0								
1 0 0 0 1 1 0 1'1								
0 1 1 1 0 1 1 1'1								
0 1 1 1 0 1 1 0 0							↑	Parity error in 5th column

**Table C**

0 1 0 1 1 0 1 1'0								
1 0 0 1 0 1 0 1'1								
0 1 1 0 1 1 1 0'0								
1 0 0 0 0 0 1 1'0								
1 0 0 0 1 1 0 1'1								
0 1 1 1 0 1 1 1'1								
0 1 1 1 0 1 1 0 0							↑	Parity errors in 2nd and 4th columns

**EXAMPLE 3.15** The block of data shown in Table 3.6a is to be stored on a magnetic tape. Create the row and column parity bits for the data using odd parity.

**Solution**

The parity bit 0 or 1 is added column-wise and row-wise such that the total number of 1s in each column and row including the data bits and parity bit is odd as shown in Table 3.6b.

**Table 3.6a** Example 3.15

Data
1 0 1 1 0
1 0 0 0 1
1 0 1 0 1
0 0 0 1 0
1 1 0 0 0
0 0 0 0 0
1 1 0 1 0

**Table 3.6b** Example 3.15

Data	Parity bit
1 0 1 1 0	0
1 0 0 0 1	1
1 0 1 0 1	0
0 0 0 1 0	0
1 1 0 0 0	1
0 0 0 0 0	1
1 1 0 1 0	0
0 1 1 0 1	Parity bit

**3.5.4 Five-bit Codes**

Some 5-bit BCD codes that have parity contained within each code for ease of error detection are shown in Table 3.7. The 63210 is a weighted code (except for the decimal digit 0). It has the useful

**Table 3.7** Five-bit BCD codes

Decimal	6 3 2 1 0	2-out-of-5	Shift-counter	5 1 1 1 1
0	0 0 1 1 0	0 0 0 1 1	0 0 0 0 0	0 0 0 0 0
1	0 0 0 1 1	0 0 1 0 1	0 0 0 0 1	0 0 0 0 1
2	0 0 1 0 1	0 0 1 1 0	0 0 0 1 1	0 0 0 1 1
3	0 1 0 0 1	0 1 0 0 1	0 0 1 1 1	0 0 1 1 1
4	0 1 0 1 0	0 1 0 1 0	0 1 1 1 1	0 1 1 1 1
5	0 1 1 0 0	0 1 1 0 0	1 1 1 1 1	1 0 0 0 0
6	1 0 0 0 1	1 0 0 0 1	1 1 1 1 0	1 1 0 0 0
7	1 0 0 1 0	1 0 0 1 0	1 1 1 0 0	1 1 1 0 0
8	1 0 1 0 0	1 0 1 0 0	1 1 0 0 0	1 1 1 1 0
9	1 1 0 0 0	1 1 0 0 0	1 0 0 0 0	1 1 1 1 1

error-detecting property that there are exactly two 1s in each code group. This code is used for storing data on magnetic tapes. The 2-out-of-5 code is a non-weighted code. It also has exactly two 1s in each code group. This code is used in the telephone and communication industries. At the receiving end, the receiver can check the number of 1s in each character received. The shift-counter code, also called the Johnson code, has the bit patterns produced by a 5-bit Johnson counter. The 51111 code is similar to the Johnson code but is weighted.

### 3.5.5 The Biquinary Code

The biquinary code shown in Table 3.8 is a weighted 7-bit BCD code. It is a parity data code. Note that each code group can be regarded as consisting of a 2-bit subgroup and a 5-bit subgroup, and each of these subgroups contains a single 1. Thus, it has the error-checking feature, for each code group has exactly two 1s and each subgroup has exactly one 1. The weights of the bit positions are 50 43210. Since there are two positions with weight 0, it is possible to encode decimal 0 with a group containing 1s, unlike other weighted codes. The biquinary code is used in the Abacus.

**Table 3.8** The biquinary code

Decimal digit	Biquinary code						
	5	0	4	3	2	1	0
0	0	1	0	0	0	0	1
1	0	1	0	0	0	1	0
2	0	1	0	0	1	0	0
3	0	1	0	1	0	0	0
4	0	1	1	0	0	0	0
5	1	0	0	0	0	0	1
6	1	0	0	0	0	1	0
7	1	0	0	0	1	0	0
8	1	0	0	1	0	0	0
9	1	0	1	0	0	0	0

### 3.5.6 The Ring-counter Code

A 10-bit ring counter will produce a sequence of 10-bit groups having the property that each group has a single 1. The ring-counter code shown in Table 3.9 is the code obtained by assigning a decimal digit to each of those ten patterns. It is a weighted code (9 8 7 6 5 4 3 2 1 0) because each bit position has a weight equal to one of the 10 decimal digits. Although this code is inefficient (for 1024 numbers can be encoded in pure binary with 10 bits as against 10 numbers only in ring-counter code), it has excellent error-detecting properties and is easier to implement.

**Table 3.9** The ring-counter code

Decimal digit	Ring-counter code									
	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	0	1	0	0
3	0	0	0	0	0	0	1	0	0	0
4	0	0	0	0	0	1	0	0	0	0
5	0	0	0	0	1	0	0	0	0	0
6	0	0	0	1	0	0	0	0	0	0
7	0	0	1	0	0	0	0	0	0	0
8	0	1	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0

## 3.6 ERROR-CORRECTING CODES

A code is said to be an error-correcting code, if the correct code word can always be deduced from an erroneous word. For a code to be a single-bit error-correcting code, the minimum distance of that code must be three. The minimum distance of a code is the smallest number of bits by which any two code words must differ. A code with minimum distance of three can not only correct single-bit errors, but also detect (but cannot correct) two-bit errors. The key to error correction is that it must be possible to detect and locate erroneous digits. If the location of an error has been determined, then by complementing the erroneous digit, the message can be corrected. One type of error-correcting code is the Hamming code. In this code, to each group of  $m$  information or message or data bits,  $k$  parity-checking bits denoted by  $P_1, P_2, \dots, P_k$  located at positions  $2^{k-1}$  from left are added to form an  $(m+k)$ -bit code word. To correct the error,  $k$  parity checks are performed on selected digits of each code word, and the position of the error bit is located by forming an error word, and the error bit is then complemented. The  $k$ -bit error word is generated by putting a 0 or a 1 in the  $2^{k-1}$ th position depending upon whether the check for parity involving the parity bit  $P_k$  is satisfied or not. The bits to be checked for parity can be noted from Table 3.10. This table lists the error positions and corresponding values of the position number for 15-bit, 12-bit, and 7-bit Hamming codes.

### 3.6.1 The 7-bit Hamming Code

To transmit four data bits, three parity bits located at positions  $2^0, 2^1$ , and  $2^2$  from left are added to make a 7-bit code word which is then transmitted. The word format would be as shown below:

$$P_1 \quad P_2 \quad D_3 \quad P_4 \quad D_5 \quad D_6 \quad D_7$$

where the  $D$  bits are the data bits and the  $P$  bits are the parity bits. From Table 1.15 we observe that  $P_1$  is to be set to a 0 or a 1 so that it establishes even parity over bits 1, 3, 5, and 7 (i.e.  $P_1 D_3 D_5 D_7$ ).  $P_2$  is to be set to a 0 or a 1 so that it establishes even parity over bits 2, 3, 6, and 7 (i.e.  $P_2 D_3 D_6 D_7$ ).  $P_4$  is to be set to a 0 or a 1 so that it establishes even parity over bits 4, 5, 6, and 7 (i.e.  $P_4 D_5 D_6 D_7$ ).

**Table 3.10** Error positions and their corresponding values

Error position	For 15-bit code				For 12-bit code				For 7-bit code		
	C <sub>4</sub>	C <sub>3</sub>	C <sub>2</sub>	C <sub>1</sub>	C <sub>4</sub>	C <sub>3</sub>	C <sub>2</sub>	C <sub>1</sub>	C <sub>3</sub>	C <sub>2</sub>	C <sub>1</sub>
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	1	1	0	1	1
4	0	1	0	0	0	1	0	0	1	0	0
5	0	1	0	1	0	1	0	1	1	0	1
6	0	1	1	0	0	1	1	0	1	1	0
7	0	1	1	1	0	1	1	1	1	1	1
8	1	0	0	0	1	0	0	0	0	0	0
9	1	0	0	1	1	0	0	1	0	0	1
10	1	0	1	0	1	0	1	0	1	0	0
11	1	0	1	1	1	0	1	1	1	1	1
12	1	1	0	0	1	1	0	0	0	0	0
13	1	1	0	1	0	0	0	0	0	0	0
14	1	1	1	0	0	0	0	0	0	0	0
15	1	1	1	1	1	1	1	1	1	1	1

The 7-bit Hamming code for the decimal digits coded in BCD and XS-3 codes is shown in Table 3.11. The minimum distance of the 7-bit Hamming code for BCD code is 3 as observed from that table.

**Table 3.11** The 7-bit hamming code for the decimal digits coded in BCD and XS-3 codes

Decimal digit	Hamming code bits							For Excess-3						
	For BCD							For Excess-3						
	P <sub>1</sub>	P <sub>2</sub>	D <sub>3</sub>	P <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>	P <sub>1</sub>	P <sub>2</sub>	D <sub>3</sub>	P <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>
0	0	0	0	0	0	0	0	1	0	0	0	0	1	1
1	1	1	0	1	0	0	1	1	0	0	1	1	0	0
2	0	1	0	1	0	1	1	0	1	0	0	1	0	1
3	1	0	0	0	0	1	1	1	1	0	0	1	1	0
4	1	0	0	1	1	0	0	0	0	0	1	1	1	1
5	0	1	0	0	1	0	1	1	1	1	0	0	0	0
6	1	1	0	0	1	1	0	0	0	1	1	0	0	1
7	0	0	0	1	1	1	1	1	0	1	1	1	0	1
8	1	1	1	0	0	0	0	0	1	1	0	0	1	1
9	0	0	1	1	0	0	1	0	1	1	1	1	0	0

At the receiving end, the message received in the Hamming code is decoded to see if any errors have occurred. Bits 1, 3, 5, 7, bits 2, 3, 6, 7, and bits 4, 5, 6, 7 are all checked for even parity. If they all check out, there is no error. If there is an error, the error bit can be located by forming a

3-bit binary number  $c_3c_2c_1$  (where  $c_1 = P_1 \oplus D_3 \oplus D_5 \oplus D_7$ ;  $c_2 = P_2 \oplus D_3 \oplus D_6 \oplus D_7$ ; and  $c_3 = D_4 \oplus D_5 \oplus D_6 \oplus D_7$ ) out of the three parity checks and the error bit is then corrected by complementing it.

Assume that a 7-bit Hamming code is received as 1111001. Analyzing the bits 1, 3, 5, 7 (1 1 0 1) of which  $P_1$  is a parity bit, an error is detected. So, put a 1 in the 1's position of the error word, i.e.  $c_1 = 1$ . Analyzing the bits 2, 3, 6, 7 (1 1 0 1) of which  $P_2$  is a parity bit, an error is detected. So, put a 1 in the 2's position of the error word, i.e.  $c_2 = 1$ . Analyzing the bits 4, 5, 6, 7 (1 0 0 1) of which  $P_4$  is a parity bit, no error is detected. So, put a 0 in the 4's position of the error word, i.e.  $c_3 = 0$ . So, the error word is  $c_3c_2c_1 = 011 = 3_{10}$ , a decimal 3. Therefore, the bit in position 3 (from left) is in error; complement it. The corrected code should be read as 1 1 0 1 0 0 1.

**EXAMPLE 3.16** Encode data bits 1101 into the 7-bit even-parity Hamming code.

**Solution**

The bit pattern is

$P_1$	$P_2$	$D_3$	$P_4$	$D_5$	$D_6$	$D_7$
1			1	1	0	1

Bits 1, 3, 5, 7 (i.e.  $P_1 1 1 1$ ) must have even parity. So,  $P_1$  must be a 1.

Bits 2, 3, 6, 7 (i.e.  $P_2 1 0 1$ ) must have even parity. So,  $P_2$  must be a 0.

Bits 4, 5, 6, 7 (i.e.  $P_4 1 0 1$ ) must have even parity. So,  $P_4$  must be a 0.

Therefore, the final code is 1 0 1 0 1 0 1.

**EXAMPLE 3.17** The message below coded in the 7-bit Hamming code is transmitted through a noisy channel. Decode the message assuming that at most a single error occurred in each code word.

1001001, 0111001, 1110110, 0011011

**Solution**

The given data is of 28 bits; split it into four groups of 7 bits each and correct for the error, if any, in each group and write the corrected data.

*First group is 1 0 0 1 0 0 1*

1, 3, 5, 7 (1001)	$\rightarrow$	no error	$\rightarrow$	put a 0 in the 1's position. $c_1 = 0$
2, 3, 6, 7 (0001)	$\rightarrow$	error	$\rightarrow$	put a 1 in the 2's position. $c_2 = 1$
4, 5, 6, 7 (1001)	$\rightarrow$	no error	$\rightarrow$	put a 0 in the 4's position. $c_3 = 0$

The error word is  $c_3c_2c_1 = 010 = 2_{10}$ . So, complement the 2nd bit (from left). Therefore, the correct code is 1 1 0 1 0 0 1.

*Second group is 0 1 1 1 0 0 1*

1, 3, 5, 7 (0101)	$\rightarrow$	no error	$\rightarrow$	put a 0 in the 1's position. $c_1 = 0$
2, 3, 6, 7 (1101)	$\rightarrow$	error	$\rightarrow$	put a 1 in the 2's position. $c_2 = 1$
4, 5, 6, 7 (10 01)	$\rightarrow$	no error	$\rightarrow$	put a 0 in the 4's position. $c_3 = 0$

The error word is  $c_3c_2c_1 = 010 = 2_{10}$ . So, complement the 2nd bit (from left). Therefore, the correct code is 0 0 1 1 0 0 1.

*Third group is 1 1 1 0 1 1 0*

1, 3, 5, 7 (1110)	→	error	→	put a 1 in the 1's position. $c_1 = 1$
2, 3, 6, 7 (1110)	→	error	→	put a 1 in the 2's position. $c_2 = 1$
4, 5, 6, 7 (0110)	→	no error	→	put a 0 in the 4's position. $c_3 = 0$

The error word is  $c_3c_2c_1 = 011 = 3_{10}$ . So, complement the 3rd bit (from left). Therefore, the correct code is 1 1 0 0 1 1 0

*Fourth group is 0 0 1 1 0 1 1*

1, 3, 5, 7 (0101)	→	no error	→	put a 0 in the 1's position. $c_1 = 0$
2, 3, 6, 7 (0111)	→	error	→	put a 1 in the 2's position. $c_2 = 1$
4, 5, 6, 7 (1011)	→	error	→	put a 1 in the 4's position. $c_3 = 1$

The error word is  $c_3c_2c_1 = 110 = 6_{10}$ . So, complement the 6th bit (from left). Therefore, the correct code is 0 0 1 1 0 0 1. So the decoded message is 1101001, 0011001, 1100110, 0011001.

### 3.6.2 The 15-bit Hamming Code

To transmit eleven data bits, four parity bits located at positions  $2^0, 2^1, 2^2$  and  $2^3$  from left are added to make a 15-bit code word which is then transmitted. The word format would be as shown below:

P<sub>1</sub> P<sub>2</sub> D<sub>3</sub> P<sub>4</sub> D<sub>5</sub> D<sub>6</sub> D<sub>7</sub> P<sub>8</sub> D<sub>9</sub> D<sub>10</sub> D<sub>11</sub> D<sub>12</sub> D<sub>13</sub> D<sub>14</sub> D<sub>15</sub>

where the D bits are the data bits and the P bits are the parity bits. From Table 3.10 we observe that P<sub>1</sub> is to be set to a 0 or a 1 so that it establishes even parity over bits 1, 3, 5, 7, 9, 11, 13 and 15 (i.e. over bits P<sub>1</sub> D<sub>3</sub> D<sub>5</sub> D<sub>7</sub> D<sub>9</sub> D<sub>11</sub> D<sub>13</sub> D<sub>15</sub>).

P<sub>2</sub> is to be set to a 0 or a 1 so that it establishes even parity over bits 2, 3, 6, 7, 10, 11, 14 and 15 (i.e. over bits P<sub>2</sub> D<sub>3</sub> D<sub>6</sub> D<sub>7</sub> D<sub>10</sub> D<sub>11</sub> D<sub>14</sub> D<sub>15</sub>).

P<sub>4</sub> is to be set to a 0 or a 1 so that it establishes even parity over bits 4, 5, 6, 7, 12, 13, 14 and 15 (i.e. over bits P<sub>4</sub> D<sub>5</sub> D<sub>6</sub> D<sub>7</sub> D<sub>12</sub> D<sub>13</sub> D<sub>14</sub> D<sub>15</sub>).

P<sub>8</sub> is to be set to a 0 or a 1 so that it establishes even parity over bits 8, 9, 10, 11, 12, 13, 14 and 15 (i.e. over bits P<sub>8</sub> D<sub>9</sub> D<sub>10</sub> D<sub>11</sub> D<sub>12</sub> D<sub>13</sub> D<sub>14</sub> D<sub>15</sub>).

**EXAMPLE 3.18** (a) Device a single error correcting code for a 11-bit group 01101110101.

(b) Test the following Hamming code sequence for 11-bit message and correct it if necessary (101001011101011).

**Solution**

(a) For the given 11-bit group the bit pattern is

P <sub>1</sub>	P <sub>2</sub>	D <sub>3</sub>	P <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>	P <sub>8</sub>	D <sub>9</sub>	D <sub>10</sub>	D <sub>11</sub>	D <sub>12</sub>	D <sub>13</sub>	D <sub>14</sub>	D <sub>15</sub>
0			1	1	1	0		1	1	1	0	1	0	1

We know that the parity bits P<sub>1</sub>, P<sub>2</sub>, P<sub>4</sub>, and P<sub>8</sub> are to be selected such that:

Bits 1,3,5,7,9,11,13,15 (i.e. P<sub>1</sub> 0101111) must have even parity. So P<sub>1</sub> must be a 1.

Bits 2,3,6,7,10,11,14,15 (i.e. P<sub>2</sub> 0101101) must have even parity. So P<sub>2</sub> must be a 0.

Bits 4,5,6,7,12,13,14,15 (i.e. P<sub>4</sub> 1100101) must have even parity. So P<sub>4</sub> must be a 0.

Bits 8,9,10,11,12,13,14,15 (i.e. P<sub>8</sub> 1110101) must have even parity. So P<sub>8</sub> must be a 1.

Putting the above parity bits the final code word is 100011011110101.

(b) The given Hamming code sequence for 11-bit message is

$P_1$	$P_2$	$D_3$	$P_4$	$D_5$	$D_6$	$D_7$	$P_8$	$D_9$	$D_{10}$	$D_{11}$	$D_{12}$	$D_{13}$	$D_{14}$	$D_{15}$
1	0	1	0	0	1	0	1	1	1	0	1	0	1	1

Bits 1, 3, 5, 7, 9, 11, 13, 15 (11001001)  $\rightarrow$  no error  $\rightarrow$  put a 0 in the 1's position.  $c_1 = 0$   
 Bits 2, 3, 6, 7, 10, 11, 14, 15 (01101011)  $\rightarrow$  error  $\rightarrow$  put a 1 in the 2's position.  $c_2 = 1$   
 Bits 4, 5, 6, 7, 12, 13, 14, 15 (00101011)  $\rightarrow$  no error  $\rightarrow$  put a 0 in the 4's position.  $c_3 = 0$   
 Bits 8, 9, 10, 11, 12, 13, 14, 15 (11101011)  $\rightarrow$  no error  $\rightarrow$  put a 0 in the 8's position.  $c_4 = 0$

The error word is  $c_4 c_3 c_2 c_1 = 2_{10}$ . So the 2nd bit (from left) is in error. Complement it. Therefore, the correct code is 111001011101011.

### 3.6.3 The 12-bit Hamming Code

To transmit eight data bits, four parity bits located at positions  $2^0, 2^1, 2^2$  and  $2^3$  from left are added to make a 12-bit code word which is then transmitted. The word format would be as shown below:

$P_1$	$P_2$	$D_3$	$P_4$	$D_5$	$D_6$	$D_7$	$P_8$	$D_9$	$D_{10}$	$D_{11}$	$D_{12}$
-------	-------	-------	-------	-------	-------	-------	-------	-------	----------	----------	----------

where the D bits are the data bits and the P bits are the parity bits. From Table 3.10 we observe that  $P_1$  is to be set to a 0 or a 1 so that it establishes even parity over bits 1, 3, 5, 7, 9, and 11 (i.e. over bits  $P_1 D_3 D_5 D_7 D_9 D_{11}$ ).

$P_2$  is to be set to a 0 or a 1 so that it establishes even parity over bits 2, 3, 6, 7, 10, and 11 (i.e. over bits  $P_2 D_3 D_6 D_7 D_{10} D_{11}$ ).

$P_4$  is to be set to a 0 or a 1 so that it establishes even parity over bits 4, 5, 6, 7, and 12 (i.e. over bits  $P_4 D_5 D_6 D_7 D_{12}$ ).

$P_8$  is to be set to a 0 or a 1 so that it establishes even parity over bits 8, 9, 10, 11, and 12 (i.e. over bits  $P_8 D_9 D_{10} D_{11} D_{12}$ ).

**EXAMPLE 3.19** (a) Given the 8-bit data word 01011011, generate the 12-bit composite word for the Hamming code that corrects and detects single errors.

(b) A 12-bit Hamming Code word containing 8 bits of data and 4 parity bits is read from memory. What is the original 8-bit word if the 12-bit read out is as follows?

- (i) 1000 1110 1010
- (ii) 1011 1000 0110
- (iii) 1011 1111 0100

**Solution**

(a) For a 8-bit data word, 4 parity bits located at positions  $2^0, 2^1, 2^2$ , and  $2^3$  from left are added to generate the composite word for the Hamming code that corrects and detects single errors. The format and the data bits are shown below.

$P_1$	$P_2$	$D_3$	$P_4$	$D_5$	$D_6$	$D_7$	$P_8$	$D_9$	$D_{10}$	$D_{11}$	$D_{12}$
0		1	0	1			1	0	1	1	

$P_1$  is to be selected so that bits 1, 3, 5, 7, 9, 11 ( $P_1 01111$ ) have even parity. So  $P_1 = 0$ .

## 110 FUNDAMENTALS OF DIGITAL CIRCUITS

$P_2$  is to be selected so that bits 2, 3, 6, 7, 10, 11 ( $P_2$  00101) have even parity . So  $P_2 = 0$ .

$P_4$  is to be selected so that bits 4, 5, 6, 7, 12 ( $P_4$  1011) have even parity. So  $P_4 = 1$ .

$P_8$  is to be selected so that bits 8, 9, 10, 11, 12 ( $P_8$  1011) have even parity. So  $P_8 = 1$ .

Based on these, the 12-bit hamming code word is 000110111011.

(b) Out of the 12 bits in the Hamming code, 4 bits located in positions 1, 2, 4, 8 from left are parity bits. The remaining 8 bits are data bits. So the original 8 bit word is made of bits 3, 5, 6, 7, 9, 10, 11, 12 from left. The data word is as shown below.

	$P_1$	$P_2$	$D_3$	$P_4$	$D_5$	$D_6$	$D_7$	$P_8$	$D_9$	$D_{10}$	$D_{11}$	$D_{12}$	Data word
(i)	∅	∅	0	∅	1	1	1	∅	1	0	1	0	01111010
(ii)	✗	∅	1	✗	1	0	0	∅	0	1	1	0	11000110
(iii)	✗	∅	1	✗	1	1	1	✗	0	1	0	0	11110100

## 3.7 ALPHANUMERIC CODES

Alphanumeric codes are codes used to encode the characters of alphabet in addition to the decimal digits. They are used primarily for transmitting data between computers and its I/O devices such as printers, keyboards and video display terminals. Because the number of bits used in most alphanumeric codes is much more than those required to encode 10 decimal digits and 26 alphabetic characters, these codes include bit patterns for a wide range of other symbols and functions as well. The most popular modern alphanumeric codes are the ASCII code and the EBCDIC code.

### 3.7.1 The ASCII Code

The American Standard Code for Information Interchange (ASCII) pronounced as ‘ASKEE’ is a widely used alphanumeric code. This is basically a 7-bit code. Since the number of different bit patterns that can be created with 7 bits is  $2^7 = 128$ , the ASCII can be used to encode both the lowercase and uppercase characters of the alphabet (52 symbols) and some special symbols as well, in addition to the 10 decimal digits. It is used extensively for printers and terminals that interface with small computer systems. Many large systems also make provisions for its accommodation. Because characters are assigned in ascending binary numbers, ASCII is very easy for a computer to alphabetize and sort.

Table 3.12 shows the ASCII code groups. For ease of presentation, they are listed with the three most significant bits of each group along the top row and the four least significant bits along the left column.

**Table 3.12** The ASCII code

		MSBs							
		0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
LSBs	0 0 0 0	NUL	DEL	Space	0	@	P	p	
	0 0 0 1	SOH	DC1	!	1	A	Q	a	q
	0 0 1 0	STX	DC2	“	2	B	R	b	r
	0 0 1 1	ETX	DC3	#	3	C	S	c	s
	0 1 0 0	EOT	DC4	\$	4	D	T	d	t
	0 1 0 1	ENQ	NAK	%	5	E	U	e	u
	0 1 1 0	ACK	SYN	&	6	F	V	f	v
	0 1 1 1	BEL	ETB	‘	7	G	W	g	w
	1 0 0 0	BS	CAN	(	8	H	X	h	x
	1 0 0 1	HT	EM	)	9	I	Y	i	y
	1 0 1 0	LF	SUB	*	:	J	Z	j	z
	1 0 1 1	VT	ESC	+	;	K	[	k	{
	1 1 0 0	FF	FS	,	<	L	\	l	
	1 1 0 1	CR	GS	-	=	M	]	m	}
	1 1 1 0	SO	RS	.	>	N	^	n	~
	1 1 1 1	SI	US	/	?	O	_	o	DLE

### Abbreviations

ACK	Acknowledge	EM	End of medium	NAK	Negative acknowledge
BEL	Bell	ENQ	Enquiry	NUL	Null
BS	Backspace	EOT	End of transmission	RS	Record separator
CAN	Cancel	ESC	Escape	SI	Shift in
CR	Carriage return	ETB	End of transmission block	SO	Shift out
DC1	Direct control 1	ETX	End of text	SOH	Start of heading
DC2	Direct control 2	FF	Form feed	STX	Start text
DC3	Direct control 3	FS	Form separator	SUB	Substitute
DC4	Direct control 4	GS	Group separator	SYN	Synchronous idle
DEL	Delete idle	HT	Horizontal tab	US	Unit separator
DLE	Data link escape	LF	Line feed	VT	Vertical tab

### 3.7.2 The EBCDIC Code

The Extended Binary Coded Decimal Interchange Code (EBCDIC) pronounced as ‘eb-si-dik’ is an 8-bit alphanumeric code. Since  $2^8$  (= 256) bit patterns can be formed with 8-bits, the EBCDIC code can be used to encode all the symbols and control characters found in ASCII. It encodes many other symbols too. In fact, many of the bit patterns in the EBCDIC code are unassigned. It is used by most large computers to communicate in alphanumeric data. Unlike ASCII, which uses a

straight binary sequence for representing characters, this code uses BCD as the basis of binary assignment. Table 3.13 shows the EBCDIC code.

**Table 3.13** The EBCDIC code

MSD (Hex)																
LSD (Hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	DS		SP	&							[	]	\	0
1	SOH	DC1	SOS			/		a	j	~		A	J			1
2	STX	DC2	FS	SYN				b	k	s		B	K	S	2	
3	ETX	DC3					c	l	t		C	L	T	3		
4	PF	RES	BYP	PN			d	m	u		D	M	U	4		
5	HT	NL	LF	RS			e	n	v		E	N	V	5		
6	LC	BS	EOB	YC			f	o	w		F	O	W	6		
7	DEL	IL	PRE	EOT			g	p	x		G	P	X	7		
8		CAN					h	q	y		H	Q	Y	8		
9		EM					i	r	z		I	R	Z	9		
A	SMM	CC	SM		Ø	!	I	:								
B	VT				.	\$	,	#								
C	FF	IFS		DC4	<	*	%	@								
D	CR	IGS	ENQ	NAK	(	)	-	'								
E	SO	IRS	ACK		+	;	>	=								
F	SI	IUS	BEL	SUB	I	,	?	,								

### SHORT QUESTIONS AND ANSWERS

- How are binary codes classified?  
A. Binary codes are classified as numeric codes and alphanumeric codes.
- What do you mean by alphanumeric codes?  
A. Alphanumeric codes are codes which represent alphanumeric information, i.e. the letters of the alphabet and decimal numbers as sequence of 0s and 1s.
- What do you mean by numeric codes?  
A. Numeric codes are codes which represent numeric information, i.e. only numbers as a series of 0s and 1s.
- What do you mean by straight binary coding?  
A. Straight binary coding means writing the numbers in binary number system.
- What do you mean by BCD codes?  
A. Numeric codes used to represent decimal digits are called binary coded decimal (BCD) codes.

6. Why are BCD codes required?

A. We are very comfortable with the decimal number system, but digital systems force us to use the binary system. Although the binary number system has many practical advantages and is widely used in digital computers, in many cases it is very convenient to work with decimal numbers especially when communication between man and machine is extensive. Since most of the numerical data generated by man are in decimal numbers, to simplify the communication process between man and machine BCD codes are used.

7. What do you mean by coding?

A. The representation of decimal digits and letters by a sequence of 0s and 1s is called coding.

8. What do you mean by a code word?

A. A sequence of binary digits which represents a decimal digit is called a code word.

9. What are the two basic types of BCD codes?

A. The two basic types of BCD codes are (a) Weighted codes and (b) Non-weighted codes.

10. What do you mean by weighted codes? Give three examples.

A. Weighted codes are the codes which obey the position-weighting principle. 8421, 2421, 84-2-1 codes are weighted codes.

11. What are the two types of weighted codes?

A. The two types of weighted codes are (a) positively-weighted codes and (b) negatively-weighted codes.

12. What do you mean by positively-weighted codes? Give three examples.

A. Positively-weighted codes are the codes in which all the weights assigned to the binary digits are positive. 2421, 5211, 8421 codes are positively weighted codes.

13. What are the required positional-weights for a positively-weighted code?

A. In every positively-weighted code, the first weight must be 1, the second weight must be either 1 or 2 and the sum of all the weights must be equal to or greater than 9.

14. What do you mean by negatively-weighted codes? Give three examples.

A. Negatively-weighted codes are the codes in which some of the weights assigned to the binary digits are negative. 84-2-1, 74-2-1, 631-1 are negatively weighed codes.

15. What is the minimum number of bits required to encode the decimal digits 0 through 9? Justify your answer.

A. From 0 to 9 there are 10 bits. With an  $n$ -bit code, the maximum number of code words possible is  $2^n$ . For a 3-bit code only 8 distinct code words are possible and with a 4-bit code, 16 distinct code words are possible. Since 10-code words are required, a minimum of 4 bits are required to encode decimal digits 0 through 9.

16. What do you mean by non-weighted codes? Give examples.

A. Non-weighted codes are codes which do not obey the position-weighting principle. XS-3 code and 2-out-of-5 code are non-weighted codes.

17. What do you mean by a sequential code? Name two sequential codes.

A. A sequential code is one in which each succeeding code word is one binary number greater than its preceding code word. The 8421 and XS-3 codes are sequential codes.

18. Which codes facilitate mathematical manipulation of data?

A. Sequential codes facilitate mathematical manipulation of data.

- 19.** What do you mean by self-complementing codes? Give examples.
- A. A self-complementing code is one in which the code word for 9's complement of  $N$ , i.e. of  $9-N$  can be obtained from the code word of  $N$  by interchanging all the 0s and 1s. The 2421, 642-3, 84-2-1and XS-3 are self-complementing codes.
- 20.** For a code to be self-complementing what must be the sum of all its weights and why?
- A. For a code to be self-complementing, the sum of all its weights must be 9. This is because whatever may be the weights, 0 is to be represented by 0000 and since in a self-complementing code, the code for 9 is the complement of the code for 0, 9 has to be represented by 1111.
- 21.** Why is the self-complementing property of a code useful?
- A. In digital circuits, subtraction operation is performed using adder circuits. The 9's complement of a decimal digit is obtained by simply complementing each bit of a digit which makes it easier to perform subtraction operation.
- 22.** What do you mean by cyclic code? What is its other name? Give an example. Mention its applications.
- A. Cyclic codes are the codes in which each successive code word differs from the preceding one in only one bit position. They are also called unit distance codes. The Gray code is a cyclic code. It is often used for translating an analog quantity such as shaft position into a digital form.
- 23.** Which BCD code is called natural binary code and why?
- A. The 8421 BCD code is called the natural binary code. It is because the 8, 4, 2, and 1 weights attached to it represent natural binary weights.
- 24.** What are the advantages and disadvantages of the 8421 BCD code?
- A. The main advantage of the 8421 BCD code is its ease of conversion to and from decimal. It is a weighted code and is also sequential. Therefore, it is used for mathematical operations. The disadvantages are (a) it is less efficient than the pure binary in the sense that it requires more bits, (b) the arithmetic operations are more complex in BCD than they are in pure binary and (c) the rules of binary addition and subtraction do not apply to the 8421 number, but only to the individual 4-bit groups.
- 25.** How is BCD addition performed?
- A. The BCD addition is performed by individually adding the corresponding digits of the decimal numbers expressed in 4-bit binary groups starting from the LSD. If there is a carry out of one group to the next group, or if the result is an illegal code, then  $6_{10}$  (0110) is added to the sum term of that group and the resulting carry is added to the next group (this is done to skip the 6 illegal states).
- 26.** How is BCD subtraction performed?
- A. The BCD subtraction is performed by subtracting the digits of each 4-bit group of the subtrahend from the corresponding 4-bit group of the minuend in binary starting from the LSD. If there is a borrow from the next group, then  $6_{10}$  (0110) is subtracted from the difference term of this group (this is done to skip the 6 illegal states).
- 27.** How many illegal states does each one of the 4-bit BCD codes have?
- A. Each one of the 4-bit BCD codes has 6 illegal states.
- 28.** What do you mean by an invalid (illegal) state? Give examples.
- A. An invalid state is a state which does not actually exist. For example, to encode decimal digits 0-9, we require a 4-bit word. 4-bits can be combined in  $2^4 = 16$  possible ways. Out of these 16 states, 10 are used to represent the decimal digits. The remaining are invalid states. For 8421 BCD code 1010, 1011, 1100, 1101, 1110, and 1111 are invalid states.

- 29.** How XS-3 code can be used for mathematical operations even though it is a non-weighted code?
- A. XS-3 code is a sequential code. Hence, it can be used for mathematical operations even though it is a non-weighted code.
- 30.** What is an XS-3 code?
- A. An XS-3 code is a BCD code in which each binary code word is the corresponding 8421 code plus 0011.
- 31.** How is addition performed in XS-3 code?
- A. To add in XS-3, add the XS-3 numbers by adding the 4-bit groups in each column starting from the LSD. If there is no carry out from the addition of any of the 4-bit groups, subtract 0011 from the sum term of those groups (because when two decimal digits are added in XS-3 and there is no carry, the result is in excess-6). If there is a carry out, add 0011 to the second term of those groups (because when there is a carry, the 6 invalid states are skipped and the result is in normal binary).
- 32.** How is subtraction performed in XS-3?
- A. To subtract in XS-3, subtract the XS-3 numbers by subtracting each 4-bit group of the subtrahend from the corresponding 4-bit group of the minuend starting from the LSD. If there is no borrow from the next 4-bit group, add 0011 to the difference term of such groups (because when decimal digits are subtracted in XS-3 and there is no borrow, the result is in normal binary). If there is a borrow, subtract 0011 from the difference term (because taking a borrow is equivalent to adding 6 invalid states, so, the result is in excess-6).
- 33.** In practice how is subtraction performed in 8421 BCD and XS-3 codes?
- A. In practice subtraction is performed in 8421 BCD and XS-3 codes by the 9's complement method or the 10's complement method.
- 34.** What is a Gray code?
- A. A Gray code is a non-weighted, reflective, unit distance code. In fact it is the most popular of the unit distance codes.
- 35.** What is the minimum distance of BCD and XS-3 codes?
- A. The minimum distance of both BCD and XS-3 codes is one.
- 36.** Why is the Gray code not suitable for mathematical operations?
- A. The Gray code is not suitable for mathematical operations because it is not a sequential code.
- 37.** What do you mean by a reflective code?
- A. A reflective code is a code in which the  $n$  least significant bits for  $2^n$  through  $2^{n+1} - 1$  are the mirror images of those for 0 through  $2^n - 1$ .
- 38.** How do you obtain an  $N$ -bit Gray code from an  $N - 1$  bit Gray code?
- A. An  $N$ -bit Gray code can be obtained from an  $N - 1$  bit Gray code by reflecting the  $N - 1$  bit code about an axis at the end of the code and putting the MSB of 0 above the axis and the MSB of 1 below the axis.
- 39.** Is the Gray code a BCD code?
- A. No. The Gray code is not a BCD code.
- 40.** What is the reason for the popularity of the Gray code?
- A. One main reason for the popularity of the Gray code is its ease of conversion to and from binary.

## 116 FUNDAMENTALS OF DIGITAL CIRCUITS

**41.** What are the applications of Gray codes?

- A. Gray codes are used in instrumentation and data acquisition systems where linear or angular displacement is measured. They are also used in shaft encoders, I/O devices, A/D converters and other peripheral equipment.

**42.** How do you convert an  $n$ -bit binary number into a Gray code?

- A. Binary to Gray conversion can be performed in two ways.

*First method:* Record the MSB of the binary as the MSB of the Gray code. Add the MSB of the binary to the next bit in binary, recording the sum and ignoring the carry if any, i.e. X-OR the bits. This sum is the next bit of the Gray code. Add the 2nd bit of the binary to the 3rd bit of the binary, the 3rd bit to the 4th bit, and so on. Record the successive sums as the successive bits of the Gray code until all the bits of the binary number are exhausted.

*Second method:* X-OR (i.e. take the modulo sum of) the bits of the binary number with those of the binary number shifted one position to the right. The LSB of the shifted number is discarded and the MSB of the Gray code number is the same as the MSB of the original binary number.

**43.** How do you convert a Gray number to binary?

- A. The Gray to binary conversion is as follows:

1. The MSB of the binary number is the same as the MSB of the Gray code; record it.
2. Add the MSB of the binary to the next significant bit of the Gray code, i.e. X-OR them; record the sum and ignore the carry.
3. Add the 2nd bit of the binary to the 3rd bit of the Gray; the 3rd bit of the binary to the 4th bit of the Gray code and so on, each time recording the sum and ignoring the carry.
4. Continue this till all the Gray bits are exhausted. The sequence of bits that has been written down is the binary equivalent of the Gray code number.

**44.** What is an XS-3 Gray code?

- A. An XS-3 Gray code is a code in which each decimal digit is encoded with the Gray code pattern of the decimal digit that is greater by 3. It has a unit distance between the patterns for 0 and 9.

**45.** When do you say that a code is error detecting?

- A. A code is said to be error detecting, if it possesses the property such that the occurrence of a single bit error transforms a valid code word into an invalid one.

**46.** What is the simplest technique for detecting errors?

- A. The simplest technique for detecting errors is that of adding an extra bit known as the parity bit to each word being transmitted.

**47.** What is a parity bit?

- A. The parity bit is an extra bit added to each word being transmitted.

**48.** How many types of parity are there? Name them.

- A. There are two types of parity. They are (a) odd parity and (b) even parity.

**49.** What do you mean by odd parity?

- A. Odd parity means setting the parity bit to a 0 or a 1 at the transmitter such that the total number of 1 bits in the word including the parity bit is an odd number.

**50.** What do you mean by even parity?

- A. Even parity means setting the parity bit to a 0 or a 1 at the transmitter such that the total number of 1 bits in the word including the parity bit is an even number.

**51.** Why is odd parity used more often than even parity?

- A. Odd parity is used more often than even parity because even parity does not detect the situation where all 0s are created by a short circuit or some other fault condition.

- 52.** What do you mean by 'distance' between two words?
- A. The distance between two words is defined as the number of digits that must change in a word so that the other word results.
- 53.** What must be the minimum distance for a code to be an error detecting code?
- A. For a code to be an error detecting code, the minimum distance between two code words must be two or three.
- 54.** Why are check sums used?
- A. Simple parity cannot detect two errors in the same word. Check sums provide a sort of two-dimensional parity to overcome this difficulty.
- 55.** What is the use of block parity?
- A. Block parity can correct any single error in a data word and can detect any two errors in a data word.
- 56.** What is the other name of shift-counter code?
- A. A shift-counter code is also called the Johnson code.
- 57.** What are the merits and demerits of the ring-counter code?
- A. The ring-counter code has excellent error-detecting properties and is easier to implement but this code is inefficient.
- 58.** What do you mean by an error-correcting code?
- A. A code is said to be an error-correcting code if the correct code word can always be deduced from an erroneous word.
- 59.** What must be the minimum distance of a code for it to be a single-bit error-correcting code?
- A. For a code to be a single-bit error-correcting code, the minimum distance of that code must be three or more. It can also detect (but cannot correct) two-bit errors.
- 60.** What is the minimum distance of a 7-bit Hamming code?
- A. The minimum distance of a 7-bit Hamming code is three or more.
- 61.** What is the merit of Hamming code?
- A. Hamming codes can, not only detect errors but also correct them.
- 62.** How are Hamming code words formed?
- A. In the Hamming code, to each group of  $m$  data bits,  $k$  parity bits located at positions  $2^{k-1}$  are added to form an  $(m + k)$ -bit code word.
- 63.** What are alphanumeric codes? For what they are used?
- A. Alphanumeric codes are codes used to encode the characters of alphabet in addition to the decimal digits. They are used primarily for transmitting data between computers and its I/O devices such as printers, key boards, and video display terminals.
- 64.** What are the two popular modern alphanumeric codes?
- A. The two most popular modern alphanumeric codes are the ASCII code (American Standard Code for Information Interchange code) and the EBCDIC code (Extended Binary Coded Decimal Interchange Code).
- 65.** Why is ASCII code a 7-bit code?
- A. ASCII code is used to represent numerals 0 through 9, letters A through Z (lower case and upper case), special characters and symbols. The total number of items to be coded is 128, which needs a minimum of 7 bits.
- 66.** A line printer is capable of printing 132 characters on a single line, and each character is represented by the ASCII code. How many bits are required to print each line?
- A.  $132 \times 7 = 924$  bits.

## 118 FUNDAMENTALS OF DIGITAL CIRCUITS

67. How many bits are required to be reserved for storing 100 numbers of a group of people, assuming that no name occupies more than 20 characters (including space). Assume ASCII code.  
A.  $100 \times 20 \times 7 = 14,000$  bits.
68. What is the number of bits used in ASCII and EBCDIC codes?  
A. ASCII is a 7-bit code and EBCDIC is an 8-bit code. ASCII uses a straight binary sequence for representing characters but EBCDIC uses BCD as the basis of binary assignment.

### REVIEW QUESTIONS

1. Write 2421, 5211, 5421, 642-3, 84-2-1 and XS-3 codes.
2. Write two sequential codes.
3. Obtain 3-bit and 4-bit Gray codes from a 2-bit Gray code by reflection.
4. Write notes on XS-3 code.
5. Write notes on Gray code.
6. Explain how errors are detected using (a) check sums, and (b) block parity.
7. Write notes on error correcting codes.
8. What is the Hamming code? How is the Hamming code word tested and corrected?

### FILL IN THE BLANKS

1. Binary codes are classified as \_\_\_\_\_ codes and \_\_\_\_\_ codes.
2. \_\_\_\_\_ are codes which represent letters of the alphabets and decimal numbers as a sequence of 0s and 1s.
3. \_\_\_\_\_ are codes which represent numeric information, i.e. only numbers as a series of 0s and 1s.
4. Numeric codes used to represent decimal digits are called \_\_\_\_\_ codes.
5. A sequence of binary digits which represents a decimal digit is called a \_\_\_\_\_.
6. The two basic types of BCD codes are (a) \_\_\_\_\_ and (b) \_\_\_\_\_.
7. \_\_\_\_\_ are the codes which obey the position weighting principle.
8. The two types of weighted codes are (a) \_\_\_\_\_ and (b) \_\_\_\_\_.
9. \_\_\_\_\_ are the codes in which some of the weights assigned to the binary digits are negative.
10. \_\_\_\_\_ coding means writing the decimal numbers in binary number system.
11. \_\_\_\_\_ are the codes which do not obey the position-weighting principle.
12. The representation of decimal digits and letters by a sequence of 0s and 1s is called \_\_\_\_\_.
13. A \_\_\_\_\_ code is one in which each succeeding code word is one binary number greater than its preceding code word.
14. \_\_\_\_\_ codes facilitate mathematical manipulation of data.
15. A \_\_\_\_\_ code is one in which the code word of the 9's complement of  $N$ , i.e. of  $9 - N$  can be obtained from the code word of  $N$  by interchanging all the 0s and 1s.
16. For a code to be self-complementing the sum of its weights must be \_\_\_\_\_.

17. \_\_\_\_\_ are the codes in which each successive code word differs from the preceding one in only one bit position.
18. Cyclic codes are also called \_\_\_\_\_ codes.
19. \_\_\_\_\_ code is a natural binary code.
20. Each one of the 4-bit BCD codes have \_\_\_\_\_ illegal states.
21. XS-3 code is a \_\_\_\_\_ code.
22. \_\_\_\_\_ is a reflective code.
23. A \_\_\_\_\_ is a code in which the  $n$  least significant bits for  $2^n$  through  $2^{n+1} - 1$  are the mirror images of those for 0 through  $2^n - 1$ .
24. The \_\_\_\_\_ is an extra bit added to each word being transmitted.
25. The two types of parity are (a) \_\_\_\_\_ and (b) \_\_\_\_\_.
26. \_\_\_\_\_ parity is used more often than \_\_\_\_\_ parity.
27. The \_\_\_\_\_ between two words is defined as the number of digits that must change in a word so that the other word results.
28. For a code to be an error detecting code, the minimum distance between two code words must be \_\_\_\_\_.
29. Simple parity cannot detect \_\_\_\_\_ errors in the same word.
30. For a code to be a single bit error correcting code, the minimum distance of that code must be \_\_\_\_\_.
31. \_\_\_\_\_ codes can, not only detect errors but also correct them.
32. The minimum distance required for a code is \_\_\_\_\_ for detecting double error.
33. The distance between code words 10010 and 10101 is \_\_\_\_\_.
34. A single parity bit attached to 8421 code makes its minimum distance \_\_\_\_\_.
35. A minimum of \_\_\_\_\_ parity bits are required for generating hamming code for single error correction in 8421 code.
36. The two popular alphanumeric codes are (a) \_\_\_\_\_ and (b) \_\_\_\_\_.
37. ASCII is a \_\_\_\_\_ digit code and EBCDIC is a \_\_\_\_\_ digit code.
38. In the Hamming code to each group of  $m$  data bits,  $k$  parity bits located at positions \_\_\_\_\_ are added to form an  $(m + k)$  bit code word.
39. The minimum distance of both BCD and XS-3 codes is \_\_\_\_\_.
40. Block parity can \_\_\_\_\_ any single error in a data word and \_\_\_\_\_ any two errors in a data word.

### OBJECTIVE TYPE QUESTIONS

1. The code used in digital systems to represent decimal digits, letters, and other special characters such as +, -, ., \*, etc. is
 

(a) hexadecimal	(b) octal	(c) natural BCD	(d) ASCII
-----------------	-----------	-----------------	-----------
2. The codes in which each successive code word differs from the preceding one in only one bit position are called
 

(a) BCD codes	(b) sequential codes
(c) self-complementing codes	(d) cyclic codes

## 120 FUNDAMENTALS OF DIGITAL CIRCUITS

3. Unit distance code is the other name of
  - (a) sequential code
  - (b) self-complementing code
  - (c) cyclic code
  - (d) XS-3 code
4. For mathematical operations, the code must be a
  - (a) sequential code
  - (b) cyclic code
  - (c) self-complementing code
  - (d) unit distance code
5.  $(11011)_2$  in BCD 8421 code is
  - (a) 00011011
  - (b) 00100111
  - (c) 11011001
  - (d) 01101100
6.  $(1D)_{16}$  in BCD 8421 code is
  - (a) 00011101
  - (b) 00101001
  - (c) 00011101
  - (d) 00100100
7.  $(44)_8$  in BCD 8421 code is
  - (a) 01000100
  - (b) 00100100
  - (c) 00110110
  - (d) 01100011
8. For a code to be self-complementing, the sum of all its weights must be
  - (a) 6
  - (b) 9
  - (c) 10
  - (d) 12
9. The following code is not a BCD code.
  - (a) Gray code
  - (b) XS-3 code
  - (c) 8421 code
  - (d) all of these
10. The parity of the binary number 11011001 is
  - (a) even
  - (b) not known
  - (c) odd
  - (d) same as the number of zeros
11. 2-out-of-5 code is
  - (a) weighted code
  - (b) self-complementing code
  - (c) non-weighted code
  - (d) alphanumeric code
12. 8421 code is
  - (a) self-complementing code
  - (b) weighted code
  - (c) non-weighted code
  - (d) alphanumeric code
13. 2421 code is
  - (a) weighted self-complementing code
  - (b) non-weighted self-complementing code
  - (c) weighted non-self-complementing code
  - (d) non-weighted and non-self-complementing code
14. The minimum distance of ASCII code is
  - (a) 1
  - (b) 2
  - (c) 3
  - (d) 4
15. A code with a minimum distance of 3 can
  - (a) detect double error and correct single error
  - (b) detect and correct double error
  - (c) detect single and correct double error
  - (d) not detect and correct any error
16. The minimum distance required for a code to detect and correct single bit error is
  - (a) 1
  - (b) 2
  - (c) 3
  - (d) 4
17. The minimum distance of a straight binary 4-bit code is
  - (a) 0
  - (b) 1
  - (c) 2
  - (d) 3



## PROBLEMS

- 3.18** (a) Generate a 4-bit Gray code directly using the mirror image property.  
(b) Generate a Hamming code for the given 11-bit message word 100, 0110, 0101 and rewrite the entire message in Hamming code.

**3.19** Generate the weighted codes for the decimal digits using the weights.  
(a) 3, 3, 2, 1  
(b) 4, 4, 3, -2

**3.20** Consider the following four codes.

<b>Code A</b>	<b>Code B</b>	<b>Code C</b>	<b>Code D</b>
0001	000	01011	000000
0010	001	01100	001111
0100	011	10010	110011
1000	010	10101	
	110		
	111		
	101		
	100		

Which of the following properties is satisfied by each of the above codes?

- (a) detects single errors
  - (b) detects double errors
  - (c) detects triple errors
  - (d) corrects single errors
  - (e) corrects double errors
  - (f) corrects single and detects double errors

**3.21** (a) Device a single error correcting code for a 11-bit group 010, 1101, 1010.  
(b) Test the following Hamming code sequence for 11-bit message and correct it if necessary:  
(100, 1101, 1110, 1011).

**3.22** (a) Given the 8-bit data word 10111001, generate the 12-bit composite word for the hamming code that corrects and detects single errors.  
(b) A 12-bit Hamming code word containing 8 bits of data and 4 parity bits is read from memory. What is the original 8-bit word if the 12-bit read out is as follows?
  - (i) 1100 1010 0110
  - (ii) 0011 0110 0101
  - (iii) 1010 1001 1101

**3.23** (a) Device a single error correcting code for a 8-bit group 0101 1011.  
(b) Test the following hamming code sequence for 8-bit message and correct it if necessary:  
(1011 0111 1110).

## VHDL PROGRAMS

### 1. VHDL PROGRAM FOR 7-BIT EVEN PARITY HAMMING CODE USING DATA FLOW MODELING

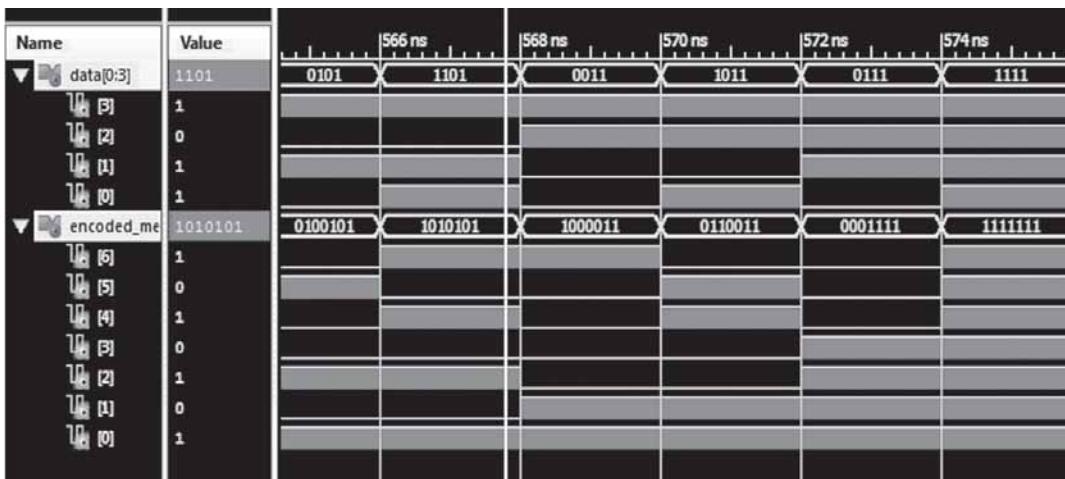
```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity SEVEN_BIT_HAMMING_CODE is
    Port ( DATA      : in STD_LOGIC_VECTOR (3 downto 0);
           -          D3      D5      D6      D7
           -          DATA (0) DATA (1) DATA (2) DATA (3)
           ENCODED_MESSAGE: out STD_LOGIC_VECTOR (6 downto 0));
end SEVEN_BIT_HAMMING_CODE;

architecture Behavioral of SEVEN_BIT_HAMMING_CODE is
signal P1,P2,P4 :STD_LOGIC;
begin
P1 <= (DATA(0) xor DATA(1) xor DATA(3));
P2 <= (DATA(0) xor DATA(2) xor DATA(3));
P4 <= (DATA(1) xor DATA(2) xor DATA(3));
ENCODED_MESSAGE <= (P1,P2,DATA(0),P4,DATA(1),DATA(2),DATA(3));
end Behavioral;

```

### SIMULATION OUTPUT:



## 2. VHDL PROGRAM FOR 12-BIT EVEN PARITY HAMMING CODE USING DATA FLOW MODELING

```

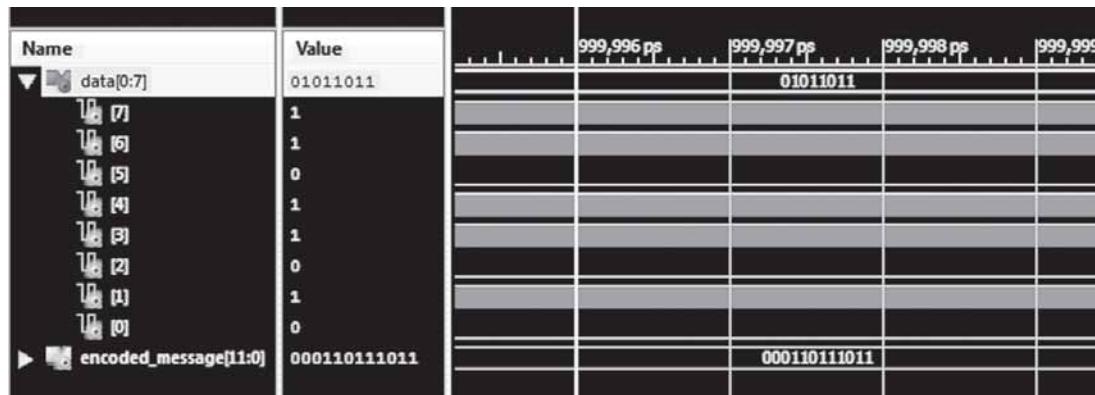
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity TWELVE_BIT_HAMMING_CODE is
    Port ( DATA      : in STD_LOGIC_VECTOR (7 downto 0);
           ENCODED_MESSAGE : out STD_LOGIC_VECTOR (11 downto 0));
end TWELVE_BIT_HAMMING_CODE;
    -      D3      D5      D6      D7      D9      D10     D11      D12
- DATA(0)  DATA(1)  DATA(2)  DATA(3)  DATA(4)  DATA(5)  DATA(6)  DATA(7)

architecture Behavioral of TWELVE_BIT_HAMMING_CODE is
signal P1,P2,P4,P8 :STD_LOGIC;
begin
P1 <= (DATA(0) xor DATA(1) xor DATA(3) xor DATA(4) xor DATA(6));
P2 <= (DATA(0) xor DATA(2) xor DATA(3) xor DATA(5) xor DATA(6));
P4 <= (DATA(1) xor DATA(2) xor DATA(3) xor DATA(7));
P8 <= (DATA(4) xor DATA(5) xor DATA(6) xor DATA(7));

ENCODED_MESSAGE <= (P1,P2,DATA(0),P4,DATA(1),DATA(2),DATA(3),P8,
                     DATA(4),DATA(5),DATA(6),DATA(7));
end Behavioral;

```

### SIMULATION OUTPUT:



## 3. VHDL PROGRAM FOR 15-BIT EVEN PARITY HAMMING CODE USING DATA FLOW MODELING

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

```

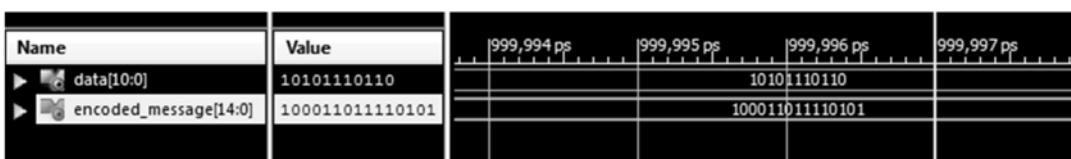
## 126 FUNDAMENTALS OF DIGITAL CIRCUITS

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity FIFTEEN_BIT_HAMMING_CODE is
    Port ( DATA      : in STD_LOGIC_VECTOR (10 downto 0);
    ENCODED_MESSAGE : out STD_LOGIC_VECTOR (14 downto 0));
end FIFTEEN_BIT_HAMMING_CODE;
    - D3      D5      D6      D7      D9      D10     D11      D12
- DATA(0)  DATA(1)  DATA(2)  DATA(3)  DATA(4)  DATA(5)  DATA(6)  DATA(7)
-           D13          D14          D15
- DATA(8)  DATA(9)  DATA(10)

architecture Behavioral of FIFTEEN_BIT_HAMMING_CODE is
signal P1,P2,P4,P8 :STD_LOGIC;
begin
P1 <= (DATA(0) xor DATA(1) xor DATA(3) xor DATA(4) xor DATA(6) xor
DATA(8) xor DATA(10));
P2 <= (DATA(0) xor DATA(2) xor DATA(3) xor DATA(5) xor DATA(6) xor
DATA(9) xor DATA(10));
P4 <= (DATA(1) xor DATA(2) xor DATA(3) xor DATA(7) xor DATA(8) xor
DATA(9) xor DATA(10));
P8 <= (DATA(4) xor DATA(5) xor DATA(6) xor DATA(7) xor DATA(8) xor
DATA(9) xor DATA(10));

ENCODED_MESSAGE <= (P1,P2,DATA(0),P4,DATA(1),DATA(2),DATA(3),P8,
DATA(4),DATA(5),DATA(6),DATA(7),DATA(8),  DATA(9),DATA(10));
end Behavioral;
```

### SIMULATION OUTPUT:



## 4. VHDL PROGRAM FOR BINARY-TO-GRAY CONVERSION USING DATA FLOW MODELING

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Binary_To_Gray is
    Port ( bin : in STD_LOGIC_VECTOR (3 downto 0);
           gray : out STD_LOGIC_VECTOR (3 downto 0));

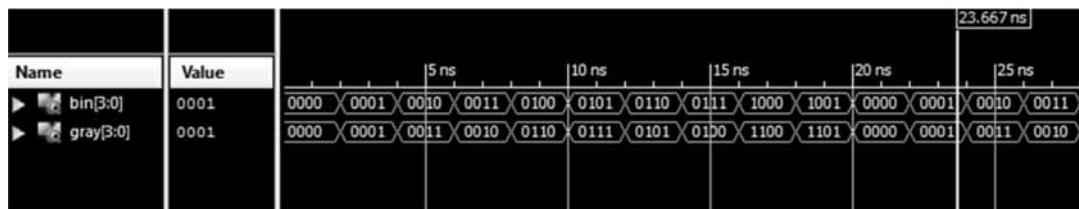
```

```

end Binary_To_Gray;
architecture Behavioral of Binary_To_Gray is
begin
gray(3) <= bin(3);
gray(2) <= bin(2) xor bin(3);
gray(1) <= bin(1) xor bin(2);
gray(0) <= bin(0) xor bin(1);
end Behavioral;

```

### SIMULATION OUTPUT:



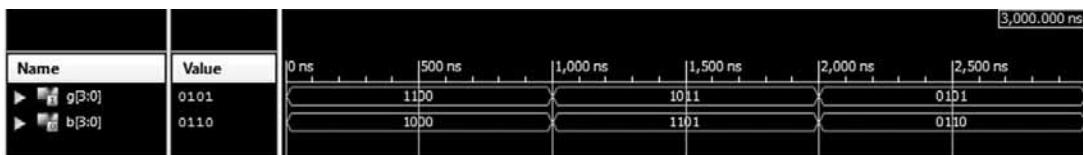
### 5. VHDL PROGRAM FOR GRAY-TO-BINARY CONVERSION USING DATA FLOW MODELING

```

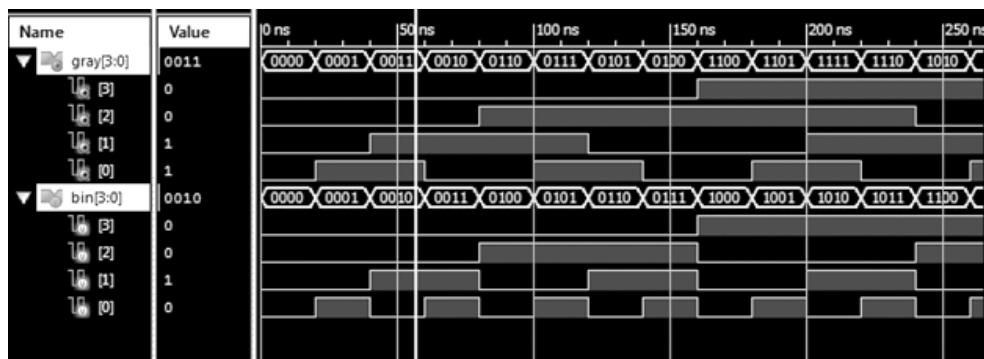
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity GRAY_BINARY is
    Port ( G: in STD_LOGIC_VECTOR(3 DOWNTO 0);
           B: out STD_LOGIC_VECTOR(3 DOWNTO 0));
end GRAY_BINARY;

architecture Behavioral of GRAY_BINARY is
begin
B(0)<= (((NOT G(3)) AND (NOT G(2)) AND (NOT G(1)) AND (NOT G(0)))
OR ((NOT G(3)) AND (NOT G(2))
AND G(1) AND (NOT G(0))) OR ((NOT G(3)) AND G(2) AND G(1) AND G(0))
OR ((NOT G(3)) AND G(2) AND (NOT G(1)) AND (NOT G(0))) OR (G(3) AND
G(2) AND (NOT G(1)) AND G(0)) OR (G(3) AND G(2) AND G(1) AND (NOT
G(0))) OR (G(3) AND (NOT G(2)) AND G(1) AND G(0)) OR (G(3)
AND (NOT G(2)) AND (NOT G(1)) AND (NOT G(0))));
B(1)<= G(3) XOR G(2) XOR G(1);
B(2)<= G(3) XOR G(2);
B(3)<= G(3);
end Behavioral;

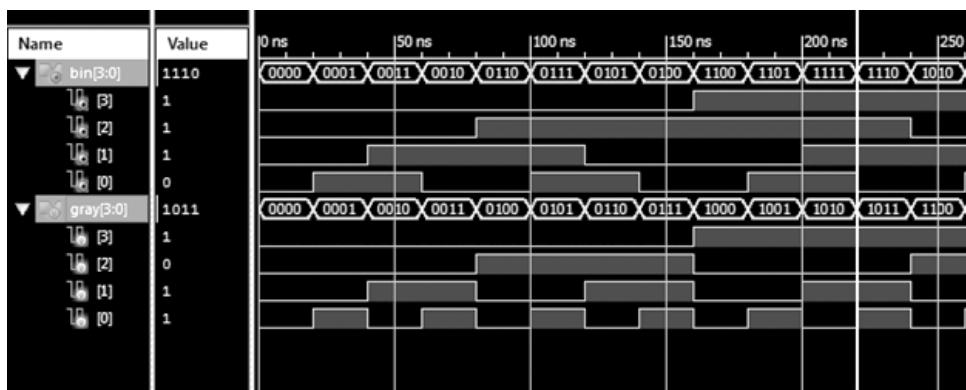
```

**SIMULATION OUTPUT:****VERILOG PROGRAMS****1. VERILOG PROGRAM FOR BINARY-TO-GRAY CONVERSION USING DATA FLOW MODELING**

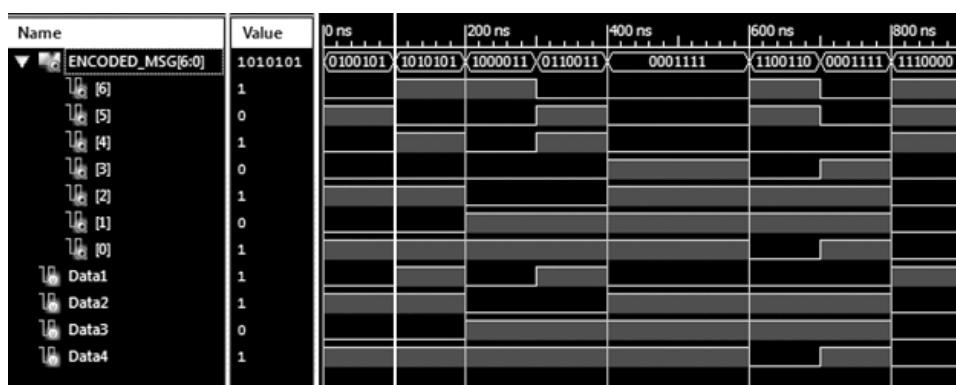
```
module binary_gray (bin,gray);
input [3:0] bin;
output [3:0] gray;
assign gray[3] = bin[3];
assign gray[2] = bin[2] ^ bin[3];
assign gray[1] = bin[1] ^ bin[2];
assign gray[0] = bin[0] ^ bin[1];
endmodule
```

**SIMULATION OUTPUT:****2. VERILOG PROGRAM FOR GRAY-TO-BINARY CONVERSION USING DATA FLOW MODELING**

```
module gray_binary(bin,gray);
input [3:0] gray;
output [3:0] bin ;
assign bin[3] = gray[3];
assign bin[2] = gray[2] ^ gray[3];
assign bin[1] = gray[1] ^ gray[2];
assign bin[0] = gray[0] ^ gray[1];
endmodule
```

**SIMULATION OUTPUT:****3. VERILOG PROGRAM FOR 7-BIT EVEN PARITY HAMMING CODE USING DATA FLOW MODELING**

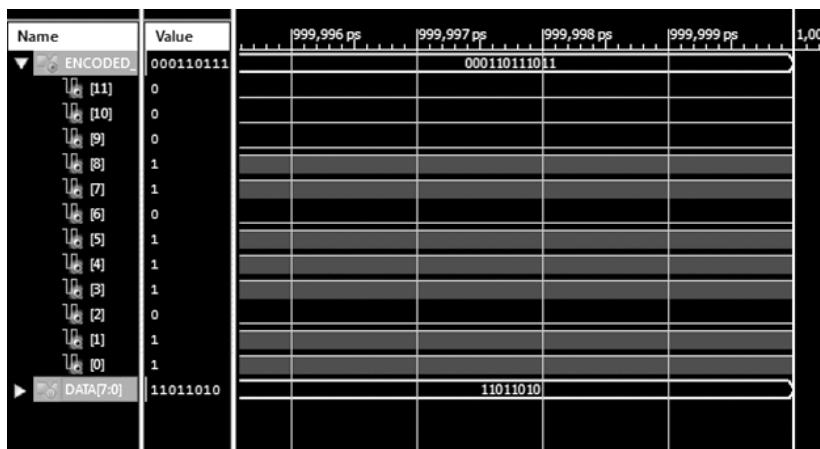
```
module Hamming_7_Code( input Data1,Data2,Data3,Data4,output [6:0] ENCODED_MSG);
    //           1      2      3      4      5      6      7
    //           P1,    P2,    D1,    P3,    D2,    D3,    D4
wire P1;
wire P2;
wire P3;
assign P1 = (Data1^Data2^Data4);
assign P2 = (Data1^Data3^Data4);
assign P3 = (Data2^Data3^Data4);
assign ENCODED_MSG = {P1,P2,Data1,P3,Data2,Data3,Data4};
endmodule
```

**SIMULATION OUTPUT:**

#### 4. VERILOG PROGRAM FOR 12-BIT EVEN PARITY HAMMING CODE USING DATA FLOW MODELING

```
module Hamming_12_Code( input [7:0] DATA, output [11:0]
ENCODED_MSG);
wire P1,P2,P4,P8;
assign P1 = (DATA[0]^DATA[1]^DATA[3]^DATA[4]^DATA[6]);
assign P2 = (DATA[0]^DATA[2]^DATA[3]^DATA[5]^DATA[6]);
assign P4 = (DATA[1]^DATA[2]^DATA[3]^DATA[7]);
assign P8 = (DATA[4]^DATA[5]^DATA[6]^DATA[7]);
assign ENCODED_MSG = {P1,P2,DATA[0],P4,DATA[1],DATA[2],DATA[3],P8,DATA[4],DATA[5],DATA[6],DATA[7]};
endmodule
```

#### SIMULATION OUTPUT:

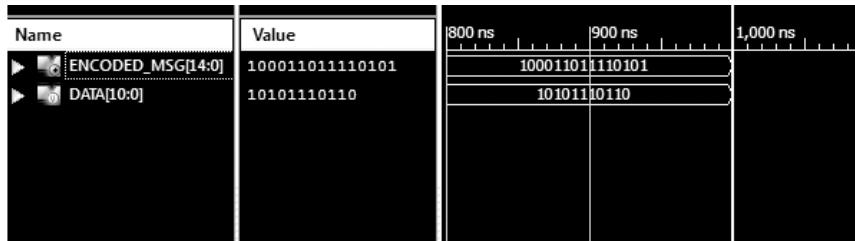


#### 5. VERILOG PROGRAM FOR 15-BIT EVEN PARITY HAMMING CODE USING DATA FLOW MODELING

```
module Hamming_15_Code( input [10:0] DATA, output [14:0]
ENCODED_MSG);
wire P1,P2,P4,P8;
assign P1= (DATA[0]^DATA[1]^
DATA[3]^DATA[4]^DATA[6]^DATA[8]^DATA[10]);
assign P2 = (DATA[0]^DATA[2]^
DATA[3]^DATA[5]^DATA[6]^DATA[9]^DATA[10]);
assign P4 = (DATA[1]^DATA[2]^
DATA[3]^DATA[7]^DATA[8]^DATA[9]^DATA[10]);
assign P8 = (DATA[4]^DATA[5]^
```

```
DATA [6] ^DATA [7] ^DATA [8] ^DATA [9] ^DATA [10]) ;  
assign ENCODED_MSG =  
{P1,P2,DATA[0],P4,DATA[1],DATA[2],DATA[3],P8,DATA[4],  
DATA[5],DATA[6],DATA[7],DATA[8],DATA[9],DATA[10]};  
endmodule
```

### SIMULATION OUTPUT:



# 4

## LOGIC GATES

### 4.1 INTRODUCTION

Logic gates are the fundamental building blocks of digital systems. The name logic gate is derived from the ability of such a device to make decisions, in the sense that it produces one output level when some combinations of input levels are present, and a different output level when other combinations of input levels are present. There are just three basic types of gates—AND, OR and NOT. The fact that computers are able to perform very complex logic operations, stems from the way these elementary gates are interconnected. The interconnection of gates to perform a variety of logical operations is called *logic design*.

Logic gates are electronic circuits because they are made up of a number of electronic devices and components. They are constructed in a wide variety of forms. They are usually embedded in large-scale integrated circuits (LSI) and very large-scale integrated circuits (VLSI) along with a large number of other devices, and are not easily accessible or identifiable. Each gate is dedicated to a specific logic operation. Logic gates are also constructed in small-scale integrated circuits (SSI), where they appear with few others of the same type. In these integrated devices, the inputs and outputs of all the gates are accessible, that is, external connections can be made to them just like discrete logic gates.

Inputs and outputs of logic gates can occur only in two levels. These two levels are termed HIGH and LOW, or TRUE and FALSE, or ON and OFF, or simply 1 and 0.

A table which lists all the possible combinations of input variables and the corresponding outputs is called a *truth table*. It shows how the logic circuit's output responds to various combinations of logic levels at the inputs.

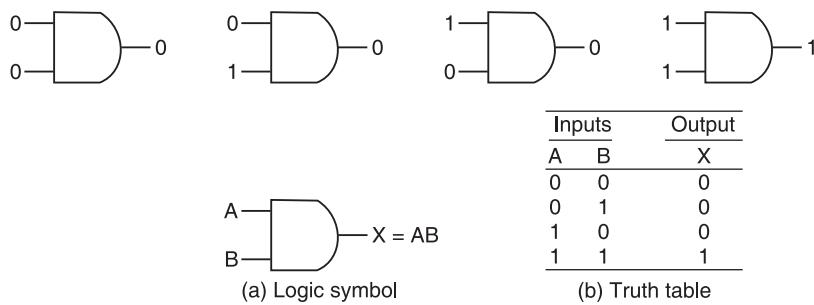
In this book, we use *level logic*, a logic in which the voltage levels represent logic 1 and logic 0. Level logic may be *positive logic* or *negative logic*. A positive logic system is the one in

which the higher of the two voltage levels represents the logic 1 and the lower of the two voltage levels represents the logic 0. A negative logic system is the one in which the lower of the two voltage levels represents the logic 1 and the higher of the two voltage levels represents the logic 0. In transistor-transistor logic (TTL, the most widely used logic family), the voltage levels are + 5 V and 0 V. In the following discussion in this chapter, logic 1 corresponds to + 5 V and logic 0 to 0 V.

## 4.2 THE AND GATE

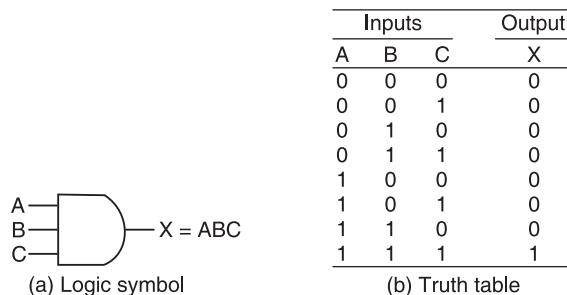
An AND gate has two or more inputs but only one output. The output assumes the logic 1 state, only when each one of its inputs is at logic 1 state. The output assumes the logic 0 state even if one of its inputs is at logic 0 state. The AND gate may, therefore, be defined as a device whose output is 1, if and only if all its inputs are 1. Hence the AND gate is also called an *all or nothing* gate.

The logic symbol and the truth table of a two-input AND gate are shown in Figure 4.1. Note that the output is 1 only when all the inputs are 1. The symbol for the AND operation is ‘·’, or we use no symbol at all.



**Figure 4.1** A two-input AND gate.

The logic symbol and the truth table of a three-input AND gate are shown in Figure 4.2.



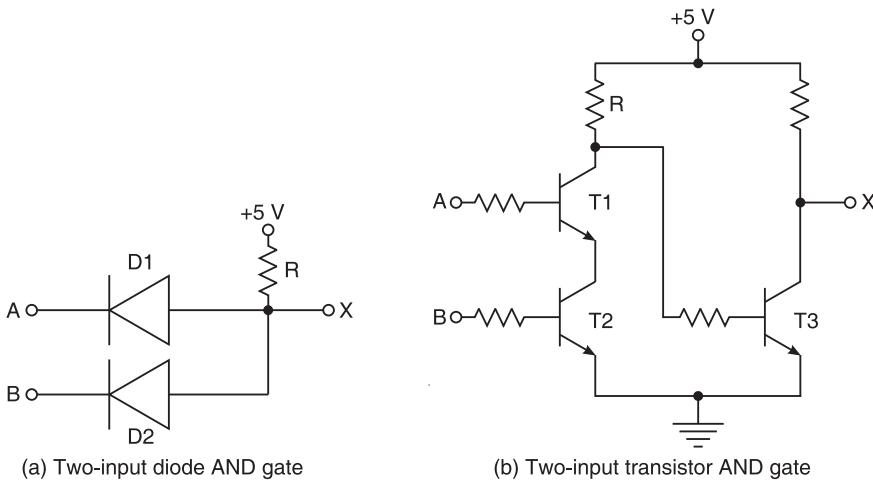
**Figure 4.2** A three-input AND gate.

With the input variables to the AND gate represented by A, B, C, . . . , the Boolean expression for the output can be written as  $X = A \cdot B \cdot C \dots$ , which is read as ‘X is equal to A and B and C . . .’ or ‘X is equal to ABC . . .’, or ‘X is equal to A dot B dot C . . .’.

#### 4.2.1 Realization of AND Gate (DL AND Gate and RTL AND Gate)

Discrete AND gates may be realized by using diodes (Diode logic) or transistors (Resistor transistor logic) as shown in Figures 4.3a and 4.3b respectively. The inputs A and B to the gates may be either 0V or +5V.

In the diode AND gate, when  $A = +5\text{ V}$  and  $B = +5\text{ V}$ , both the diodes D1 and D2 are OFF. So, no current flows through R and, therefore, no voltage drop occurs across R. Hence, the output  $X \approx 5\text{ V}$ . When  $A = 0\text{ V}$  or  $B = 0\text{ V}$  or when both A and B are equal to 0V, the corresponding diode D1 or D2 is ON or both diodes are ON and act as short-circuits (ideal case), and, therefore, the output  $X \approx 0\text{ V}$ . In practical circuits,  $X = 0.6\text{ V}$  or  $0.7\text{ V}$  which is treated as logic 0.



**Figure 4.3** Discrete AND gates.

In the transistor AND gate, when  $A = 0\text{ V}$  and  $B = 0\text{ V}$  or when  $A = 0\text{ V}$  and  $B = +5\text{ V}$  or when  $A = +5\text{ V}$  and  $B = 0\text{ V}$ , both the transistors T1 and T2 are OFF. Transistor T3 gets enough base drive from the supply through R and so, T3 will be ON. Hence, the output voltage  $X = V_{ce(sat)} \approx 0\text{ V}$ . When both A and B are equal to +5V, both the transistors T1 and T2 will be ON and, therefore, the voltage at the collector of transistor T1 will drop. So, T3 does not get enough base drive and, therefore, remains OFF. Hence no current flows through the collector resistor of T3 and, therefore, no voltage drop occurs across it. Hence output voltage,  $X \approx 5\text{ V}$ . The truth table for the above gate circuits is as shown below.

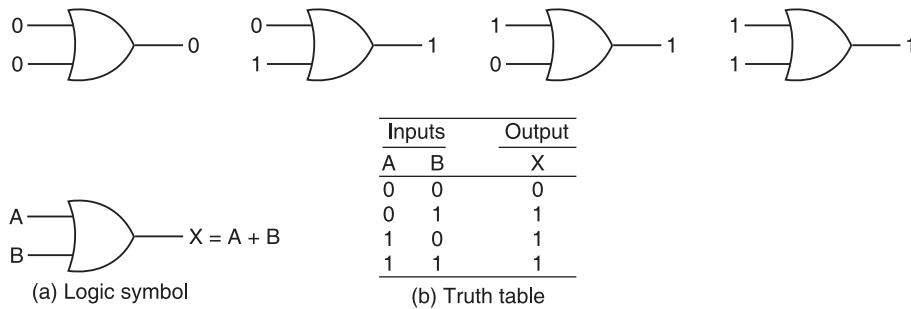
Truth table		
Inputs		Output
A	B	X
0 V	0 V	0 V
0 V	5 V	0 V
5 V	0 V	0 V
5 V	5 V	5 V

The IC 7408 contains four two-input AND gates, the IC 7411 contains three three-input AND gates, and the IC 7421 contains two four-input AND gates.

### 4.3 THE OR GATE

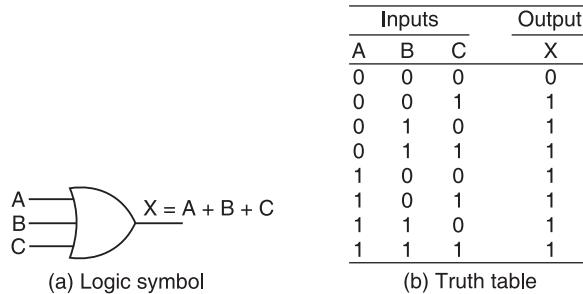
Like an AND gate, an OR gate may have two or more inputs but only one output. The output assumes the logic 1 state, even if one of its inputs is in logic 1 state. Its output assumes the logic 0 state, only when each one of its inputs is in logic 0 state. An OR gate may, therefore, be defined as a device whose output is 1, even if one of its inputs is 1. Hence an OR gate is also called an *any* or *all* gate. It can also be called an inclusive OR gate because it includes the condition ‘both the inputs can be present’.

The logic symbol and the truth table of a two-input OR gate are shown in Figure 4.4. Note that the output is 1 even if one of the inputs is 1.



**Figure 4.4** A two-input OR gate.

The logic symbol and the truth table of a three-input OR gate are shown in Figure 4.5.

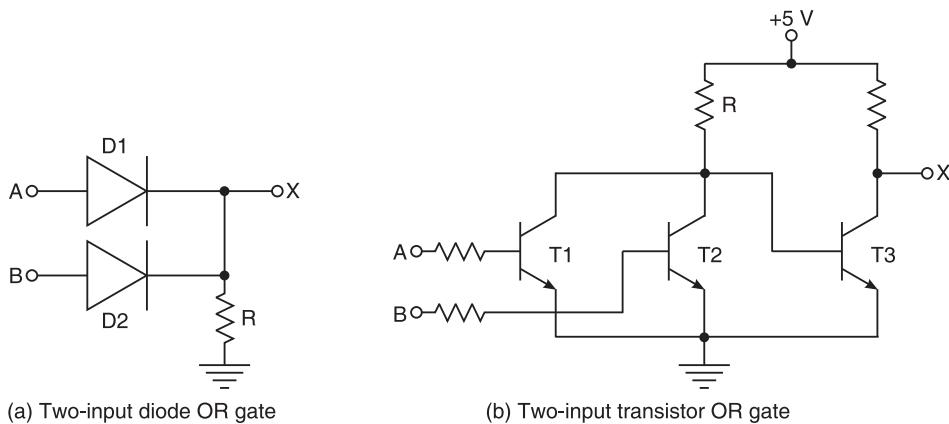


**Figure 4.5** A three-input OR gate.

The symbol for the OR operation is ‘+’. With the input variables to the OR gate represented by A, B, C . . . , the Boolean expression for the output can be written as  $X = A + B + C + \dots$ . This is read as ‘X is equal to A or B or C or . . .’, or ‘X is equal to A plus B plus C plus . . .’. Discrete OR gates may be realized by using diodes or transistors.

#### 4.3.1 Realization of OR Gate (DL OR Gate and RTL OR Gate)

Discrete OR gates may be realized by using diodes (Diode logic) or transistors (Resistor transistor logic) as shown in Figures 4.6a and 4.6b, respectively. The inputs A and B to the gates may be either 0 V or + 5 V.

**Figure 4.6** Discrete OR gates.

In the diode OR gate, when  $A = 0 \text{ V}$  and  $B = 0 \text{ V}$ , both the diodes D1 and D2 are OFF. No current flows through R, and so, no voltage drop occurs across R. Hence, the output voltage  $X = 0 \text{ V}$ . When either  $A = + 5 \text{ V}$  or  $B = + 5 \text{ V}$  or when both A and B are equal to  $+ 5 \text{ V}$ , the corresponding diode D1 or D2 is ON or both D1 and D2 are ON and act as short-circuits (ideal case) and, therefore, output  $X \approx 5 \text{ V}$ . In practice,  $X = + 5 \text{ V} - \text{diode drop} = + 5 \text{ V} - 0.7 \text{ V} = 4.3 \text{ V}$ , which is regarded as logic 1.

In the transistor OR gate, when  $A = 0 \text{ V}$  and  $B = 0 \text{ V}$ , both the transistors T1 and T2 are OFF. Transistor T3 gets enough base drive from  $+ 5 \text{ V}$  through R and, therefore, it will be ON. The output voltage,  $X = V_{ce(sat)} \approx 0 \text{ V}$ . When either  $A = + 5 \text{ V}$  or  $B = + 5 \text{ V}$  or when both A and B are equal to  $+ 5 \text{ V}$ , the corresponding transistor T1 or T2 is ON or both T1 and T2 will be ON and, therefore, the voltage at the collector of T1 is  $= V_{ce(sat)} \approx 0 \text{ V}$ . This cannot forward bias the base-emitter junction of T3 and, therefore, it will remain OFF. Hence, the output voltage will be  $X = 5 \text{ V}$  (logic 1 level).

The truth table for the above OR gate circuits is as shown below.

Truth table		
Inputs		Output
A	B	X
0 V	0 V	0 V
0 V	5 V	5 V
5 V	0 V	5 V
5 V	5 V	5 V

The IC 7432 contains four two-input OR gates.

#### 4.4 THE NOT GATE (INVERTER)

A NOT gate, also called an *inverter*, has only one input and, of course, only one output. It is a device whose output is always the complement of its input. That is, the output of a NOT gate assumes the logic 1 state when its input is in logic 0 state and assumes the logic 0 state when its input is in logic 1 state. The logic symbol and the truth table of an inverter are shown in Figures 4.7a and 4.7b, respectively.

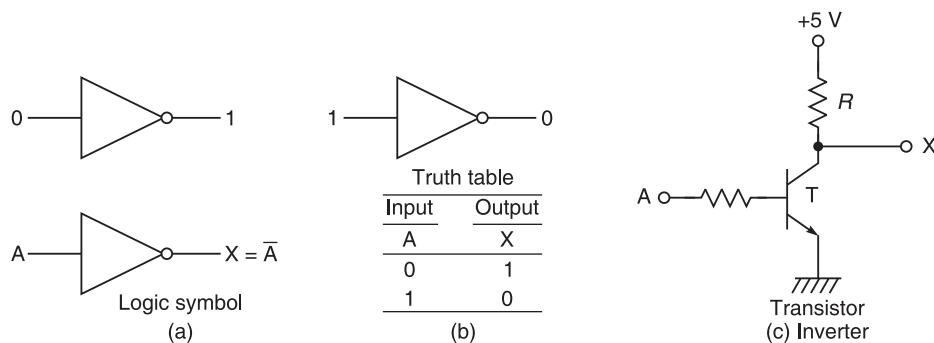


Figure 4.7 The inverter.

The symbol for NOT operation is ‘-’ (bar). When the input variable to the NOT gate is represented by A and the output variable by X, the expression for the output is  $X = \bar{A}$ . This is read as ‘X is equal to A bar’. A discrete NOT gate may be realized using a transistor. The IC 7404 contains six inverters.

#### 4.4.1 Realization of NOT Gate (RTL Logic)

A discrete NOT gate may be realized using a transistor (Resistor transistor logic) as shown in Figure 4.7c. The input to the gate may be 0 V or + 5 V. When  $A = 0$  V, the transistor T is OFF. As no current flows through R, no voltage drop occurs across R. Hence, the output voltage  $X = + 5$  V. When the input  $A = + 5$  V, T is ON and the output voltage  $X = V_{ce(sat)} \approx 0$  V. The truth table for the NOT gate circuit is as shown below.

The IC 7404 contains six inverters.

Truth table	
Input	Output
A	X
0 V	5 V
5 V	0 V

The IC 7404 contains six inverters.

Logic circuits of any complexity can be realized using only AND, OR and NOT gates. Logic circuits which use these three gates only are called AND/OR/INVERT, i.e. AOI logic circuits. Logic circuits which use AND gates and OR gates only are called AND/OR, i.e. AO logic circuits.

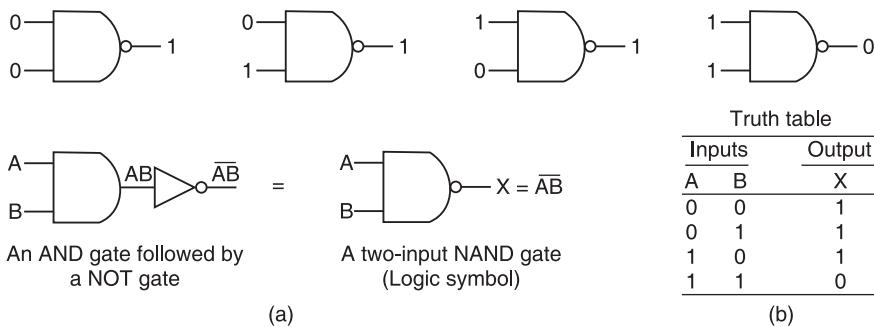
## 4.5 THE UNIVERSAL GATES

Though logic circuits of any complexity can be realized by using only the three basic gates (AND, OR and NOT), there are two universal gates (NAND and NOR), each of which can also realize logic circuits single-handedly. The NAND and NOR gates are therefore, called universal building blocks. Both NAND and NOR gates can perform all the three basic logic functions (AND, OR and NOT). Therefore, AOI logic can be converted to NAND logic or NOR logic.

### 4.5.1 The NAND Gate

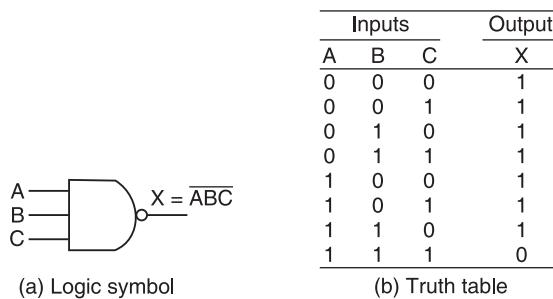
NAND means NOT AND, i.e. the AND output is NOTed. So, a NAND gate is a combination of an AND gate and a NOT gate (Figure 4.8a). In fact NAND is a contraction of the word NOT-AND.

The expression for the output of the NAND gate can, therefore, be written as  $X = \overline{ABC\dots}$  and is read as ‘X is equal to  $A \cdot B \cdot C \dots$  whole bar’. The output is logic 0 level, only when each of the inputs assumes a logic 1 level. For any other combination of inputs, the output is a logic 1 level. The logic symbol and the truth table of a two-input NAND gate are shown in Figures 4.8a and b.



**Figure 4.8** A two-input NAND gate.

The logic symbol and truth table for a three-input NAND gate are shown in Figures 4.9a and 4.9b, respectively.



**Figure 4.9** A three-input NAND gate.

**Bubbled OR gate:** Looking at the truth table of a two-input NAND gate, we see that the output X is 1 when either  $A = 0$  or  $B = 0$  or when both A and B are equal to 0, i.e. if either  $\overline{A} = 1$  or  $\overline{B} = 1$  or both  $\overline{A}$  and  $\overline{B}$  are equal to 1. Therefore, the NAND gate can perform the OR function. The corresponding output expression is,  $X = \overline{A} + \overline{B}$ . So, a NAND function can also be realized by first inverting the inputs and then ORing the inverted inputs. Thus, a NAND gate is a combination of two NOT gates and an OR gate (see Figure 4.10a). Hence, from Figures 4.8a and 4.10a, we can express the output of a two-input NAND gate as

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

The OR gate with inverted inputs (Figure 4.10a) is called a *bubbled* OR gate. So, a NAND gate is equivalent to a bubbled OR gate whose truth table is shown in Figure 4.10b. A bubbled OR

gate is also called a *negative OR* gate. Since its output assumes the HIGH state even if any one of its inputs is 0, the NAND gate is also called an active-LOW OR gate.

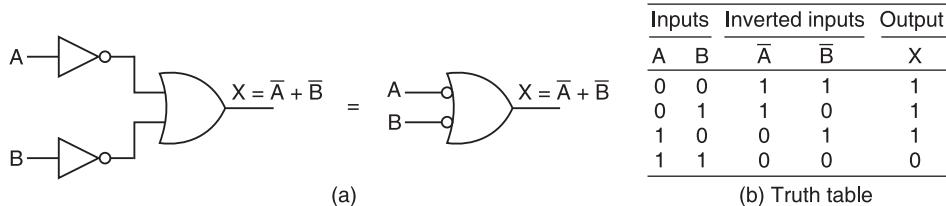


Figure 4.10 Bubbled OR gate.

**NAND gate as an inverter:** A NAND gate can also be used as an inverter by tying all its input terminals together and applying the signal to be inverted to the common terminal (Figure 4.11a), or by connecting all its input terminals except one, to logic 1 and applying the signal to be inverted to the remaining terminal as shown in Figure 4.11b. In the latter form, it is said to act as a controlled inverter.

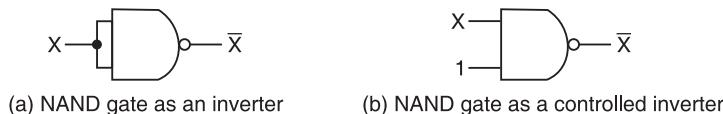


Figure 4.11 NAND gate as an inverter.

**Bubbled NAND gate:** The bubbled NAND gate is equivalent to OR gate as shown in Figure 4.12.

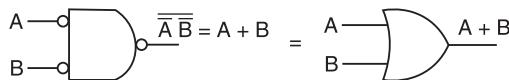


Figure 4.12 Bubbled NAND gate.

**Realization of NAND gate (DTL NAND gate):** A discrete two-input NAND gate (Diode transistor logic) is shown in Figure 4.13a. When  $A = +5\text{ V}$  and  $B = +5\text{ V}$ , both the diodes D1 and D2 are OFF. The transistor T gets enough base drive from the supply through  $R$  and, therefore, T is ON and the output  $X = V_{CE(sat)} \approx 0\text{ V}$ . When  $A = 0\text{ V}$  or  $B = 0\text{ V}$  or when both A and B are equal to  $0\text{ V}$ , the transistor T is OFF and, therefore, output  $\approx +5\text{ V}$ . The truth table is as shown in Figure 4.13b.

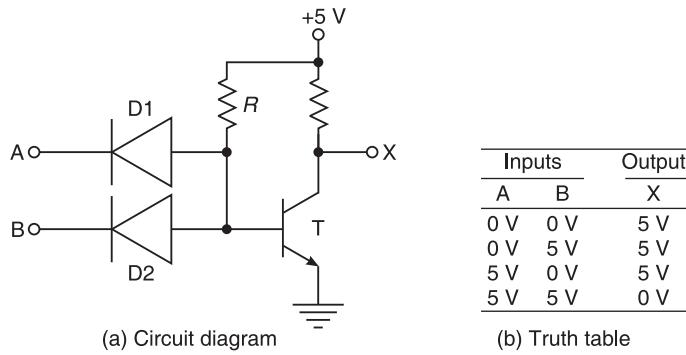
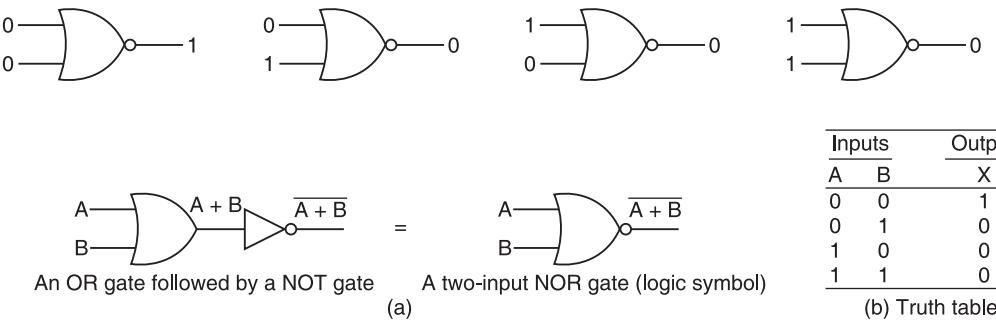


Figure 4.13 Discrete two-input NAND gate.

The IC 7400 contains four two-input NAND gates; the IC 7410 contains three three-input NAND gates; the IC 7420 contains two four-input NAND gates; and the IC 7430 contains one eight-input NAND gate.

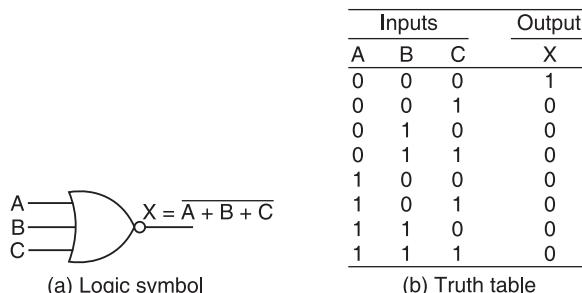
#### 4.5.2 The NOR Gate

NOR means NOT OR, i.e. the OR output is NOTed. So, a NOR gate is a combination of an OR gate and a NOT gate (Figure 4.14a). In fact NOR is a contraction of the word NOT-OR. The expression for the output of the NOR gate is,  $X = \overline{A + B + C + \dots}$  and is read as ‘X is equal to A plus B plus C plus ... whole bar’. The output is logic 1 level, only when each one of its inputs assumes a logic 0 level. For any other combination of inputs, the output is a logic 0 level. The logic symbol and the truth table of a two-input NOR gate are shown in Figures 4.14a and 4.14b respectively.



**Figure 4.14** A two-input NOR gate.

The logic symbol and truth table for a three-input NOR gate are shown in Figures 4.15a and 4.15b, respectively.



**Figure 4.15** A three-input NOR gate.

**Bubbled AND gate:** Looking at the truth table of a two-input NOR gate, we see that the output X is 1 only when both A and B are equal to 0, i.e. only when both  $\bar{A}$  and  $\bar{B}$  are equal to 1. That means, a NOR gate is equivalent to an AND gate with inverted inputs and the corresponding output expression is,  $X = \bar{A}\bar{B}$ . So, a NOR function can also be realized by first inverting the inputs and then ANDing those inverted inputs. Thus, a NOR gate is a combination of two NOT gates and an AND gate (see Figure 4.16a). Hence, from Figures 4.14a and 4.16a, we can see that the output of a two-input NOR gate is,  $X = \overline{A + B} = \bar{A}\bar{B}$ .

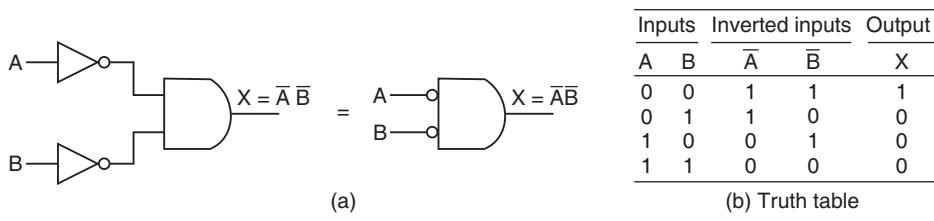


Figure 4.16 Bubbled AND gate.

The AND gate with inverted inputs (Figure 4.16a) is called a *bubbled* AND gate. So, a NOR gate is equivalent to a bubbled AND gate whose truth table is shown in Figure 4.16b. A bubbled AND gate is also called a *negative* AND gate. Since its output assumes the HIGH state only when all its inputs are in LOW state, a NOR gate is also called an active-LOW AND gate.

**NOR gate as an inverter:** A NOR gate can also be used as an inverter, by tying all its input terminals together and applying the signal to be inverted to the common terminal (Figure 4.17a) or by connecting all its input terminals except one to logic 0, and applying the signal to be inverted to the remaining terminal as shown in Figure 4.17b. In the latter form it is said to act as a controlled inverter.

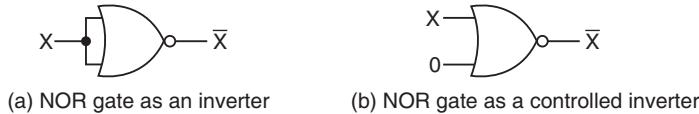


Figure 4.17 NOR gate as an inverter.

**Bubbled NOR gate:** The bubbled NOR gate is equivalent to an AND gate as shown in Figure 4.18.

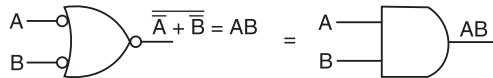


Figure 4.18 Bubbled NOR gate.

**Realization of NOR gate (RTL NOR gate):** A discrete two-input NOR gate (Resistor transistor logic) is shown in Figure 4.19a. When  $A = 0\text{ V}$  and  $B = 0\text{ V}$ , both transistors T1 and T2 are OFF; so, no current flows through  $R$  and, therefore, no voltage drop occurs across  $R$ . Hence, the output voltage  $X \approx +5\text{ V}$  (logic 1). When either  $A = +5\text{ V}$  or  $B = +5\text{ V}$  or when both  $A$  and  $B$  are equal to  $+5\text{ V}$ , the corresponding transistor T1 or T2 or both T1 and T2 are ON. Therefore,  $X$  is at  $V_{CE(sat)}$  with respect to ground and equal to  $0\text{ V}$  (logic 0). The truth table is shown in Figure 4.19b.

The IC 7402 contains four two-input NOR gates; the IC 7427 contains three three-input NOR gates; and the IC 7425 contains two four-input NOR gates.

The implication of the concept of universal gates is as follows. The cost of a large digital system such as a computer will be less if the variety of logic gates is reduced. Experience has shown that the cost of a digital system depends on the variety and not so much on the multiplicity of components. Most digital systems use only one type of gate—NAND or NOR to produce any Boolean function and consequently secure economic advantage by mass producing one universal gate.

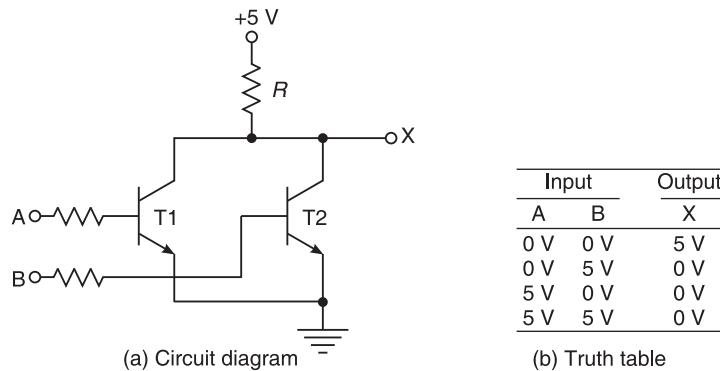


Figure 4.19 Discrete two-input NOR gate.

## 4.6 THE EXCLUSIVE-OR (X-OR) GATE

An X-OR gate is a two input, one output logic circuit, whose output assumes a logic 1 state when one and only one of its two inputs assumes a logic 1 state. Under the conditions when both the inputs assume the logic 0 state, or when both the inputs assume the logic 1 state, the output assumes a logic 0 state.

Since an X-OR gate produces an output 1 only when the inputs are not equal, it is called an *anti-coincidence* gate or *inequality detector*. The output of an X-OR gate is the modulo sum of its two inputs. The name Exclusive-OR is derived from the fact that its output is a 1, only when exclusively one of its inputs is a 1 (it excludes the condition when both the inputs are 1).

The logic symbol and truth table of a two-input X-OR gate are shown in Figures 4.20a and 4.20b, respectively. If the input variables are represented by A and B and the output variable by X, the expression for the output of this gate is written as  $X = A \oplus B = A\bar{B} + \bar{A}B$  and read as ‘X is equal to A ex-or B’.

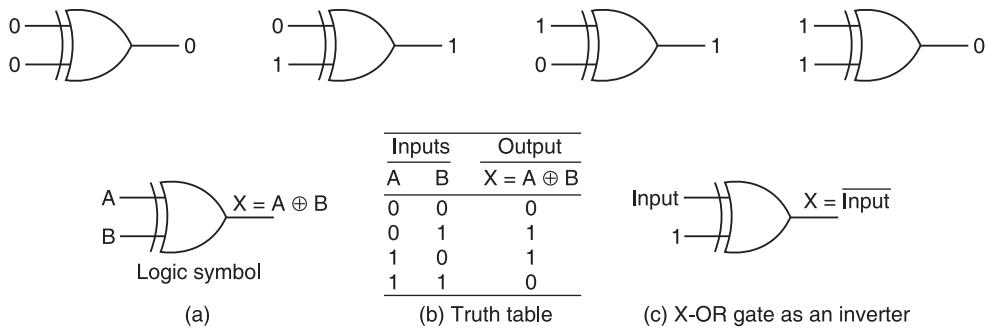


Figure 4.20 Exclusive-OR gate.

Three or more variable X-OR gates do not exist. When more than two variables are to be X-ORed, a number of two-input X-OR gates will be used. The X-OR of a number of variables assumes a 1 state only when an odd number of input variables assume a 1 state.

The Boolean expression whose value is equal to 1 only when an odd number of its variables are equal to 1 is called an odd function.

#### 4.6.1 X-OR Gate as an Inverter

An X-OR gate can be used as an inverter by connecting one of the two input terminals to logic 1 and feeding the input sequence to be inverted to the other terminal as shown in Figure 4.20c. If the input bit is a 0, the output is,  $0 \oplus 1 = 1$ , and if the input bit is a 1, the output is,  $1 \oplus 1 = 0$ . In fact, we can say that an X-OR gate can be used as a controlled inverter, that is, one of its inputs can be used to decide whether the signal at the other input will be inverted or not.

The TTL IC 7486 contains four X-OR gates. The CMOS IC 74C86 contains four X-OR gates. High speed CMOS IC 74HC86 contains four X-OR gates.

### 4.7 PROPERTIES OF EXCLUSIVE-OR

$$1. A \oplus 1 = \bar{A}$$

*Proof:*  $A \oplus 1 = A \cdot \bar{1} + \bar{A} \cdot 1 = A \cdot 0 + \bar{A} = \bar{A}$

$$2. A \oplus 0 = A$$

*Proof:*  $A \oplus 0 = A \cdot \bar{0} + \bar{A} \cdot 0 = A \cdot 1 + 0 = A$

$$3. A \oplus A = 0$$

*Proof:*  $A \oplus A = A \cdot \bar{A} + \bar{A} \cdot A = 0 + 0 = 0$

$$4. A \oplus \bar{A} = 1$$

*Proof:*  $A \oplus \bar{A} = A \cdot \bar{\bar{A}} + \bar{A} \cdot \bar{\bar{A}} = A + \bar{A} = 1$

$$5. AB \oplus AC = A(B \oplus C)$$

*Proof:*  $AB \oplus AC = AB \cdot \bar{AC} + \bar{AB} \cdot AC = AB(\bar{A} + \bar{C}) + (\bar{A} + \bar{B}) AC$   
 $= ABC + A\bar{B}C = A(\bar{B}\bar{C} + \bar{B}C) = A(B \oplus C)$

$$6. \text{If } A \oplus B = C, \text{ then } A \oplus C = B, B \oplus C = A, A \oplus B \oplus C = 0$$

*Proof:*  $A \oplus B \oplus C = (A \oplus B) \oplus (A \oplus B) = (A \oplus B) \cdot \bar{(A \oplus B)} + \bar{(A \oplus B)} \cdot (A \oplus B) = 0 + 0 = 0$

*Proof:*  $A \oplus C = A \oplus (A \oplus B) = A \cdot \bar{(A \oplus B)} + \bar{A} \cdot (A \oplus B) = A(AB + \bar{A}\bar{B}) + \bar{A}(A\bar{B} + \bar{A}B)$   
 $= AB + \bar{A}\bar{B} = B(A + \bar{A}) = B \cdot 1 = B$

*Proof:*  $B \oplus C = B \oplus (A \oplus B) = B \cdot \bar{A \oplus B} + \bar{B}(A \oplus B) = B \cdot (AB + \bar{A}\bar{B}) + \bar{B}(A\bar{B} + \bar{A}B)$   
 $= AB + A\bar{B} = A(B + \bar{B}) = A$

### 4.8 THE EXCLUSIVE-NOR (X-NOR) GATE

An X-NOR gate is a combination of an X-OR gate and a NOT gate. The X-NOR gate is a two input, one output logic circuit, whose output assumes a 1 state only when both the inputs assume a 0 state or when both the inputs assume a 1 state. The output assumes a 0 state, when one of the inputs assumes a 0 state and the other a 1 state. It is also called a *coincidence* gate, because its output is 1 only when its inputs coincide. It can be used as an *equality detector* because it outputs a 1 only when its inputs are equal.

The logic symbol and truth table of a two-input X-NOR gate are shown in Figures 4.21a and 4.21b, respectively. If the input variables are represented by A and B and the output variable by X, the expression for the output of this gate is written as

$$X = A \odot B = AB + \overline{A}\overline{B} = \overline{A} \oplus \overline{B} = \overline{AB} + \overline{\overline{AB}}$$

and read as ‘X is equal to A ex-nor B’.

Three or more variable X-NOR gates do not exist. When a number of variables are to be X-NORED, a number of two-input X-NOR gates can be used. The X-NOR of a number of variables assumes a 1 state, only when an even number (including zero) of input variables assume a 0 state.

The Boolean expression whose value is equal to 1 only when an even number of its variables are equal to 1 is called an even function.

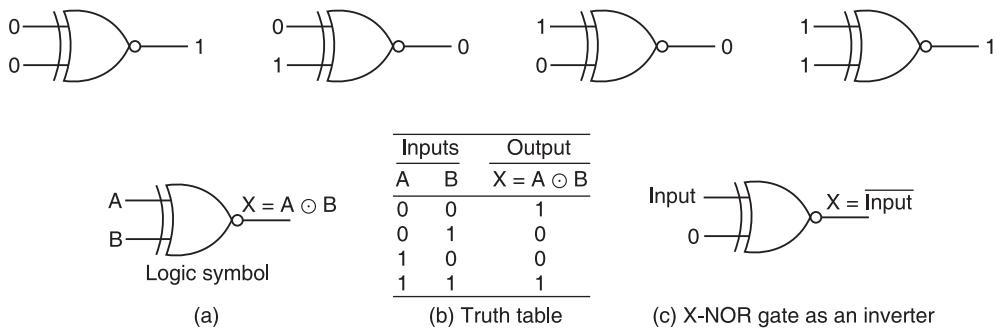


Figure 4.21 Exclusive-NOR gate.

#### 4.8.1 X-NOR Gate as an Inverter

An X-NOR gate can be used as an inverter by connecting one of the two input terminals to logic 0 and feeding the input sequence to be inverted to the other terminal as shown in Figure 4.21c. If the input bit is a 0, the output is,  $0 \odot 0 = 1$ , and if the input bit is a 1, the output  $1 \odot 0 = 0$ . In fact, we can say that an X-NOR gate can be used as a controlled inverter, that is, one of its inputs can be used to decide whether the signal at the other input will be inverted or not. The X-NOR gate can be used as a comparator too.

The X-NOR of two variables A and B is the complement of the X-OR of those two variables. That is,

$$A \odot B = \overline{A} \oplus B$$

But the X-NOR of three variables A, B and C is not equal to the complement of the X-OR of A, B, and C. That is,

$$A \odot B \odot C \neq \overline{A} \oplus B \oplus C$$

However, the X-NOR of a number of variables is equal to the complement of the X-OR of those variables only when the number of variables involved is even.

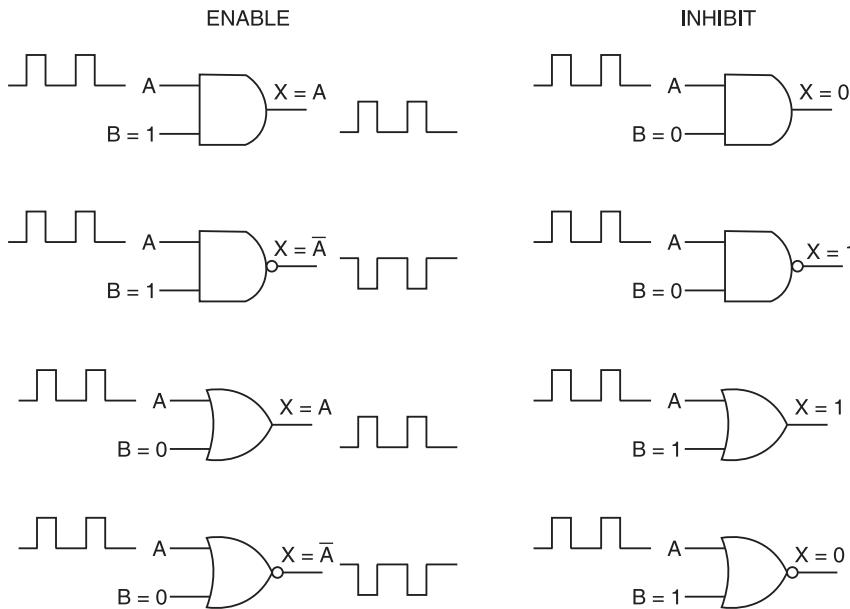
The TTL IC 74LS266, the CMOS IC 74C266 and the high speed CMOS IC 74HC266 contain four each X-NOR gates.

## 4.9 INHIBIT CIRCUITS

AND, OR, NAND, and NOR gates can be used to control the passage of an input logic signal through to the output. This is shown in Figure 4.22 where a logic signal A is applied to one input of each of the above mentioned logic gates. The other input of each gate is the control input B. The logic level at this control input will determine whether the input signal is enabled to reach the output or inhibited from reaching the output.

We notice that when AND and OR gates are enabled, the output follows the A input exactly. Conversely, when NAND and NOR gates are enabled, the output is exactly the inverse of the A input.

We also see that when AND and NOR gates are inhibited, they produce a constant LOW output. Conversely, the NAND and OR gates produce a constant HIGH output in the inhibited condition.



**Figure 4.22** Enable and inhibit circuits.

There will be many situations in digital circuit design, where the passage of a logic signal is either enabled or inhibited depending on the conditions present at one or more control inputs.

**EXAMPLE 4.1** Find the logical equivalent of the following expressions.

- |                        |                        |                 |                 |
|------------------------|------------------------|-----------------|-----------------|
| (a) $A \oplus 0$       | (b) $A \oplus 1$       | (c) $A \odot 0$ | (d) $A \odot 1$ |
| (e) $1 \oplus \bar{A}$ | (f) $0 \oplus \bar{A}$ |                 |                 |

**Solution**

- $A \oplus 0 = A \cdot \bar{0} + \bar{A} \cdot 0 = A \cdot 1 + \bar{A} \cdot 0 = A + 0 = A$
- $A \oplus 1 = A \cdot \bar{1} + \bar{A} \cdot 1 = A \cdot 0 + \bar{A} \cdot 1 = 0 + \bar{A} = \bar{A}$
- $A \odot 0 = \bar{A} \cdot \bar{0} + A \cdot 0 = \bar{A} \cdot 1 + A \cdot 0 = \bar{A} + 0 = \bar{A}$

$$(d) A \odot 1 = \bar{A} \cdot \bar{1} + A \cdot 1 = \bar{A} \cdot 0 + A \cdot 1 = 0 + A = A$$

$$(e) 1 \oplus \bar{A} = 1 \cdot \bar{A} + \bar{1} \cdot \bar{A} = 1 \cdot A + 0 \cdot \bar{A} = A + 0 = A$$

$$(f) 0 \oplus \bar{A} = 0 \cdot \bar{A} + \bar{0} \cdot \bar{A} = 0 \cdot A + 1 \cdot \bar{A} = 0 + \bar{A} = \bar{A}$$

**EXAMPLE 4.2** Show that  $A \oplus B = A\bar{B} + \bar{A}B$  and construct the corresponding logic diagrams.

**Solution**

The truth tables constructed below show that  $A \oplus B = A\bar{B} + \bar{A}B$ . The corresponding logic diagrams are also shown in Figure 4.23.

A	B	$A \oplus B$	A	B	$\bar{A}$	$\bar{B}$	$\bar{A}\bar{B}$	$A\bar{B}$	$A\bar{B} + \bar{A}B$
0	0	0	0	0	1	1	0	0	0
0	1	1	0	1	1	0	1	0	1
1	0	1	1	0	0	1	0	1	1
1	1	0	1	1	0	0	0	0	0

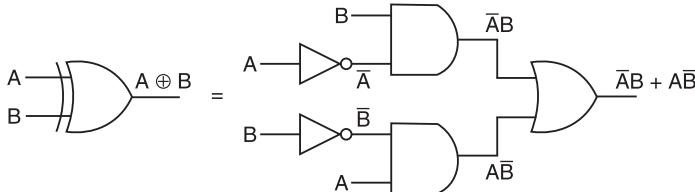


Figure 4.23 Example 4.2.

**EXAMPLE 4.3** Show that  $A \odot B = AB + \bar{A}\bar{B} = \bar{A} \oplus B = \bar{A}\bar{B} + A\bar{B}$ . Also, construct the corresponding logic diagrams.

**Solution**

The truth tables constructed below show that  $A \odot B = AB + \bar{A}\bar{B} = \bar{A} \oplus B = \bar{A}\bar{B} + A\bar{B}$ . The corresponding logic diagrams are also shown in Figure 4.24.

A	B	$A \odot B$	A	B	$\bar{A}$	$\bar{B}$	$AB$	$\bar{A}\bar{B}$	$AB + \bar{A}\bar{B}$	$\bar{A}\bar{B} + A\bar{B}$
0	0	1	0	0	1	1	0	1	1	1
0	1	0	0	1	1	0	0	0	0	0
1	0	0	1	0	0	1	0	0	0	0
1	1	1	1	1	0	0	1	0	1	1

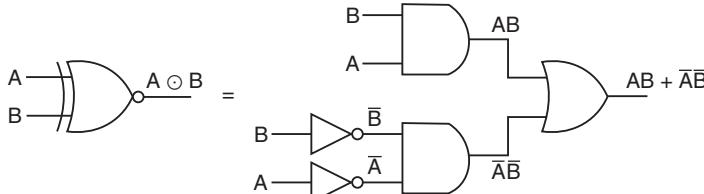


Figure 4.24 Example 4.3.

**EXAMPLE 4.4** Show that  $(A + B)\overline{AB}$  is equivalent to  $A \oplus B$ . Also, construct the corresponding logic diagrams.

**Solution**

The truth tables constructed below show that  $(A + B)\overline{AB} = A \oplus B$ . The logic diagrams are also shown in Figure 4.25.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

A	B	$A + B$	$AB$	$\overline{AB}$	$(A + B)\overline{AB}$
0	0	0	0	1	0
0	1	1	0	1	1
1	0	1	0	1	1
1	1	1	1	0	0

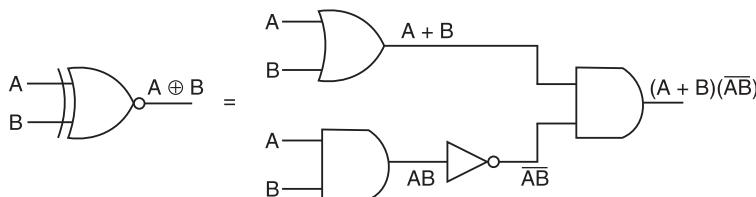


Figure 4.25 Example 4.4.

**EXAMPLE 4.5** Derive a logic expression that equals 1 only when the two binary numbers  $A_1A_0$  and  $B_1B_0$  have the same value. Draw the logic diagram and construct the truth table to verify the logic.

**Solution**

The numbers  $A_1A_0$  and  $B_1B_0$  have the same value when their MSBs ( $A_1$  and  $B_1$ ) coincide and their LSBs ( $A_0$  and  $B_0$ ) coincide. So, we must AND the coincidence  $(A_1 \odot B_1)$  with the coincidence  $(A_0 \odot B_0)$ . Therefore, the desired logic expression is  $(A_1 \odot B_1)(A_0 \odot B_0)$ . The logic diagram and truth table are shown in Figure 4.26. Four variables can combine in 16 ways.

$B_1$	$B_0$	$A_1$	$A_0$	$A_1 \odot B_1$	$A_0 \odot B_0$	$(A_1 \odot B_1)(A_0 \odot B_0)$
0	0	0	0	1	1	1
0	0	0	1	1	0	0
0	0	1	0	0	1	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	1	0	1	1	0	0
0	1	1	0	0	0	0
0	1	1	1	1	1	1
1	0	0	0	0	1	0
1	0	0	1	0	0	0
1	0	1	0	1	1	1
1	0	1	1	1	0	0
1	1	0	0	0	0	0
1	1	0	1	0	1	0
1	1	1	0	1	0	0
1	1	1	1	1	1	1

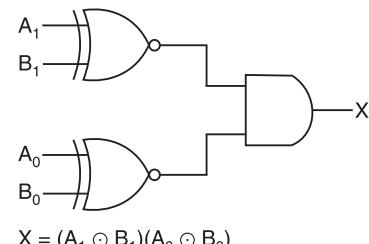


Figure 4.26 Example 4.5.

**EXAMPLE 4.6** Show that  $AB + (\overline{A} + \overline{B})$  is equivalent to  $A \odot B$ . Also construct the corresponding logic diagrams.

**Solution**

The truth tables constructed below show that  $AB + (\overline{A} + \overline{B}) = A \odot B$ . The corresponding logic diagrams are also shown in Figure 4.27.

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

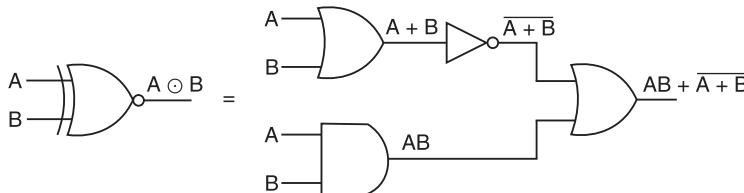


Figure 4.27 Example 4.6.

**EXAMPLE 4.7** Determine which of the following expressions are equivalent to  $A \oplus B$  and which to  $A \odot B$ ?

- (a)  $\overline{A} \oplus B$       (b)  $\overline{A} \odot B$       (c)  $\overline{A} \oplus \overline{B}$       (d)  $\overline{A} \odot \overline{B}$   
 (e)  $A \oplus \overline{B}$       (f)  $A \odot \overline{B}$

**Solution**

- (a)  $\overline{A} \oplus B = \overline{A}\overline{B} + \overline{\overline{A}}B = \overline{A}\overline{B} + AB = A \odot B$   
 (b)  $\overline{A} \odot B = \overline{A}\overline{B} + \overline{A}B = A\overline{B} + \overline{A}B = A \oplus B$   
 (c)  $\overline{A} \oplus \overline{B} = \overline{A}\overline{B} + \overline{\overline{A}}\overline{B} = \overline{A}\overline{B} + AB = A \oplus B$   
 (d)  $\overline{A} \odot \overline{B} = \overline{A}\overline{\overline{B}} + \overline{A}\overline{B} = AB + \overline{A}\overline{B} = A \odot B$   
 (e)  $A \oplus \overline{B} = A\overline{B} + \overline{A}\overline{B} = AB + \overline{A}\overline{B} = A \odot B$   
 (f)  $A \odot \overline{B} = A\overline{B} + \overline{A}\overline{B} = A\overline{B} + \overline{A}B = A \oplus B$

## 4.10 PULSED OPERATION OF LOGIC GATES

In a majority of applications, the inputs to a gate are not stationary levels, but are voltages that change frequently between two logic levels and can be classified as pulse waveforms. We will now look at the operation of all logic gates discussed till now with pulsed waveforms. All the logic gates obey the truth table operation, regardless of whether the inputs are constant levels or pulsed levels. The timing diagrams of various gates for different inputs are shown in the following examples.

**EXAMPLE 4.8** For a two-input AND gate, determine its output waveform in relation to input waveforms of Figure 4.28a.

**Solution**

The output waveform X is shown in Figure 4.28b.

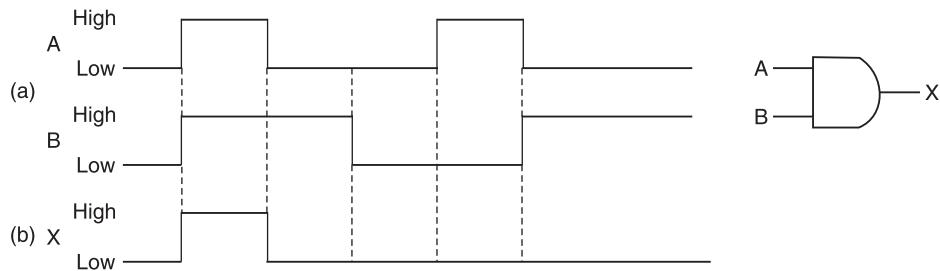


Figure 4.28 Example 4.8.

**EXAMPLE 4.9** If the three waveforms A, B, and C shown in Figure 4.29a are applied to a three-input AND gate, determine the resulting output waveform.

**Solution**

The output waveform X is shown in Figure 4.29b.

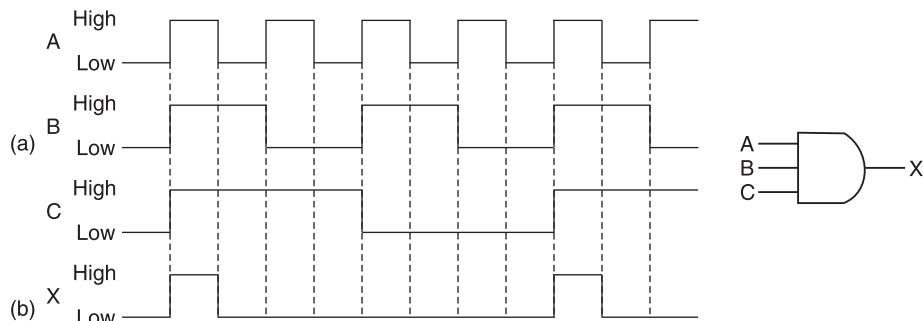


Figure 4.29 Example 4.9.

**EXAMPLE 4.10** For a two-input OR gate, determine its output waveform in relation to the inputs A and B shown in Figure 4.30a.

**Solution**

The output waveform X is shown in Figure 4.30b.

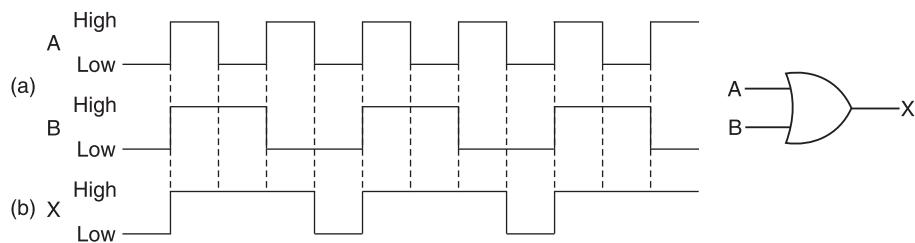


Figure 4.30 Example 4.10.

**EXAMPLE 4.11** For a three-input OR gate, determine its output waveform in proper relation to the inputs A, B and C shown in Figure 4.31a.

**Solution**

The output waveform is shown in Figure 4.31b.

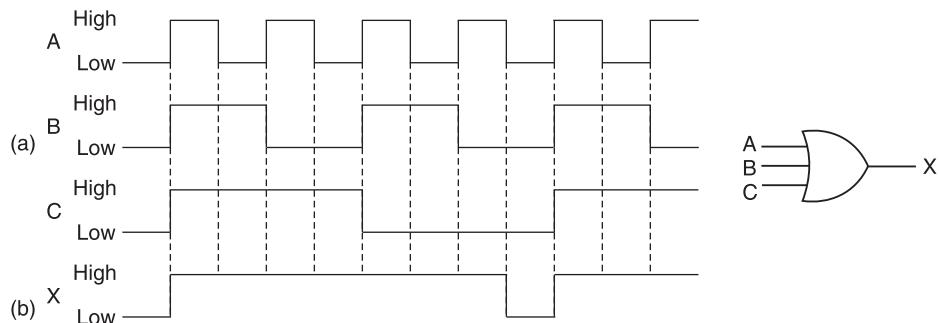


Figure 4.31 Example 4.11.

**EXAMPLE 4.12** If the waveform shown in Figure 4.32a is applied to an inverter, determine the resulting output waveform.

**Solution**

The output waveform is shown in Figure 4.32b.

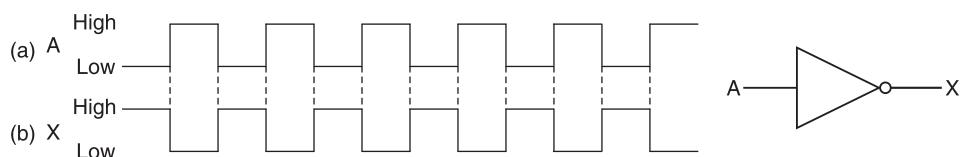


Figure 4.32 Example 4.12.

**EXAMPLE 4.13** The waveforms A and B shown in Figure 4.33a are applied to a two-input NAND gate. Determine the output waveform.

**Solution**

The output of a NAND gate is LOW only when all its inputs are HIGH. The output waveform is shown in Figure 4.33b.

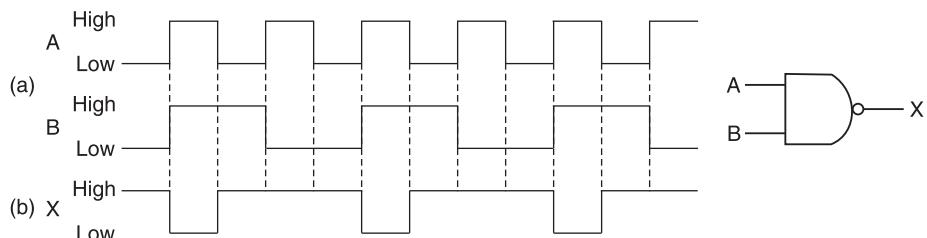


Figure 4.33 Example 4.13.

**EXAMPLE 4.14** The waveforms A, B and C shown in Figure 4.34a are applied to a three-input NAND gate. Determine the output waveform.

**Solution**

The output waveform is shown in Figure 4.34b.

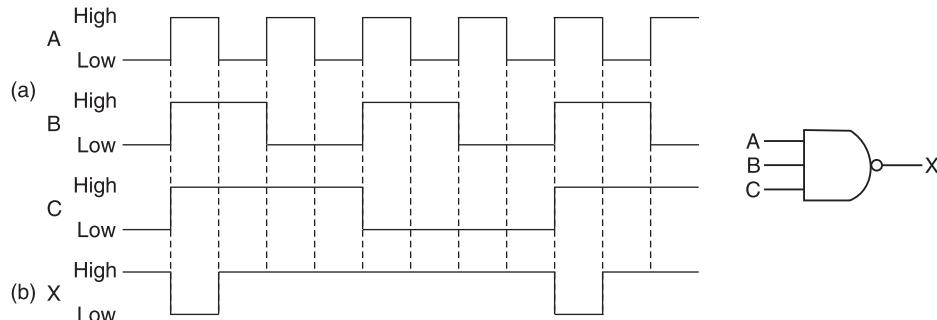


Figure 4.34 Example 4.14.

**EXAMPLE 4.15** For the two-input NAND gate operating as a negative OR gate, determine the output waveform when the input waveforms A and B are as shown in Figure 4.35a.

**Solution**

The output of an active-LOW OR gate is HIGH, if either A is LOW or B is LOW or both A and B are LOW. The output waveform is shown in Figure 4.35b.

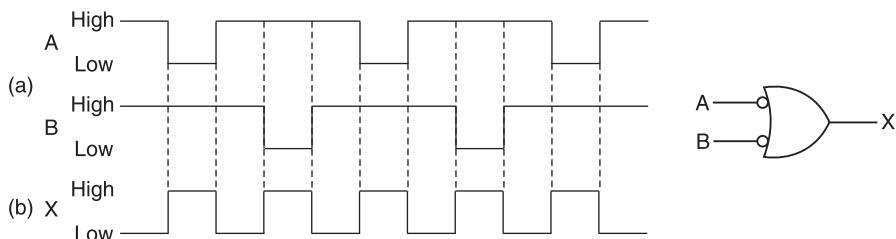


Figure 4.35 Example 4.15.

**EXAMPLE 4.16** If the waveforms A and B shown in Figure 4.36a are applied to a two-input NOR gate, determine the resulting output waveform.

**Solution**

The output of a NOR gate is HIGH only when all its inputs are LOW. The output waveform is shown in Figure 4.36b.

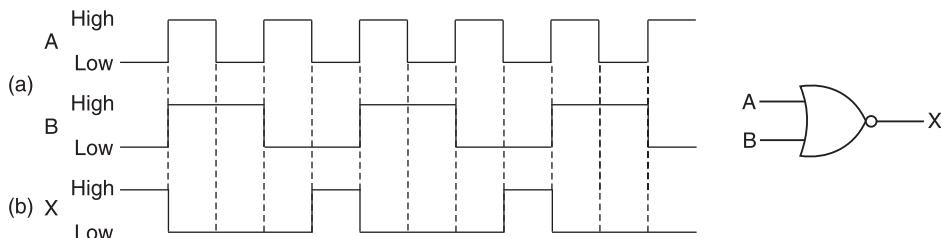


Figure 4.36 Example 4.16.

**EXAMPLE 4.17** The waveforms A and B shown in Figure 4.37a are applied to an active-LOW AND gate. What is the output waveform?

**Solution**

The NOR gate is an active-LOW AND gate. The output of an active-LOW AND gate is HIGH, only when all its inputs are LOW. The output waveform is shown in Figure 4.37b.

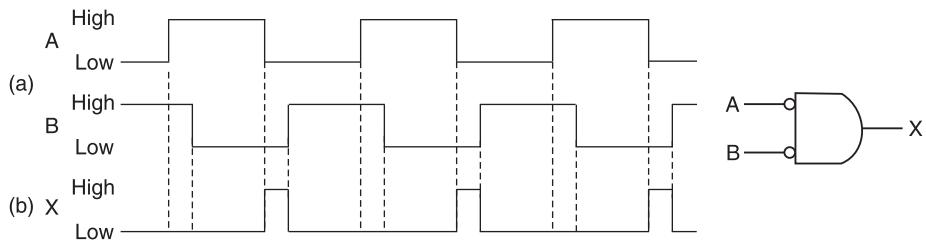


Figure 4.37 Example 4.17.

**EXAMPLE 4.18** If the waveforms A and B shown in Figure 4.38a are applied to a two-input X-OR gate, determine the output waveform.

**Solution**

The output of an X-OR gate is HIGH, only when the inputs are not equal. The output waveform is shown in Figure 4.38b.

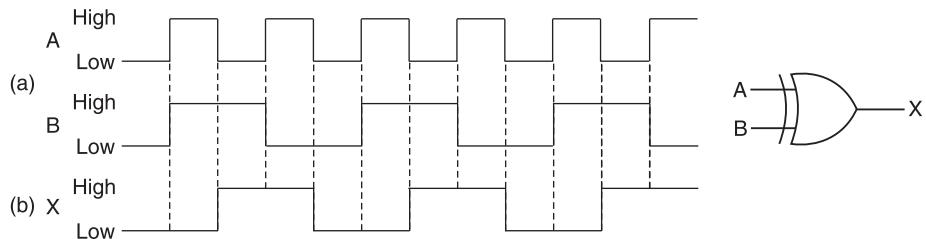


Figure 4.38 Example 4.18.

**EXAMPLE 4.19** If the three waveforms A, B and C, shown in Figure 4.39a, are to be X-ORED, determine the output waveform.

**Solution**

The X-OR output of a number of variables is HIGH, only when an odd number of input variables are HIGH. The output waveform is shown in Figure 4.39b.

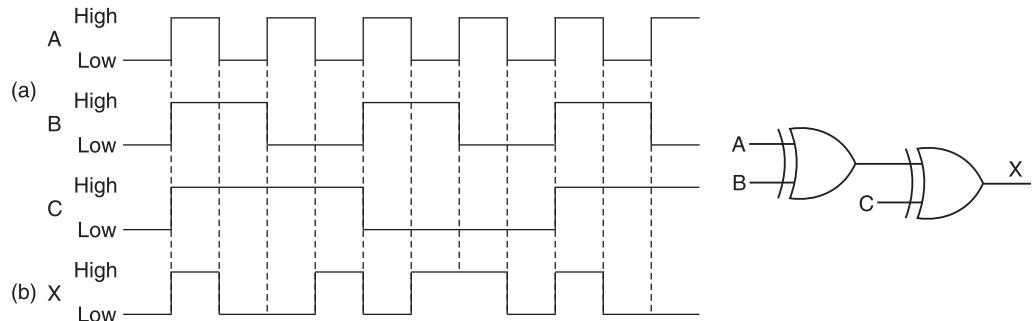


Figure 4.39 Example 4.19.

**EXAMPLE 4.20** If the waveforms A and B shown in Figure 4.40a are applied to a two-input X-NOR gate, determine the output waveform.

**Solution**

The output of an X-NOR gate is HIGH, only when the inputs are equal. The output waveform is shown in Figure 4.40b.

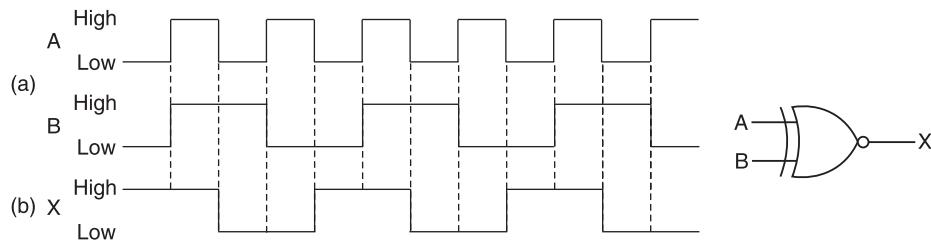


Figure 4.40 Example 4.20.

**EXAMPLE 4.21** If the waveforms A, B, and C shown in Figure 4.41a are X-NORED, determine the output waveform.

**Solution**

The X-NOR output of a number of variables is HIGH, only when an even number of input variables (including 0) are LOW. The output waveform is shown in Figure 4.41b.

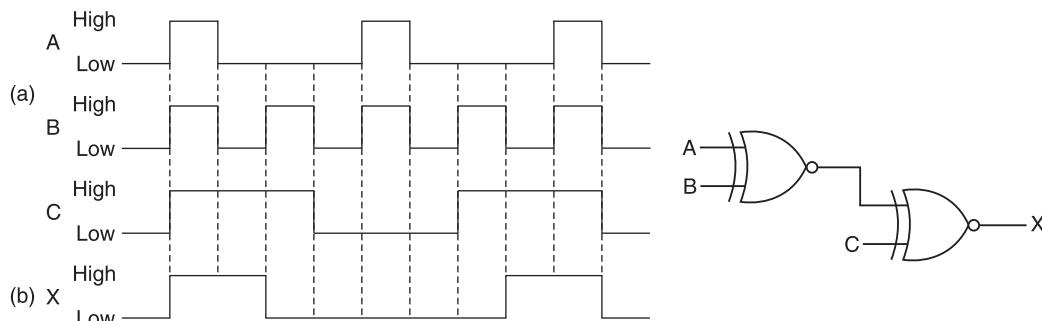


Figure 4.41 Example 4.21.

**EXAMPLE 4.22** Determine the output waveform for the circuit shown in Figure 4.42c, when the inputs A and B shown in Figure 4.42a are applied to it.

**Solution**

The output waveform Y in proper time relationship to inputs A and B is shown in Figure 4.42b. When both the inputs are HIGH, or both the inputs are LOW, the output is LOW. The output is HIGH only when one of the inputs is HIGH. So, it is an X-OR operation. It is an anti-coincidence circuit.

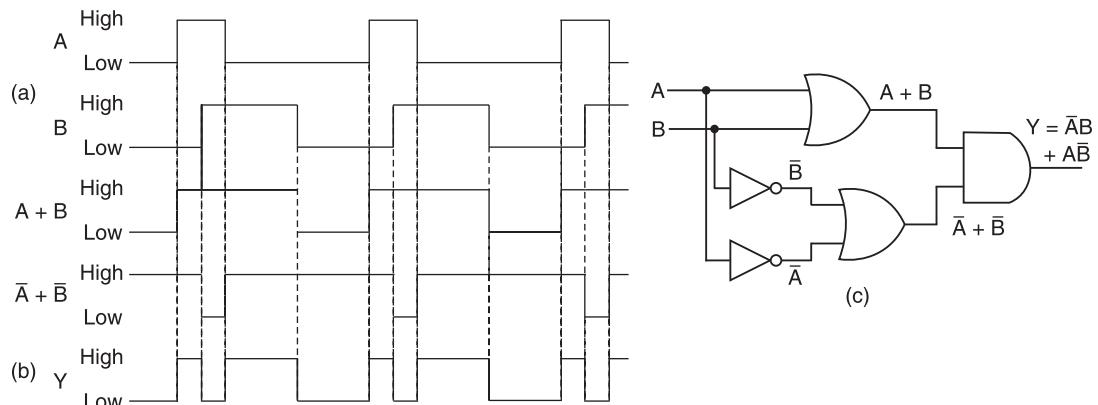


Figure 4.42 Example 4.22.

### SHORT QUESTIONS AND ANSWERS

1. What are logic gates?  
A. Logic gates are the fundamental building blocks of digital systems. They are circuits in which the output depends upon the inputs according to some logic rules.
2. What is a logic circuit?  
A. A logic circuit is an electronic circuit that operates on digital signals in accordance with a logic function.
3. What is logic design?  
A. The interconnection of gates to perform a variety of logical operations is called logic design.
4. What is a truth table?  
A. A truth table is a table which lists all possible combinations of inputs and the corresponding outputs.
5. What is a positive logic system?  
A. A positive logic system is one in which the higher of the two voltage levels represents logic 1 and the lower of the two voltage levels represents logic 0.
6. What is a negative logic system?  
A. A negative logic system is one in which the higher of the two voltage levels represents logic 0 and the lower of the two voltage levels represent logic 1.
7. What is the maximum number of outputs a logic gate can have?  
A. All logic gates can have only one output.
8. What do you mean by level logic?  
A. Level logic means voltage levels represent logic 0 and logic 1.
9. What is an AND gate?  
A. An AND gate is a logic gate with two or more inputs and only one output. It outputs a 1 only when each one of its inputs is a 1. It outputs a 0 even if one of its inputs is a 0. It is a basic gate. It is also called an *all or nothing* gate. It performs AND operation.

- 10.** Which gate is called an all or nothing gate? Why?
- A. An AND gate is called an all or nothing gate; because it produces a 1 only in one case when all its inputs are a 1. In all other cases its output is a zero.
- 11.** What is an OR-gate?
- A. An OR gate is a logic gate with two or more inputs and only one output. It outputs a 1 even if one of its inputs is a 1. It outputs a 0 only when each one of its input is a 0. It is also called an *any* or *all* gate. It is a basic gate. It performs OR operation.
- 12.** Which gate is called any or all gate? Why?
- A. An OR gate is called any or all gate; because it produces a 1 even if one of its inputs is a 1. It produces a 0 only when all the inputs are a 0.
- 13.** What is a NOT gate?
- A. A NOT gate is a logic gate with only one input and one output. Its output is always the complement of its input. It is a basic gate. It performs NOT operation.
- 14.** What is an inverter?
- A. An inverter is nothing but a NOT gate.
- 15.** Which gates can be used as inverters in addition to the NOT gate?
- A. NAND gate, NOR gate, X-OR gate and X-NOR gate can also be used as inverters in addition to the NOT gate.
- 16.** Why AND, OR and NOT gates are called basic gates?
- A. AND, OR, and NOT gates are called basic gates or basic building blocks because any digital circuit of any complexity can be built by using only these three gates.
- 17.** What is a NAND gate?
- A. A NAND gate is a logic gate with two or more inputs and only one output. Its output is 0 only when each one of its input is a 1. Its output is 1 even if one of its input is a 0. It performs NAND operation. It is a universal gate. It can be used as an inverter. It is an AND gate followed by an Inverter. Or a NAND gate is a logic circuit whose output is logic 0 if and only if all of its inputs are logic 1.
- 18.** What do you mean by a bubbled OR gate?
- A. The OR gate with inverted inputs is called a bubbled OR gate. It is also called a negative OR gate. It is an active-LOW OR gate. It is equivalent to a NAND gate.
- 19.** What type of gate is equivalent to a NAND gate followed by an inverter?
- A. A NAND gate followed by an inverter is equivalent to an AND gate.
- 20.** A bubbled NAND gate is equivalent to which gate?
- A. A bubbled NAND gate is equivalent to an OR gate.
- 21.** How can a NAND gate be used as an inverter?
- A. A NAND gate can be used as an inverter by tying all the input terminals together and feeding the signal to be inverted to the common terminal, or by tying all but one input terminal to logic 1 and feeding the signal to be inverted to the remaining terminal.
- 22.** What is a NOR gate?
- A. A NOR gate is a logic gate with two or more inputs and only one output. Its output is 1 only when each one of its inputs is a 0. Its output is 0 even if one of its inputs is a 1. It performs NOR operation. It is a universal gate. It can be used as an inverter also. It is an OR gate followed by an Inverter. Or a NOR gate is a logic gate whose output is logic 1 if and only if all of its inputs are logic 0.

## 156 FUNDAMENTALS OF DIGITAL CIRCUITS

23. How can a NOR gate be used as an inverter?
  - A. A NOR gate can be used as an inverter by tying all the input terminals together and feeding the signal to be inverted to the common terminal, or by tying all but one input terminal to logic 0 and feeding the signal to be inverted to the remaining terminal.
24. What do you mean by a bubbled AND gate?
  - A. The AND gate with inverted inputs is called a bubbled AND gate. It is also called a negative AND gate. It is an active-LOW AND gate. It is equivalent to a NOR gate.
25. A bubbled NOR gate is equivalent to which gate?
  - A. A bubbled NOR gate is equivalent to an AND gate.
26. What type of gate is equivalent to a NOR gate followed by an inverter?
  - A. A NOR gate followed by an inverter is equivalent to an OR gate.
27. Why are NAND and NOR gates called universal gates?
  - A. NAND and NOR gates are called universal gates or universal building blocks because any digital circuit of any complexity can be built by using only NAND gates or only NOR gates.
28. What is an X-OR gate?
  - A. An X-OR gate is a logic gate with only two inputs and one output. Its output is 1 only when its inputs are not equal, i.e. when the inputs do not coincide. Its output is 0 when the inputs are equal, i.e. when the inputs coincide. It is called an anticoincidence gate or inequality detector. It can be used as an inverter. Or an X-OR gate is a two input gate whose output is logic 0 when both the inputs are equal and logic 1 when they are unequal.
29. How can an X-OR gate be used as an inverter?
  - A. An X-OR gate can be used as an inverter by connecting one of its two input terminals to logic 1 and feeding the signal to be inverted to the other terminal. It is a controlled inverter.
30. What is an X-NOR gate?
  - A. An X-NOR gate is a logic gate with only two inputs and one output. Its output is 1 only when its inputs are equal, i.e. when the inputs coincide. Its output is 0 when the inputs are not equal, i.e. when the inputs do not coincide. It is called an equality detector or a coincidence gate. It can be used as an inverter. Or an X-NOR gate is a two-input gate whose output is logic 1 when both the inputs are equal and logic 0 when they are unequal.
31. How can an X-NOR gate be used as an inverter?
  - A. An X-NOR gate can be used as an inverter by connecting one input terminal to logic 0 and feeding the signal to be inverted to the other terminal. It is a controlled inverter.
32. How do you realize three or more input X-OR and X-NOR gates?
  - A. Three or more input X-OR and X-NOR gates do not exist. When three or more variables are to be X-ORED or X-NORED, a number of two input X-OR and X-NOR gates may be used.
33. What do you mean by an active-LOW input gate?
  - A. An active-LOW input gate is one for which the inputs are normally in high state and become effective only when they go to the LOW-state.
34. An AND gate in positive logic system is equivalent to which gate in negative logic system?
  - A. An AND gate in positive logic system is equivalent to an OR gate in negative logic system.
35. An OR gate in positive logic system is equivalent to which gate in negative logic system?
  - A. An OR gate in positive logic system is equivalent to an AND gate in negative logic system.

## **REVIEW QUESTIONS**

1. Draw the logic symbols, construct the truth tables, and with the help of circuit diagrams explain the working of the following gates:  
(i) AND                      (ii) OR                      (iii) NOT                      (iv) NAND  
(v) NOR
  2. Show an arrangement to X-OR and X-NOR the inputs A, B, C and D.

## **FILL IN THE BLANKS**

1. The interconnection of gates to perform a variety of logical operations is called \_\_\_\_\_.
  2. A table which lists all possible combinations of inputs and the corresponding outputs is called a \_\_\_\_\_.
  3. \_\_\_\_\_ is called an all or nothing gate.
  4. \_\_\_\_\_ is called an any or all gate.
  5. All logic gates have \_\_\_\_\_ output.
  6. \_\_\_\_\_ gate is called an equality detector.
  7. \_\_\_\_\_ gate is called an inequality detector.
  8. A bubbled AND gate is equivalent to a \_\_\_\_\_.
  9. A bubbled OR gate is equivalent to a \_\_\_\_\_.
  10. \_\_\_\_\_, \_\_\_\_\_, and \_\_\_\_\_ gates are called basic gates.
  11. \_\_\_\_\_ and \_\_\_\_\_ gates are called universal gates.
  12. \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_, and \_\_\_\_\_ gates can be used as inverters in addition to NOT gate.
  13. A bubbled NAND gate is equivalent to a \_\_\_\_\_ gate.
  14. A bubbled NOR gate is equivalent to a \_\_\_\_\_ gate.

## **OBJECTIVE TYPE QUESTIONS**

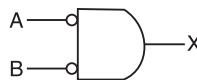
1. A combinational circuit can be designed using only
    - (a) AND gates
    - (b) OR gates
    - (c) OR and X-NOR gates
    - (d) NOR gates
  2. A combinational circuit can be designed using only
    - (a) AND gates
    - (b) OR gates
    - (c) OR and X-NOR gates
    - (d) NAND gates
  3. What is the logic which controls a stair case light associated with two switches A and B located at the bottom and top of the stair case respectively?
    - (a) OR
    - (b) AND
    - (c) X-OR
    - (d) X-NOR

## 158 FUNDAMENTALS OF DIGITAL CIRCUITS

4. The NAND gate can function as a NOT gate if
  - (a) all inputs are connected together
  - (b) inputs are left open
  - (c) one input is set to 0
  - (d) one input is set to 1
5. The NOR gate can function as a NOT gate if
  - (a) all inputs are connected together
  - (b) inputs are left open
  - (c) one input is set to 0
  - (d) one input is set to 1
6. An X-OR gate gives a high output
  - (a) if there are odd number of 1s
  - (b) if it has even number of 0s
  - (c) if the decimal value of digital word is even
  - (d) for odd decimal value.
7. An exclusive NOR gate is logically equivalent to
  - (a) inverter followed by an X-OR gate
  - (b) X-OR gate followed by an inverter
  - (c) NOT gate followed by a NOR gate
  - (d) complement of a NOR gate
8. The X-OR and X-NOR gates can have how many inputs?
  - (a) 2
  - (b) 1
  - (c) 4
  - (d) any number
9. The logic expression  $AB + \overline{A}\overline{B}$  can be implemented by giving the inputs A and B to a two-input
  - (a) NOR gate
  - (b) NAND gate
  - (c) X-OR gate
  - (d) X-NOR gate
10. The logic expression  $A\overline{B} + \overline{A}B$  can be implemented by giving inputs A and B to a two-input
  - (a) NOR gate
  - (b) NAND gate
  - (c) X-OR gate
  - (d) X-NOR gate
11. Which of the following is known as a mod-2 adder?
  - (a) X-OR gate
  - (b) X-NOR gate
  - (c) NAND gate
  - (d) NOR gate
12. What is the minimum number of two-input NAND gates used to perform the function of 2-input OR gate?
  - (a) one
  - (b) two
  - (c) three
  - (d) four
13. NOT gates are to be added to the inputs of which gate to convert it to a NAND gate?
  - (a) OR
  - (b) AND
  - (c) NOT
  - (d) X-OR
14. NOT gates are to be added to the inputs of which gate to convert it to a NOR gate?
  - (a) OR
  - (b) AND
  - (c) NAND
  - (d) X-NOR
15. What logic function is produced by adding inverters to the inputs of an AND gate?
  - (a) OR
  - (b) NOR
  - (c) NAND
  - (d) X-OR
16. What logic function is produced by adding inverters to the inputs of an OR gate?
  - (a) NOR
  - (b) NAND
  - (c) AND
  - (d) X-NOR
17. What logic function is produced by adding an inverter to each input and output of an AND gate?
  - (a) NOR
  - (b) NAND
  - (c) X-OR
  - (d) OR
18. What logic function is produced by adding an inverter to each input and output of an OR gate?
  - (a) NAND
  - (b) NOR
  - (c) AND
  - (d) X-OR
19. Which of the following gates is known as a coincidence detector?
  - (a) AND gate
  - (b) NAND gate
  - (c) X-OR gate
  - (d) X-NOR gate
20. Which of the following gates is known as anticoincidence detector?
  - (a) OR gate
  - (b) X-OR gate
  - (c) AND gate
  - (d) X-NOR gate

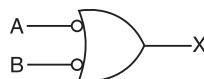


**160** FUNDAMENTALS OF DIGITAL CIRCUITS



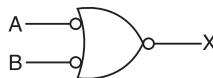
- (a) if and only if both inputs are HIGH      (b) if and only if both the inputs are LOW  
(c) if one of the inputs is LOW                (d) if one of the inputs is HIGH

- 53.** For the gate shown in the figure, the output will be HIGH



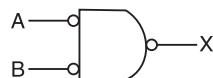


- 54.** For the gate shown in the figure, the output will be LOW



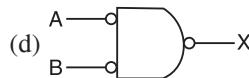
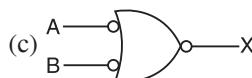
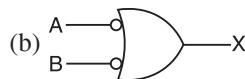
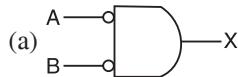


- 55.** For the gate shown in the figure, the output will be LOW





- 56.** Which of the gates shown in the figure is an AND gate?



- 57.** Which of the gates shown in the figure of Q. 56 is a NOR gate?

58. Which of the gates shown in the figure of Q. 56 is an OR gate?

- 59.** Which of the gates shown in the figure of Q. 56 is a NAND gate?

60. Consider the expression  $Z = A \oplus B \oplus C$  where A, B, C are the input variables and Z is the output variable. Z will be logic 0 if

- (a) an even number of input variables are 1      (b) an odd number of input variables are 1  
(c) an even number of input variables are 0      (d) an odd number of inputs variables are 0

61. Consider the expression  $Z = A \oplus B \oplus C$  where A, B, C are the input variables and Z is the output variable. Z will be logic 1 if

- (a) an even number of input variables are 1      (b) an odd number of input variables are 1  
(c) an even number of input variables are 0      (d) an odd number of inputs variables are 0

- 62.** For checking the parity of a digital word, it is preferable to use



63. The most suitable gate to check whether the number of 1s in a digital word is even or odd is  
(a) X-OR                          (b) NAND                          (c) NOR                          (d) AND, OR, and NOT

## PROBLEMS

- 4.1** Show an arrangement to X-OR and X-NOR the inputs A, B, C and D.

**4.2** Draw the logic diagram and construct the truth table for each of the following expressions:

(a)  $X = A + B + \overline{CD}$       (b)  $Y = (AB)(\overline{A} + \overline{B}) + \overline{EF}$       (c)  $Z = \overline{AB} + \overline{CD} + ABC$

**4.3** Draw a logic diagram that implements:

(a)  $A = (Y_1 \oplus Y_2)(Y_3 \odot Y_4) + (Y_5 \oplus Y_6 \oplus Y_7)$   
 (b)  $A = (X_1 \odot X_2) \oplus (X_3 \odot X_4) + (X_4 \oplus X_5) \odot (X_6 \oplus X_7)$

**4.4** Two square waves, A of 1 kHz and B of 2 kHz frequency, are applied as inputs to the following logic gates. Draw the output waveform in each case.

(a) AND      (b) OR      (c) NAND      (d) NOR  
 (e) X-OR      (f) X-NOR

**4.5** Three square waves A, B and C of frequency 1, 2 and 4 kHz, respectively, are to be

(a) ANDed      (b) ORed      (c) NANDed      (d) NORed  
 (e) X-ORed      (f) X-NORed

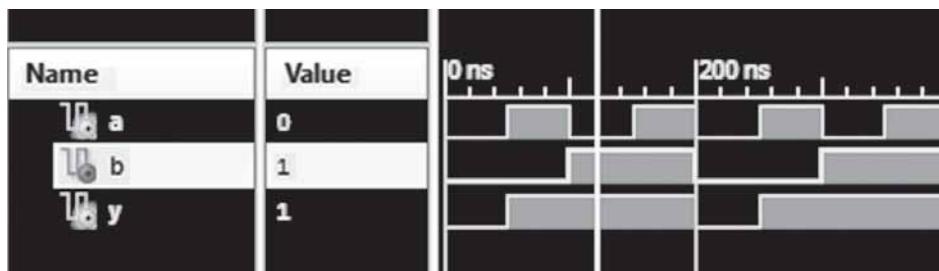
Draw the resultant waveform in each case.

## VHDL PROGRAMS

### 1. VHDL PROGRAM FOR AND GATE USING DATA FLOW MODELING

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ANDGATE is
Port (A, B: in STD_LOGIC;
Y: out STD_LOGIC);
end ANDGATE;
architecture Behavioral of ANDGATE is
begin
Y <= A AND B;
end Behavioral;
```

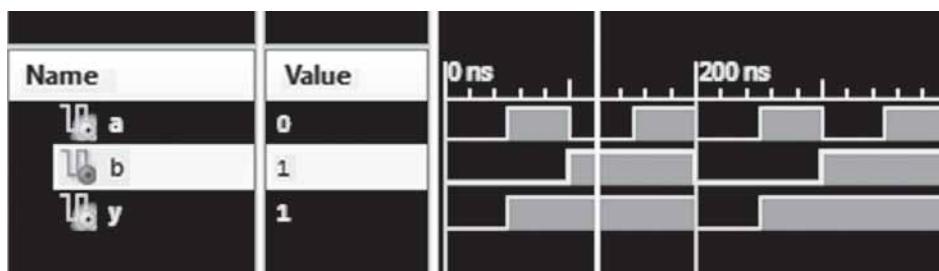
#### SIMULATION OUTPUT:



### 2. VHDL PROGRAM FOR OR GATE USING DATA FLOW MODELING

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ORGATE is
Port ( A,B : in STD_LOGIC;
Y : out STD_LOGIC);
end ORGATE;
architecture Behavioral of ORGATE is
begin
Y <= A OR B;
end Behavioral;
```

#### SIMULATION OUTPUT:

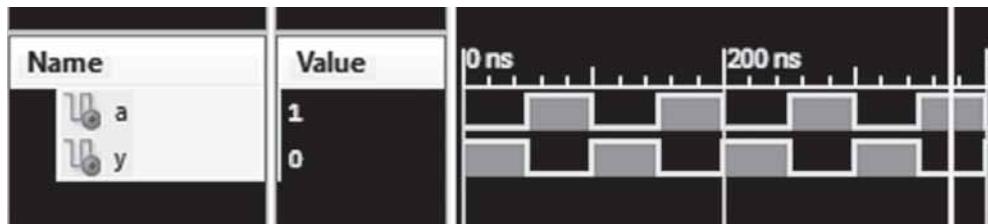


**3. VHDL PROGRAM FOR NOT GATE USING DATA FLOW MODELING**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity NOTGATE is
Port ( A : in STD_LOGIC;
Y : out STD_LOGIC);
end NOTGATE;
architecture Behavioral of NOTGATE is
begin
Y <= (NOT (A));
end Behavioral;

```

**SIMULATION OUTPUT:****4. VHDL PROGRAM FOR NAND GATE USING DATA FLOW MODELING**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity NANDGATE is
Port ( A,B : in STD_LOGIC;
Y : out STD_LOGIC);
end NANDGATE;
architecture Behavioral of NANDGATE is
begin
Y <= A NAND B;
end Behavioral;

```

**SIMULATION OUTPUT:**

## 5. VHDL PROGRAM FOR NOR GATE USING DATA FLOW MODELING

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity NORGATE is
Port ( A,B : in STD_LOGIC;
Y : out STD_LOGIC);
end NORGATE;
architecture Behavioral of NORGATE is
begin
Y <= A NOR B;
end Behavioral;
```

### SIMULATION OUTPUT:



## 6. VHDL PROGRAM FOR X-OR GATE USING DATA FLOW MODELING

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity XORGATE is
Port ( A,B : in STD_LOGIC;
Y : out STD_LOGIC);
end XORGATE;
architecture Behavioral of XORGATE is
begin
Y <= A XOR B;
end Behavioral;
```

### SIMULATION OUTPUT:

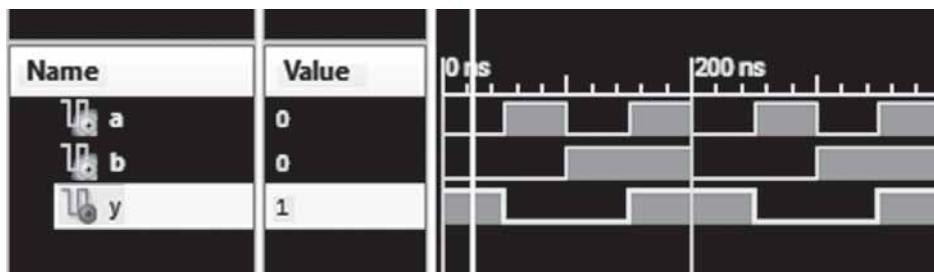


**7. VHDL PROGRAM FOR X-NOR GATE USING DATA FLOW MODELING**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity XNORGATE is
Port ( A,B : in STD_LOGIC;
Y : out STD_LOGIC);
end XNORGATE;
architecture Behavioral of XNORGATE is
begin
Y <= A XNOR B;
end Behavioral;

```

**SIMULATION OUTPUT:****8. VHDL PROGRAM IN STRUCTURAL MODELING FOR OR GATE USING NAND GATE**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ORGATE is
Port (A, B: in STD_LOGIC;
Y: out STD_LOGIC);
end ORGATE;
architecture Behavioral of ORGATE is
component NANDGATE is
Port ( A,B : in STD_LOGIC;
Y : out STD_LOGIC);
end component;
signal n3,n4:STD_LOGIC;
begin
x1:NANDGATE port map(A,A,n3);
x2:NANDGATE port map(B,B,n4);
x3:NANDGATE port map(n3,n4,Y);
end Behavioral;

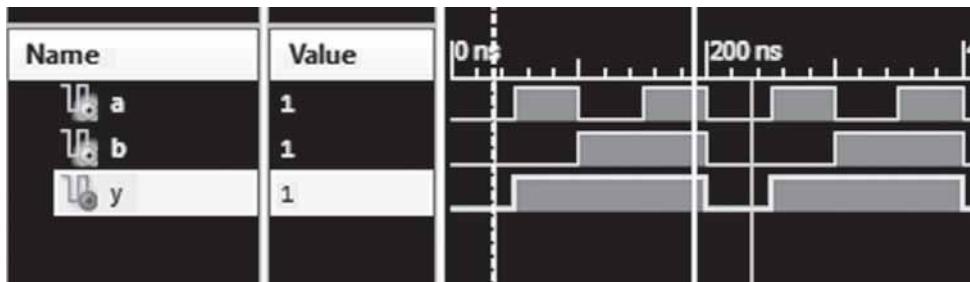
```

**COMPONENT INSTANTIATION VHDL PROGRAM FOR 2-INPUT NAND GATE**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity NANDGATE is
Port ( A,B : in STD_LOGIC;
Y : out STD_LOGIC);
end NANDGATE;
architecture Behavioral of NANDGATE is
begin
Y <= A NAND B;
end Behavioral;

```

**SIMULATION OUTPUT:****9. VHDL PROGRAM IN STRUCTURAL MODELING FOR AND GATE USING NOR GATE**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ANDGATE is
Port (A, B: in STD_LOGIC;
Y: out STD_LOGIC);
end ANDGATE;
architecture Behavioral of ANDGATE is
component NOR_GATE is
Port ( A,B : in STD_LOGIC;
Y : out STD_LOGIC);
end component;
signal n3,n4:STD_LOGIC;
begin
x1:NOR_GATE port map(A,A,n3);
x2:NOR_GATE port map(B,B,n4);
x3:NOR_GATE port map(n3,n4,Y);
end Behavioral;

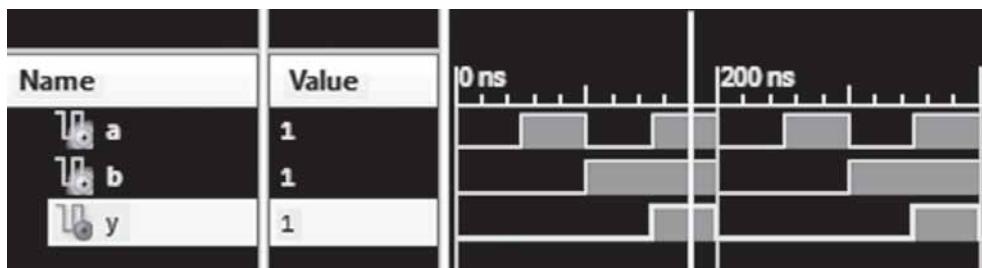
```

**COMPONENT INSTANTIATION VHDL PROGRAM FOR 2-INPUT NOR GATE**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity NORGATE is
Port ( A,B : in STD_LOGIC;
Y : out STD_LOGIC);
end NORGATE;
architecture Behavioral of NORGATE is
begin
Y <= A NOR B;
end Behavioral;

```

**SIMULATION OUTPUT:****10. VHDL PROGRAM IN STRUCTURAL MODELING FOR X-OR GATE USING NAND GATE**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity XORGATE is
Port (A, B: in STD_LOGIC;
Y: out STD_LOGIC);
end XORGATE;
architecture Behavioral of XORGATE is
component NANDGATE is
Port ( A,B : in STD_LOGIC;
Y : out STD_LOGIC);
end component;
signal n1,n2,n3:STD_LOGIC;
begin
x1:NANDGATE port map(A,B,n1);
x2:NANDGATE port map(A,n1,n2);
x3:NANDGATE port map(B,n1,n3);
x4:NANDGATE port map(n2,n3,Y);
end Behavioral;

```

**COMPONENT INSTANTIATION VHDL PROGRAM FOR 2-INPUT X-OR GATE**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity NANDGATE is
Port ( A,B : in STD_LOGIC;
Y : out STD_LOGIC);
end NANDGATE;
architecture Behavioral of NANDGATE is
begin
Y <= A NAND B;
end Behavioral;

```

**SIMULATION OUTPUT:****VERILOG PROGRAMS****1. VERILOG PROGRAM FOR AND GATE USING DATA FLOW MODELING**

```

module and_gate(
    input a,b,
    output y
);
assign y = a & b;
endmodule

```

**SIMULATION OUTPUT:**

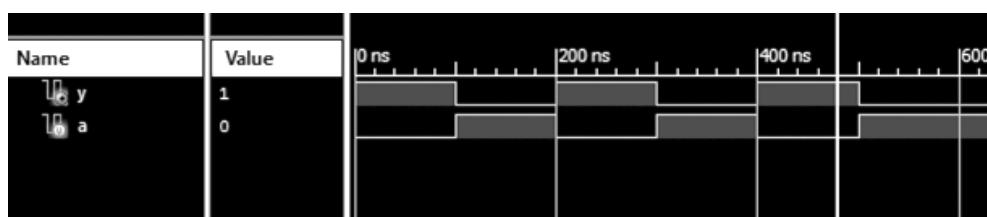
**2. VERILOG PROGRAM FOR OR GATE USING DATA FLOW MODELING**

```
module or_gate(
    input a,b,
    output y
);
    assign y = a | b;
endmodule
```

**SIMULATION OUTPUT:****3. VERILOG PROGRAM FOR NOT GATE USING DATA FLOW MODELING**

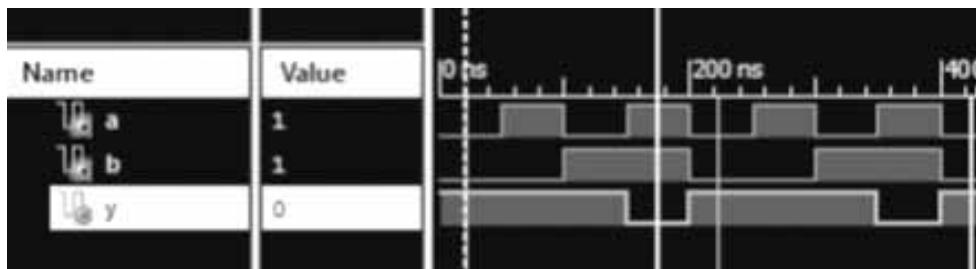
```
module NOTGATE (
    input a,
    output y
);

    assign y = ~a;
endmodule
```

**SIMULATION OUTPUT:****4. VERILOG PROGRAM FOR NAND GATE USING DATA FLOW MODELING**

```
module nand_gate(
    input a,b,
    output y
);
```

```
assign y = (~(a & b));
endmodule
```

**SIMULATION OUTPUT:****5. VERILOG PROGRAM FOR NOR GATE USING DATA FLOW MODELING**

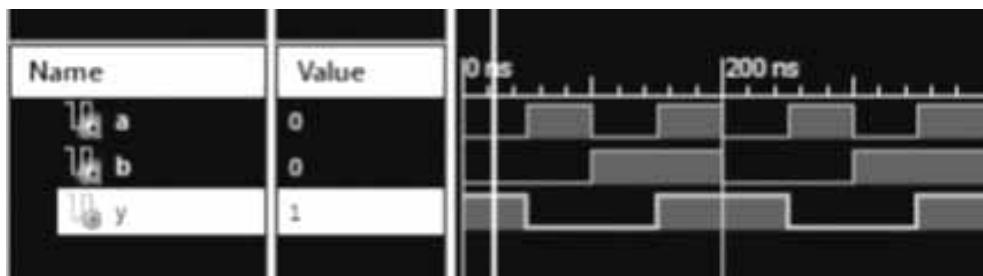
```
module Nor_Gate(
    input a,b,
    output y
);
assign y = (~(a | b));
endmodule
```

**SIMULATION OUTPUT:****6. VERILOG PROGRAM FOR X-OR GATE USING DATA FLOW MODELING**

```
module Xor_Gate(
    input a,b,
    output y
);
assign y = a ^ b;
endmodule
```

**SIMULATION OUTPUT:****7. VERILOG PROGRAM FOR X-NOR GATE USING DATA FLOW MODELING**

```
module Xnor_Gate(
    input a,b,
    output y
);
assign y = (~(a ^ b));
endmodule
```

**SIMULATION OUTPUT:****8. VERILOG PROGRAM IN STRUCTURAL MODELING FOR OR GATE USING NAND GATE**

```
module nand_gate(
    input a,b,
    output y
);
assign y = (~(a & b));
endmodule

module or_gate(a,b,y);

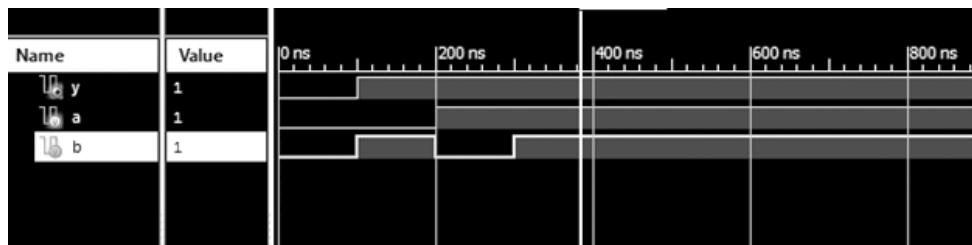
```

```

input a,b;
output y;
wire N2,N3;
nand_gate G1(a,a,N2);
nand_gate G2(b,b,N3);
nand_gate G3(N2,N3,y);
endmodule

```

#### SIMULATION OUTPUT:



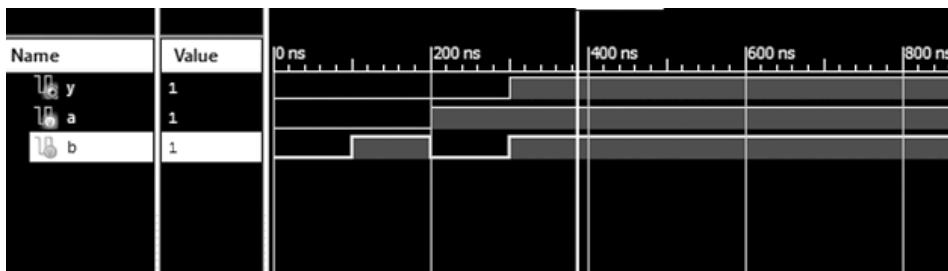
#### 9. VERILOG PROGRAM IN STRUCTURAL MODELING FOR AND GATE USING NOR GATE

```

module Nor_Gate(
    input a,b,
    output y
);
assign y = (~(a | b));
endmodule

module and_gate(a,b,y);
input a,b;
output y;
wire N2,N3;
Nor_Gate G1(a,a,N2);
Nor_Gate G2(b,b,N3);
Nor_Gate G3(N2,N3,y);
Endmodule

```

**SIMULATION OUTPUT:****10. VERILOG PROGRAM IN STRUCTURAL MODELING FOR NOR GATE USING NAND GATE**

```

module nand_gate(
    input a,b,
    output y
);
assign y = (~(a & b)) ;

endmodule

module Nor_Gate(a,b,y) ;
input a,b;
output y;
wire N2,N3,N4;
nand_gate G1(a,a,N2);
nand_gate G2(b,b,N3);
nand_gate G3(N2,N3,N4);
nand_gate G4(N4,N4,y);
endmodule

```

**SIMULATION OUTPUT:**

## 11. VERILOG PROGRAM IN STRUCTURAL MODELING FOR X-OR GATE USING NAND GATE

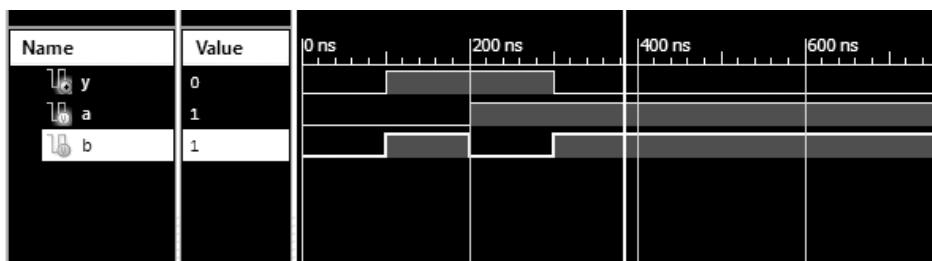
```

module nand_gate(
    input a,b,
    output y
);
assign y = (~(a & b));
endmodule

module Xor_Gate(a,b,y);
input a,b;
output y;
wire N1,N2,N3;
nand_gate G1(a,b,N1);
nand_gate G2(a,N1,N2);
nand_gate G3(b,N1,N3);
nand_gate G4(N2,N3,Y);
endmodule

```

### SIMULATION OUTPUT:



# 5

## BOOLEAN ALGEBRA

### 5.1 INTRODUCTION

Switching circuits are also called logic circuits, gate circuits, and digital circuits. Switching algebra is also called Boolean algebra. Hence the terms switching expressions and Boolean expressions mean the same thing. Boolean algebra is a system of mathematical logic. It is an algebraic system consisting of the set of elements {0,1}, two binary operators called OR and AND and one unary operator called NOT. It is the basic mathematical tool in the analysis and synthesis of switching circuits. It is a way to express logic functions algebraically. Any complex logic statement can be expressed by a Boolean function. The Boolean algebra is governed by certain well-developed rules and laws. In the applications of Boolean algebra in this book, we use capital letters to represent the variables. Any single variable, or a function of the variables can have a value of either a 0 or a 1. The binary digits 0 and 1 are used to represent the two voltage levels that occur within the digital logic circuit. In this book, we follow positive logic. Hence binary 1 represents the higher of the two voltage levels (+ 5 V), and binary 0 represents the lower of the two voltage levels (0 V). Ideally, no other voltages ever occur at the inputs or outputs. In actual practice, however, any voltage above some level (2 V, for example) is treated as logic 1 (TRUE, ON, HIGH) and any voltage below some level (0.8 V, for example) is treated as logic 0 (FALSE, OFF, LOW).

Boolean algebra differs from both the ordinary algebra and the binary number system. In Boolean algebra,  $A + A = A$  and  $A \cdot A = A$ , because the variable A has only a logical value. It doesn't have any numerical significance. In ordinary algebra,  $A + A = 2A$  and  $A \cdot A = A^2$ , because the variable A has a numerical value here. In Boolean algebra,  $1 + 1 = 1$ , whereas in the binary number system,  $1 + 1 = 10$ , and in ordinary algebra,  $1 + 1 = 2$ . There is nothing like subtraction or division in Boolean algebra. Also, there are no negative or fractional numbers in

Boolean algebra. In Boolean algebra, the multiplication and addition of the variables and functions are also only logical. They actually represent logic operations. Logical multiplication is the same as the AND operation, and logical addition is the same as the OR operation. There are only two constants 0 and 1 within the Boolean system, whereas in ordinary algebra, you can have any number of constants. A variable or function of variables in Boolean algebra can assume only two values, either a 0 or a 1, whereas the variables or functions in ordinary algebra can assume an infinite number of values.

Thus, in Boolean algebra

$$\text{If } A = 1, \quad \text{then } A \neq 0$$

$$\text{If } A = 0, \quad \text{then } A \neq 1.$$

Any functional relation in Boolean algebra can be proved by the method of *perfect induction*. Perfect induction is a method of proof, whereby a functional relation is verified for every possible combination of values that the variables may assume. This can be done by forming a truth table. A truth table shows how a logic circuit responds to various combinations of logic levels at its inputs.

## 5.2 LOGIC OPERATIONS

The AND, OR and NOT are the three basic operations or functions that are performed in Boolean algebra. In addition, there are some derived operations such as NAND, NOR, X-OR and X-NOR that are also performed in Boolean algebra. These operations have been described in detail earlier in the context of logic gates.

### 5.2.1 AND Operation

The AND operation in Boolean algebra is similar to multiplication in ordinary algebra. In fact, it is logical multiplication as performed by the AND gate. It is represented by  $\cdot$ ,  $\wedge$ ,  $\cap$  (Intersection).

### 5.2.2 OR Operation

The OR operation in Boolean algebra is similar to addition in ordinary algebra. In fact, it is logical addition as performed by the OR gate. It is represented by  $+$ ,  $\vee$ ,  $\cup$  (Union).

### 5.2.3 NOT Operation

The NOT operation in Boolean algebra is nothing but complementation or inversion, that is, negation as performed by the NOT gate. The NOT operation is indicated by a bar ' $\bar{\cdot}$ ' over the variable or prime. Hierarchy among the operators is parenthesis first, then AND, and OR last.

### 5.2.4 NAND Operation

The NAND operation in Boolean algebra is equivalent to AND operation plus NOT operation, i.e. it is the negation of the AND operation as performed by the NAND gate.

### 5.2.5 NOR Operation

The NOR operation in Boolean algebra is equivalent to OR operation plus NOT operation, i.e. it is the negation of the OR operation as performed by the NOR gate.

### 5.2.6 X-OR and X-NOR Operations

The X-OR and X-NOR operations on variables A and B in Boolean algebra are denoted by  $A \oplus B (= A\bar{B} + \bar{A}B)$  and  $A \odot B (= AB + \bar{A}\bar{B})$ , respectively. These operations have been described in detail earlier.

The X-OR operation is also called the modulo-2 addition since it assigns to each pair of elements its modulo-2 sum.

## 5.3 AXIOMS AND LAWS OF BOOLEAN ALGEBRA

*Axioms* or *postulates* of Boolean algebra are a set of logical expressions that we accept without proof and upon which we can build a set of useful theorems. Actually, axioms are nothing more than the definitions of the three basic logic operations that we have already discussed: AND, OR, and INVERT. Each axiom can be interpreted as the outcome of an operation performed by a logic gate.

AND operation	OR operation	NOT operation
Axiom 1: $0 \cdot 0 = 0$	Axiom 5: $0 + 0 = 0$	Axiom 9: $\bar{1} = 0$
Axiom 2: $0 \cdot 1 = 0$	Axiom 6: $0 + 1 = 1$	Axiom 10: $\bar{0} = 1$
Axiom 3: $1 \cdot 0 = 0$	Axiom 7: $1 + 0 = 1$	
Axiom 4: $1 \cdot 1 = 1$	Axiom 8: $1 + 1 = 1$	

### 5.3.1 Complementation Laws

The term *complement* simply means to invert, i.e. to change 0s to 1s and 1s to 0s. The five laws of complementation are as follows:

Law 1:	$\bar{0} = 1$
Law 2:	$\bar{1} = 0$
Law 3:	If $A = 0$ , then $\bar{\bar{A}} = 1$
Law 4:	If $A = 1$ , then $\bar{\bar{A}} = 0$
Law 5:	$\bar{\bar{A}} = A$ (double complementation law)

Notice that the double complementation does not change the function.

### 5.3.2 AND Laws

The four AND laws are as follows:

Law 1:	$A \cdot 0 = 0$ (Null law)
Law 2:	$A \cdot 1 = A$ (Identity law)
Law 3:	$A \cdot A = A$
Law 4:	$A \cdot \bar{A} = 0$

### 5.3.3 OR Laws

The four OR laws are as follows:

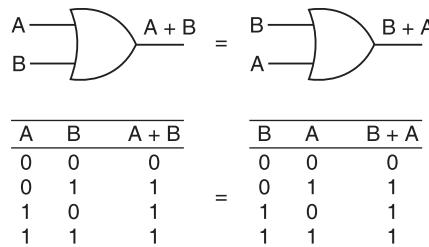
- |        |                            |
|--------|----------------------------|
| Law 1: | $A + 0 = A$ (Null law)     |
| Law 2: | $A + 1 = 1$ (Identity law) |
| Law 3: | $A + A = A$                |
| Law 4: | $A + \bar{A} = 1$          |

### 5.3.4 Commutative Laws

Commutative laws allow change in position of AND or OR variables. There are two commutative laws.

$$\text{Law 1: } A + B = B + A$$

This law states that, A OR B is the same as B OR A, i.e. the order in which the variables are ORed is immaterial. This means that it makes no difference which input of an OR gate is connected to A and which to B. We give below the truth tables illustrating this law.

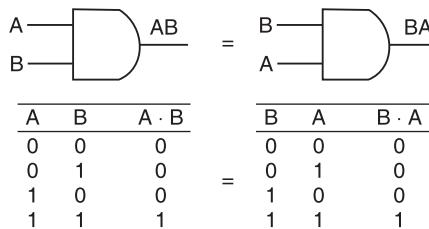


This law can be extended to any number of variables. For example,

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

$$\text{Law 2: } A \cdot B = B \cdot A$$

This law states that A AND B is the same as B AND A, i.e. the order in which the variables are ANDed is immaterial. This means that it makes no difference which input of an AND gate is connected to A and which to B. The truth tables given below illustrate this law.



This law can be extended to any number of variables. For example,

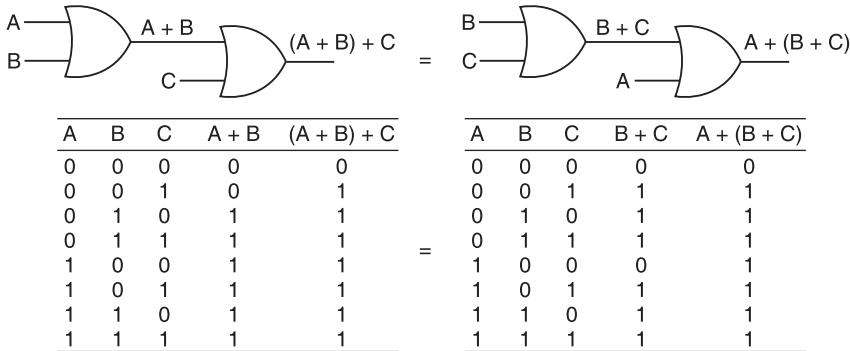
$$A \cdot B \cdot C = B \cdot C \cdot A = C \cdot A \cdot B = B \cdot A \cdot C$$

### 5.3.5 Associative Laws

The associative laws allow grouping of variables. There are two associative laws.

$$\text{Law 1: } (A + B) + C = A + (B + C)$$

A OR B ORed with C is the same as A ORed with B OR C. This law states that the way the variables are grouped and ORed is immaterial. The truth tables given next illustrate this law.

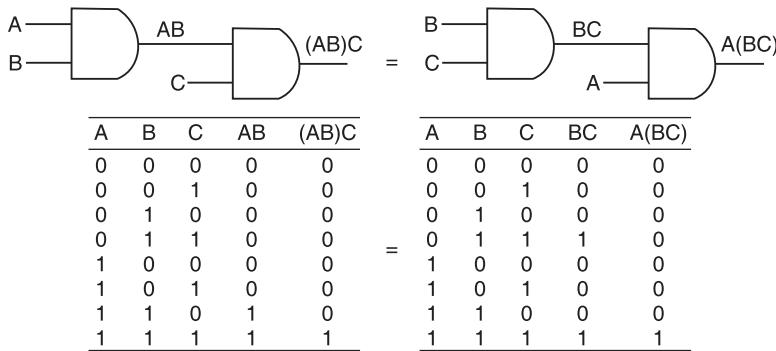


This law can be extended to any number of variables. For example,

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

$$\text{Law 2: } (A \cdot B)C = A(B \cdot C)$$

A AND B ANDed with C is the same as A ANDed with B AND C. This law states that the way the variables are grouped and ANDed is immaterial. See the truth tables below:



This law can be extended to any number of variables. For example,

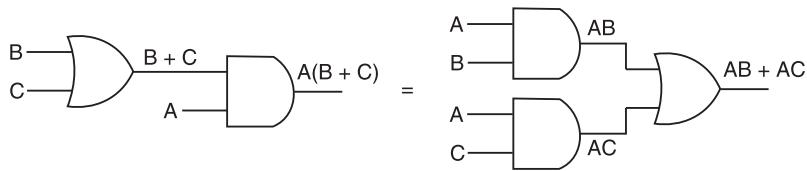
$$A(BCD) = (ABC)D = (AB)(CD)$$

### 5.3.6 Distributive Laws

The distributive laws allow factoring or multiplying out of expressions. There are two distributive laws.

$$\text{Law 1: } A(B + C) = AB + AC$$

This law states that ORing of several variables and ANDing the result with a single variable is equivalent to ANDing that single variable with each of the several variables and then ORing the products. The truth table given below illustrates this law.



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

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

This law applies to single variables as well as combinations of variables. For example,

$$ABC(D + E) = ABCD + ABCE$$

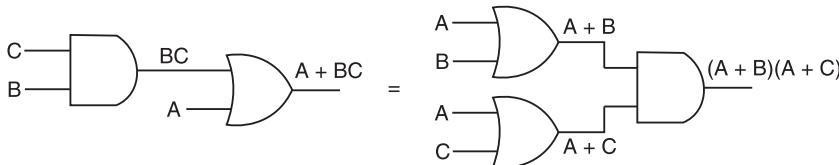
$$AB(CD + EF) = ABCD + ABEF$$

The distributive property is often used in the reverse. That is, given  $AB + AC$ , we replace it by  $A(B + C)$ ; and  $ABC + ABD$  by  $AB(C + D)$ .

$$\text{Law 2: } A + BC = (A + B)(A + C)$$

This law states that ANDing of several variables and ORing the result with a single variable is equivalent to ORing that single variable with each of the several variables and then ANDing the sums. This can be proved algebraically as shown below. Also, the truth tables given next illustrate this law.

$$\begin{aligned}
 \text{RHS} &= (A + B)(A + C) \\
 &= AA + AC + BA + BC \\
 &= A + AC + AB + BC \\
 &= A(1 + C + B) + BC \\
 &= A \cdot 1 + BC \quad (\because 1 + C + B = 1 + B = 1) \\
 &= A + BC \\
 &= \text{LHS}
 \end{aligned}$$



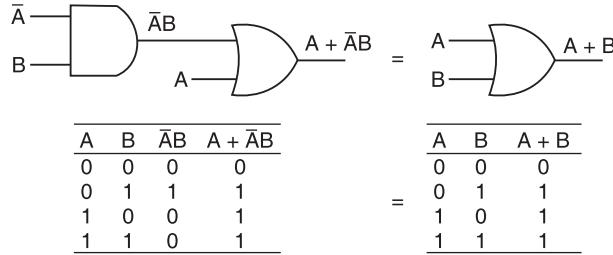
A	B	C	$BC$	$A + BC$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

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

### 5.3.7 Redundant Literal Rule (RLR)

$$\text{Law 1: } A + \bar{A}B = A + B$$

This law states that ORing of a variable with the AND of the complement of that variable with another variable, is equal to the ORing of the two variables. See the truth tables given below.

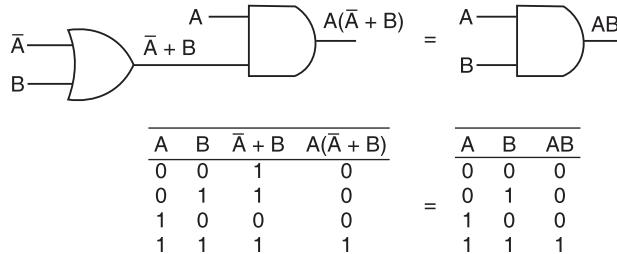


This law can be proved algebraically as shown below.

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

$$\text{Law 2: } A(\bar{A} + B) = AB$$

This law states that ANDing of a variable with the OR of the complement of that variable with another variable, is equal to the ANDing of the two variables. See the truth tables given below.



This law can be proved algebraically as shown below.

$$\begin{aligned} A(\bar{A} + B) &= A\bar{A} + AB \\ &= 0 + AB \\ &= AB \end{aligned}$$

Complement of a term appearing in another term is redundant.

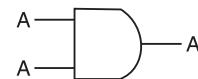
### 5.3.8 Idempotence Laws

$$\text{Law 1: } A \cdot A = A$$

Idempotence means the same value. We are already familiar with the following laws:

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

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

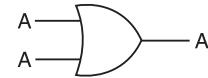


This law states that ANDing of a variable with itself is equal to that variable only.

$$\text{Law 2: } A + A = A$$

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

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



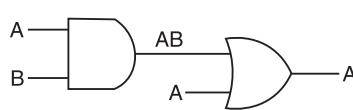
This law states that ORing of a variable with itself is equal to that variable only.

### 5.3.9 Absorption Laws

There are two laws:

$$\text{Law 1: } A + A \cdot B = A$$

This law states that ORing of a variable (A) with the AND of that variable (A) and another variable (B) is equal to that variable itself (A).



A	B	AB	$A + AB$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Algebraically, we have

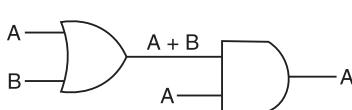
$$A + A \cdot B = A(1 + B) = A \cdot 1 = A$$

Therefore,

$$A + A \cdot \text{Any term} = A$$

$$\text{Law 2: } A(A + B) = A$$

This law states that ANDing of a variable (A) with the OR of that variable (A) and another variable (B) is equal to that variable itself (A).



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

Algebraically, we have

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

Therefore,

$$A(A + \text{Any term}) = A$$

If a term appears in toto in another term, then the latter term becomes redundant and may be removed from the expression without changing its value. Removal of a term is equivalent to replacing that term by 0 if it is in a sum or by 1 if it is in a product.

### 5.3.10 Consensus Theorem (Included Factor Theorem)

$$\text{Theorem 1: } AB + \bar{A}C + BC = AB + \bar{A}C$$

*Proof:* LHS = AB +  $\bar{A}C + BC$

$$\begin{aligned}&= AB + \bar{A}C + BC(A + \bar{A}) \\&= AB + \bar{A}C + BCA + B\bar{C} \\&= AB(1 + C) + \bar{A}C(1 + B) \\&= AB(1) + \bar{A}C(1) \\&= AB + \bar{A}C \\&= RHS\end{aligned}$$

This theorem can be extended to any number of variables. For example,

$$AB + \bar{A}C + BCD = AB + \bar{A}C$$

$$LHS = AB + \bar{A}C + BCD = AB + \bar{A}C + BC + BCD = AB + \bar{A}C + BC = AB + \bar{A}C = RHS$$

**Theorem 2:**  $(A + B)(\bar{A} + C)(B + C) = (A + B)(\bar{A} + C)$

*Proof:* LHS =  $(A + B)(\bar{A} + C)(B + C) = (A\bar{A} + AC + B\bar{A} + BC)(B + C)$   
 $= (AC + BC + \bar{A}B)(B + C)$   
 $= ABC + BC + \bar{A}B + AC + BC + \bar{A}BC = AC + BC + \bar{A}B$

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

If a sum of products comprises a term containing A and a term containing  $\bar{A}$ , and a third term containing the left-out literals of the first two terms, then the third term is redundant, that is, the function remains the same with and without the third term removed or retained.

This theorem can be extended to any number of variables. For example,

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

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

### 5.3.11 Transposition Theorem

**Theorem:**  $AB + \bar{A}C = (A + C)(\bar{A} + B)$

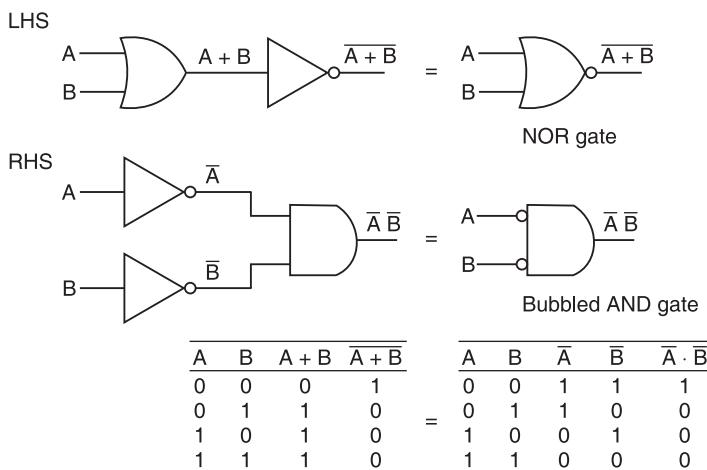
*Proof:* RHS =  $(A + C)(\bar{A} + B)$   
 $= A\bar{A} + C\bar{A} + AB + CB$   
 $= 0 + \bar{A}C + AB + BC$   
 $= \bar{A}C + AB + BC(A + \bar{A})$   
 $= AB + ABC + \bar{A}C + \bar{A}BC$   
 $= AB + \bar{A}C$   
 $= LHS$

### 5.3.12 De Morgan's Theorem

De Morgan's theorem represents two of the most powerful laws in Boolean algebra.

$$\text{Law 1: } \overline{A + B} = \overline{A}\overline{B}$$

This law states that the complement of a sum of variables is equal to the product of their individual complements. What it means is that the complement of two or more variables ORed together, is the same as the AND of the complements of each of the individual variables. Schematically, each side of this law can be represented as:



It shows that the NOR gate is equivalent to a bubbled AND gate. This has also been shown quite simply by truth tables.

This law can be extended to any number of variables or combinations of variables. For example,

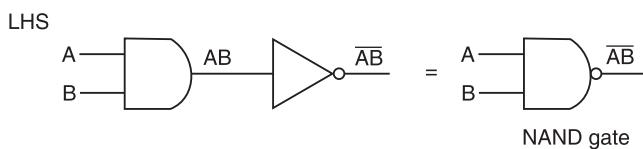
$$\overline{A + B + C + D + \dots} = \overline{A}\overline{B}\overline{C}\overline{D} \dots$$

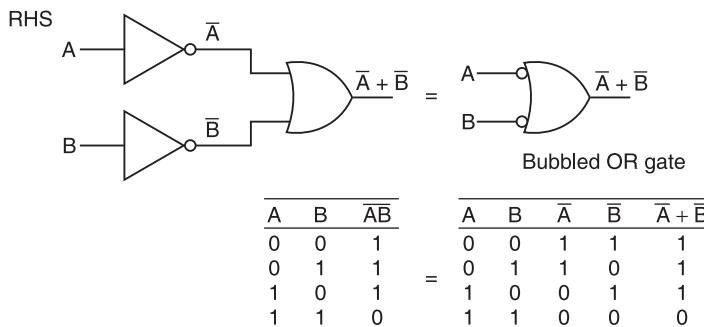
$$\overline{AB + CD + EFG + \dots} = (\overline{AB})(\overline{CD})(\overline{EFG}) \dots = (\overline{A} + \overline{B})(\overline{C} + \overline{D})(\overline{E} + \overline{F} + \overline{G}) \dots$$

It may thus be seen that this law permits removal of individual variables from under a NOT sign and transformation from a sum-of-products form to a product-of-sums form.

$$\text{Law 2: } \overline{AB} = \overline{A} + \overline{B}$$

This law states that the complement of the product of variables is equal to the sum of their individual complements. That is, the complement of two or more variables ANDed together, is equal to the sum of the complements of each of the individual variables. Schematically, we have





It shows that the NAND gate is equivalent to a bubbled OR gate. This has also been shown quite simply by truth tables.

This law can be extended to any number of variables or combinations of variables. For example,

$$\overline{ABCD \dots} = \overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}} + \overline{\overline{D}} + \dots$$

$$\overline{(AB)(CD)(EFG) \dots} = \overline{\overline{AB}} + \overline{\overline{CD}} + \overline{\overline{EFG}} + \dots$$

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

It may also be seen that like law 1, law 2 also permits removal of individual variables from under a NOT sign, and transformation from a product-of-sums form to a sum-of-products form.

It may be seen that the transformations

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

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

can be extended to complicated expressions by the following three steps:

1. Complement the entire given function.
2. Change all the ANDs to ORs and all the ORs to ANDs.
3. Complement each of the individual variables.
4. Change all 0s to 1s and 1s to 0s.

This procedure is called *demorganization* or complementation of switching expressions. It is

$$f(A, B, C, \dots, 0, 1, +, \cdot_c) = f(\overline{A}, \overline{B}, \overline{C}, \dots, 1, 0, \cdot, +)$$

### 5.3.13 Shannon's Expansion Theorem

Shannon's expansion theorem states that any switching expression can be decomposed with respect to a variable A into two parts, one containing A and the other containing  $\overline{A}$ . This concept is useful in decomposing complex machines into an interconnection of smaller components.

$$f(A, B, C, \dots) = A \cdot f(1, B, C, \dots) + \overline{A} \cdot f(0, B, C, \dots)$$

$$f(A, B, C, \dots) = [A + f(0, B, C, \dots)] \cdot [\overline{A} + f(1, B, C, \dots)]$$

**EXAMPLE 5.1** Demorganize  $f = \overline{(A + \overline{B})(C + \overline{D})}$

**Solution**

The given function is

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

Complement the entire function	$= (A + \bar{B})(C + \bar{D})$
Change ANDs to ORs and ORs to ANDs	$= A \cdot \bar{B} + C \cdot \bar{D}$
Complement the variables	$= \bar{A} \cdot B + \bar{C} \cdot D$

**EXAMPLE 5.2** Apply Demorgan's theorem to the expression  $f = \overline{\overline{AB}}(CD + \bar{E}F)(\overline{\overline{AB}} + \overline{\overline{CD}})$ .

**Solution**

The given expression is

$$\begin{aligned} f &= \overline{\overline{AB}}(CD + \bar{E}F)(\overline{\overline{AB}} + \overline{\overline{CD}}) \\ &= \overline{\overline{AB}} + \overline{\overline{CD}} + \overline{\overline{EF}} + \overline{\overline{AB}} + \overline{\overline{CD}} \\ &= AB + (\overline{\overline{CD}} \cdot \overline{\overline{EF}}) + (\overline{\overline{AB}} \cdot \overline{\overline{CD}}) \\ &= AB + (\bar{C} + \bar{D})(E + \bar{F}) + ABCD \end{aligned}$$

A second method of performing demorganization is 'Break the line, change the sign'. For example, if we wish to demorganize the expression  $\overline{AB} + \overline{CDE}$ , we can break the line between A and B, and the line between C and D, and that between D and E and change the sign from ANDing to ORing. This yields  $\bar{A} + \bar{B} + \bar{C} + \bar{D} + \bar{E}$ .

**EXAMPLE 5.3** Reduce the expression  $f = \overline{\overline{AB}} + \overline{\overline{A}} + AB$ .

**Solution**

The given expression is

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

Break the upper line between  $\overline{AB}$  and  $\overline{\overline{A}}$ , and that between  $\overline{\overline{A}}$  and AB, and change the signs between  $\overline{AB}$ ,  $\overline{\overline{A}}$ , AB and simplify.

$$\begin{aligned} f &= \overline{\overline{AB}} \cdot \overline{\overline{A}} \cdot \overline{\overline{AB}} \\ &= AB \cdot A \cdot \overline{AB} \\ &= AB \cdot \overline{AB} \\ &= 0 \end{aligned}$$

Alternatively:

Break the lower line between A and B and change the sign between them and simplify.

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

Break the line between  $\overline{\overline{A}}$  and  $\overline{B}$ , and that between  $\overline{B}$  and AB, and change the sign between them.

$$f = \overline{\overline{A}} \cdot \overline{\overline{B}} \cdot \overline{\overline{AB}} = AB \cdot \overline{AB} = 0$$

Also,

$$f = \overline{\overline{AB}} + \overline{\overline{A}} + AB = 1 + \overline{A} = \overline{1} = 0$$

### 5.3.14 Additional Theorems

**Theorem 1:**  $X \cdot f(X, \bar{X}, Y, \dots, Z) = X \cdot f(1, 0, Y, \dots, Z)$

This theorem states that if a function containing expressions/terms with  $X$  and  $\bar{X}$  is multiplied by  $X$ , then all the  $X$ s and  $\bar{X}$ s in the function can be replaced by 1s and 0s, respectively. This is permissible because,

$$X \cdot X = X = X \cdot 1 \text{ and } X \cdot \bar{X} = 0 = X \cdot 0$$

**Theorem 2:**  $X + f(X, \bar{X}, Y, \dots, Z) = X + f(0, 1, Y, \dots, Z)$

This theorem states that if a function containing expressions/terms with  $X$  and  $\bar{X}$  is added to  $X$ , then all the  $X$ s and  $\bar{X}$ s in the function can be replaced by 0s and 1s, respectively. This is permissible because,

$$X + X = X = X + 0 \quad \text{and} \quad X + \bar{X} = 1 = X + 1.$$

**Theorem 3:**  $f(X, \bar{X}, Y, \dots, Z) = X \cdot f(1, 0, Y, \dots, Z) + \bar{X} \cdot f(0, 1, Y, \dots, Z)$ ,

**Theorem 4:**  $f(X, \bar{X}, Y, \dots, Z) = [X + f(0, 1, Y, \dots, Z)] \cdot [\bar{X} + f(1, 0, Y, \dots, Z)]$

## 5.4 DUALITY

We know that in a positive logic system the more positive of the two voltage levels is represented by a 1 and the more negative by a 0. In a negative logic system the more positive of the two voltage levels is represented by a 0 and the more negative by a 1. This distinction between positive and negative logic systems is important because an OR gate in the positive logic system becomes an AND gate in the negative logic system, and vice versa. Positive and negative logics thus give rise to a basic duality in all Boolean identities. When changing from one logic system to another, 0 becomes 1 and 1 becomes 0. Furthermore, an AND gate becomes an OR gate and an OR gate becomes an AND gate. Given a Boolean identity, we can produce a dual identity by changing all ‘+’ signs to ‘.’ signs, all ‘.’ signs to ‘+’ signs, and complementing all 0s and 1s. The variables are not complemented in this process.

The implication of the duality concept is that once a theorem or statement is proved, the dual also thus stands proved. This is called the principle of duality.

$$[f(A, B, C, \dots, 0, 1, +, \cdot)]_d = f(A, B, C, \dots, 1, 0, \cdot, +)$$

Relations between complement and dual

$$f_c(A, B, C, \dots) = \overline{f(\bar{A}, \bar{B}, \bar{C}, \dots)} = f_d(\bar{A}, \bar{B}, \bar{C}, \dots)$$

$$f_d(A, B, C, \dots) = \overline{f(\bar{A}, \bar{B}, \bar{C}, \dots)} = f_c(\bar{A}, \bar{B}, \bar{C}, \dots)$$

The first relation states that the complement of a function  $f(A, B, C, \dots)$  can be obtained by complementing all the variables in the dual function  $f_d(\bar{A}, \bar{B}, \bar{C}, \dots)$ . Likewise, the second relation states that the dual can be obtained by complementing all the literals in  $\overline{f(A, B, C, \dots)}$ . Some dual identities are given as follows:

### 5.4.1 Duals

<i>Given Expression</i>	<i>Dual</i>
1. $\bar{0} = 1$	$\bar{1} = 0$
2. $0 \cdot 1 = 0$	$1 + 0 = 1$
3. $0 \cdot 0 = 0$	$1 + 1 = 1$
4. $1 \cdot 1 = 1$	$0 + 0 = 0$
5. $A \cdot 0 = 0$	$A + 1 = 1$
6. $A \cdot 1 = A$	$A + 0 = A$
7. $A \cdot A = A$	$A + A = A$
8. $A \cdot \bar{A} = 0$	$A + \bar{A} = 1$
9. $A \cdot B = B \cdot A$	$A + B = B + A$
10. $A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$
11. $A \cdot (B + C) = AB + AC$	$A + BC = (A + B)(A + C)$
12. $A(A + B) = A$	$A + AB = A$
13. $A \cdot (A \cdot B) = A \cdot B$	$A + A + B = A + B$
14. $\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}\bar{B}$
15. $(A + B)(\bar{A} + C)(B + C) = (A + B)(\bar{A} + C)$	$AB + \bar{A}C + BC = AB + \bar{A}C$
16. $(A + C)(\bar{A} + B) = AB + \bar{A}C$	$AC + \bar{A}B = (A + B)(\bar{A} + C)$
17. $A + \bar{B}C = (A + \bar{B})(A + C)$	$A(\bar{B} + C) = (A\bar{B} + AC)$
18. $(A + B)(C + D) = AC + AD + BC + BD$	$(AB + CD) = (A + C)(A + D)(B + C)(B + D)$
19. $A + B = AB + \bar{A}B + A\bar{B}$	$AB = (A + B)(\bar{A} + B)(A + \bar{B})$
20. $A + B(C + \overline{DE}) = A + B\bar{C}DE$	$A[B + (C \cdot \overline{D + E})] = A \cdot (B + \bar{C} + D + E)$
21. $\overline{\overline{AB} + \bar{A} + AB} = 0$	$\overline{A + B \cdot \bar{A} \cdot (A + B)} = 1$
22. $AB + \bar{A}C + A\bar{B}C (AB + C) = 1$	$(A + B)(\bar{A} + C) \cdot [(A + \bar{B} + C) + (A + B)C] = 0$
23. $ABD + ABCD = ABD$	$(A + B + D)(A + B + C + D) = (A + B + D)$
24. $\overline{\overline{AB} + ABC} + A(B + A\bar{B}) = 0$	$(A + \bar{B}) \cdot (A + B + C) \cdot (A + [B(A + \bar{B})]) = 1$
25. $A + \bar{B}C(A + \bar{B}C) = A + \bar{B}C$	$A \cdot [(\bar{B} + C) + A \cdot (\bar{B} + C)] = A \cdot (\bar{B} + C)$

## 5.5 REDUCING BOOLEAN EXPRESSIONS

Every Boolean expression must be reduced to as simple a form as possible before realization, because every logic operation in the expression represents a corresponding element of hardware. Realization of a digital circuit with the minimal expression, therefore, results in reduction of cost and complexity and the corresponding increase in reliability. To reduce Boolean expressions, all the laws of Boolean algebra may be used. The techniques used for these reductions are similar to those used in ordinary algebra. The procedure is:

- (a) Multiply all variables necessary to remove parentheses.
- (b) Look for identical terms. Only one of those terms be retained and all others dropped. For example,

$$AB + AB + AB + AB = AB$$

- (c) Look for a variable and its negation in the same term. This term can be dropped. For example,

$$A \cdot B\bar{B} = A \cdot 0 = 0; \quad ABC\bar{C} = AB \cdot 0 = 0$$

- (d) Look for pairs of terms that are identical except for one variable which may be missing in one of the terms. The larger term can be dropped. For example,

$$AB\bar{C}\bar{D} + AB\bar{C} = AB\bar{C}(\bar{D} + 1) = AB\bar{C} \cdot 1 = AB\bar{C}$$

- (e) Look for pairs of terms which have the same variables, with one or more variables complemented. If a variable in one term of such a pair is complemented while in the second term it is not, then such terms can be combined into a single term with that variable dropped. For example,

$$AB\bar{C}\bar{D} + AB\bar{C}D = AB\bar{C}(\bar{D} + D) = AB\bar{C} \cdot 1 = AB\bar{C}$$

$$AB(C + D) + AB(\bar{C} + \bar{D}) = AB[(C + D) + (\bar{C} + \bar{D})] = AB \cdot 1 = AB$$

**EXAMPLE 5.4** Reduce the expression  $f = A[B + \bar{C}(AB + A\bar{C})]$ .

**Solution**

The given expression is

$$f = A[B + \bar{C}(AB + A\bar{C})]$$

Demorganize  $\bar{AB} + A\bar{C}$

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

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

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

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

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

Simplify

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

Simplify

$$= A(B + \bar{C}\bar{A} + 0 + \bar{C}\bar{B}\bar{A} + 0)$$

Simplify

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

Simplify

$$= AB + 0 + 0$$

Simplify

$$= AB$$

**EXAMPLE 5.5** Reduce the expression  $f = A + B[AC + (B + \bar{C})D]$ .

**Solution**

The given expression is

$$f = A + B[AC + (B + \bar{C})D]$$

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

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

Expand  $B(AC + BD + \bar{C}D)$

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

Write in order

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

Factor

$$= A(1 + BC) + BD(1 + \bar{C})$$

Reduce

$$= A \cdot 1 + BD \cdot 1$$

Simplify

$$= A + BD$$

**EXAMPLE 5.6** Reduce the expression  $f = \overline{(A + \overline{BC})(A\bar{B} + ABC)}$ .

**Solution**

The given expression is

$$f = \overline{(A + \overline{BC})(A\bar{B} + ABC)}$$

$$\begin{aligned}
 \text{Demorganize } & (A + \overline{BC}) = (\overline{ABC})(A\bar{B} + ABC) \\
 \text{Simplify} & = (\overline{ABC})(A\bar{B} + ABC) \\
 \text{Multiply} & = \overline{ABC} A\bar{B} + \overline{ABC} ABC \\
 \text{Rearrange} & = A\bar{A}B\bar{C} + A\bar{A}BCC \\
 & = 0 + 0 = 0
 \end{aligned}$$

**EXAMPLE 5.7** Reduce the expression  $f = (B + BC)(B + \overline{BC})(B + D)$ .

**Solution**

The given expression is

$$\begin{aligned}
 f &= (B + BC)(B + \overline{BC})(B + D) \\
 &= (BB + BCB + B\overline{BC} + BC\overline{BC})(B + D) \\
 &= (B + BC + 0 + 0)(B + D) \\
 &= B(1 + C)(B + D) \\
 &= B(B + D) \\
 &= BB + BD \\
 &= B(1 + D) = B
 \end{aligned}$$

**EXAMPLE 5.8** Show that  $AB + A\overline{BC} + B\overline{C} = AC + B\overline{C}$ .

**Solution**

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

**EXAMPLE 5.9** Show that  $A\overline{BC} + B + B\overline{D} + A\overline{BD} + \overline{AC} = B + C$ .

**Solution**

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

**EXAMPLE 5.10** Simplify the function

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

**Solution**

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

Applying redundant literal rule (RLR) to the 1st and 2nd terms of  $f$

$$f = A + BC + \overline{AB} + \overline{AC} = A + \overline{AB} + \overline{AC} + BC$$

Applying RLR to 1st and 2nd terms, and 1st and 3rd terms

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

Applying Idempotence law to 1st and 3rd terms

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

Applying RLR to 2nd and 4th terms

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

Applying idempotence law to the 3rd and 4th terms

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

**EXAMPLE 5.11** Using the consensus theorem show that

$$f(A, B, C) = A\overline{B} + B\overline{C} + C\overline{A} = \overline{AB} + \overline{BC} + \overline{CA}$$

**Solution**

Applying the consensus theorem to the 1st and 2nd, 2nd and 3rd, and 3rd and 1st terms of LHS, we get three redundant terms as  $A\overline{C}$ ,  $B\overline{A}$ , and  $C\overline{B}$ . Adding them to LHS

$$f = A\overline{B} + B\overline{C} + C\overline{A} + \overline{AB} + \overline{BC} + \overline{CA}$$

Now applying consensus theorem to 4th and 5th, 5th and 6th, 6th and 4th terms, the terms 3, 1, and 2 become redundant. So removing 1st, 2nd and 3rd terms, we have

$$f = \overline{AB} + \overline{BC} + \overline{CA} = RHS$$

This is an example of  $f(A, B, C) = f(\overline{A}, \overline{B}, \overline{C})$ , that is, by complementing all the literals the function remains the same.

**EXAMPLE 5.12** Simplify the function

$$f(A, B, C, D) = \overline{AB} + \overline{BC} + \overline{AD} + CD$$

**Solution**

Applying the consensus theorem to the 2nd and 4th terms, we may add the redundant term  $\overline{BD}$ .

$$\therefore f = \overline{AB} + \overline{BC} + \overline{AD} + CD + \overline{BD}.$$

Applying the consensus theorem to the 3rd and 5th terms, the term  $\overline{AB}$  becomes redundant. Removing that term from  $f$

$$f = \overline{BC} + \overline{AD} + CD + \overline{BD}$$

Applying the consensus theorem to the 1st and 3rd terms, the term  $\overline{BD}$  becomes redundant. Removing that term from  $f$

$$f = \overline{BC} + \overline{AD} + CD$$

**EXAMPLE 5.13** Using the consensus theorem show that

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

**Solution**

Observe that in LHS, 1st and 4th terms contain  $A, \bar{A}$ ; 2nd and 1st terms contain  $B, \bar{B}$ ; 3rd and 2nd terms contain  $C, \bar{C}$ ; 4th and 3rd terms contain  $D, \bar{D}$ . So applying the consensus theorem and adding redundant terms obtained, we get

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

Applying the consensus theorem to the 1st and 8th, 2nd and 5th, 3rd and 6th, and 4th and 7th terms and adding the redundant terms to LHS, we get

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

Looking at the terms of the 2nd set and 3rd set, we observe that the 1st set is redundant. So remove it. Looking at the terms of the 3rd set, we observe that 2nd set is redundant. So remove it. The terms of the 3rd set are identical with the RHS of the given expression. So  $\text{LHS} = \text{RHS}$ .

**EXAMPLE 5.14** Define the connective \* for two valued variables A, B, and C as follows:

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

Let  $C = A * B$ , and determine which of the following is valid.

- (a)  $A = B * C$       (b)  $B = A * C$       (c)  $A * B * C = 1$

**Solution**

If

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

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

- (a)  $B * C = BC + \bar{B}\bar{C} = B(AB + \bar{A}\bar{B}) + \bar{B}(\bar{A}\bar{B} + A\bar{B}) = AB + 0 + 0 + A\bar{B} = A(B + \bar{B}) = A$   
 (b)  $A * C = \bar{A}\bar{C} + AC = \bar{A}(A\bar{B} + \bar{A}\bar{B}) + A(\bar{A}\bar{B} + AB) = 0 + \bar{A}\bar{B} + AB = B(A + \bar{A}) = B$   
 (c)  $A * B * C = C * C$  (since  $A * B = C$ )  $= \bar{C}\bar{C} + CC = \bar{C} + C = 1$

So all the three are valid.

## 5.6 FUNCTIONALLY COMPLETE SETS OF OPERATIONS

OR, AND and NOT (+, ·,  $\neg$ ) form a functionally complete set in the sense that any function can be realized using these operators in SOP or POS form. With the help of De Morgan's theorem, it is possible to produce  $A \cdot B$  using only the set of operators (+,  $\neg$ ). Likewise  $A + B$  can be realized using only the operators ( $\cdot$ ,  $\neg$ ). These sets, each comprising two operators only ( $\cdot$ ,  $\neg$ ) or (+,  $\neg$ ) are said to be functionally complete sets. Further using only the NAND operator or the NOR operator, it is possible to produce all the Boolean operations. Hence each one of them forms a functionally complete single element set.

## 5.7 BOOLEAN FUNCTIONS AND THEIR REPRESENTATION

A function of ‘n’ Boolean variables denoted by  $f(x_1, x_2, \dots, x_n)$  is another variable of algebra and takes one of the two possible values, 0 and 1. The various ways of representing a given function are given in this section.

**1. Sum-of-products (SOP) form:** This form is also called the Disjunctive Normal Form (DNF). For example,  $f(A, B, C) = \bar{A}B + \bar{B}C$ .

**2. Product-of-sums (POS) form:** This form is also called the Conjunctive Normal Form (CNF). The function of above equation may also be written in the form shown in equation below. By multiplying it out and using the consensus theorem, we can see that it is the same as

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

**3. Truth table form:** In this form, the function is specified by listing all possible combinations of values assumed by the variables and the corresponding values of the function (see Table 5.1).

**Table 5.1** Truth table for  $f(A, B, C) = \bar{A}B + \bar{B}C$

Decimal code	A	B	C	$f(A, B, C)$
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

**4. Standard sum-of-products form:** This form is also called Disjunctive Canonical Form (DCF). It is also called the Expanded Sum of Products Form or Canonical Sum-of-Products Form. In this form, the function is the sum of a number of product terms where each product term contains all the variables of the function either in complemented or uncomplemented form. This can be derived from the truth table by finding the sum of all the terms that correspond to those combinations (rows) for which ‘f’ assumes the value 1. It can also be obtained from the SOP form algebraically as shown below.

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

A product term which contains all the variables of the function either in complemented or uncomplemented form is called a *minterm*. A minterm assumes the value 1 only for one combination of the variables. An  $n$  variable function can have in all  $2^n$  minterms. The sum of the minterms whose value is equal to 1 is the standard sum of products form of the function.

The minterms are often denoted as  $m_0, m_1, m_2, \dots$ , where the suffixes are the decimal codes of the combinations. For a 3-variable function  $m_0 = \bar{A}\bar{B}\bar{C}$ ,  $m_1 = \bar{A}\bar{B}C$ ,  $m_2 = \bar{A}B\bar{C}$ ,  $m_3 = \bar{A}BC$ ,  $m_4 = A\bar{B}\bar{C}$ ,  $m_5 = A\bar{B}C$ ,  $m_6 = AB\bar{C}$ , and  $m_7 = ABC$ . Another way of representing the function in canonical SOP form is by showing the sum of minterms for which the function equals 1.

Thus

$$f(A, B, C) = m_1 + m_2 + m_3 + m_5$$

Yet another way of representing the function in DCF is by listing the decimal codes of the minterms for which  $f = 1$ .

Thus

$$f(A, B, C) = \Sigma m(1, 2, 3, 5)$$

where  $\Sigma m$  represents the sum of all the minterms whose decimal codes are given in the parenthesis.

**5. Standard product-of-sums form:** This form is also called Conjunctive Canonical Form (CCF). It is also called Expanded Product-of-Sums Form or Canonical Product-of-Sums Form. This is derived by considering the combinations for which  $f = 0$ . Each term is a sum of all the variables. A variable appears in uncomplemented form if it has a value of 0 in the combination and appears in complemented form if it has a value of 1 in the combination. For example, the sum corresponding to row 4 in the above truth table (Table 2.1) is  $(\bar{A} + B + C)$ . This means that wherever  $\bar{A}$  is 0, (same as  $A = 1$ ), B is 0 and C is 0, this term becomes 0 and being a term in a product, the function assumes the value 0. Thus, the function  $f(A, B, C) = (\bar{A} + \bar{B})(A + B)$  is given by the product of sums

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

A sum term which contains each of the  $n$  variables in either complemented or uncomplemented form is called a *maxterm*. A maxterm assumes the value 0 only for one combination of the variables. For all other combinations it will be 1. There will be at the most  $2^n$  maxterms. The product of maxterms corresponding to the rows for which  $f = 0$ , is the standard or canonical product of sums form of the function.

Maxterms are often represented as  $M_0, M_1, M_2, \dots$ , where the suffixes denote their decimal code. Thus the CCF of  $f$  may be written as

$$f(A, B, C) = M_0 \cdot M_4 \cdot M_6 \cdot M_7 \quad \text{or simply as}$$

$$f(A, B, C) = \prod M(0, 4, 6, 7)$$

where  $\prod$  represents the product of all maxterms whose decimal code is given within the parenthesis. The symbol ‘ $\wedge$ ’ is also used in place of  $\prod$ .

**6. Venn diagram form:** As the algebra of sets and Boolean algebra are similar systems, Boolean expressions can be represented by a Venn diagram in which each variable is considered as a set. The AND operation is considered as an intersection and the OR operation is considered as a union. Complementation corresponds to outside the set. Thus the function  $f = \bar{A}B + \bar{B}C$  is represented as shown in Figure 5.1.

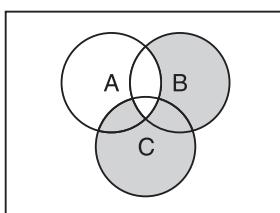


Figure 5.1 Venn diagram form.

**7. Octal designation:** Look at the truth table of the function and the column in which the truth values of the function for each combination (minterm) are indicated. If these values are indicated in the order starting from  $m_7$  at the extreme left and proceeding to  $m_0$  at the extreme right, the resulting string of 0s and 1s is called the *characteristic vector* for the function. If written as an octal number, it becomes the octal designation for the function which is illustrated below.

$m_7$	$m_6$	$m_5$	$m_4$	$m_3$	$m_2$	$m_1$	$m_0$
0	0	1	0	1	1	1	0

Octal designation of  $f(A, B, C) = (0, 5, 6)_8$ .

**8. Karnaugh map:** In this representation we put the truth table in a compact form by labelling the rows and columns of a map. This is extremely useful and extensively used in the minimization of functions of 3, 4, 5 or 6 variables. The rows and columns are assigned a binary code (Gray code) such that two adjacent rows or columns differ in one bit only. Notice that the column on the extreme left is adjacent to the column on the extreme right. Likewise the top row and the bottom row are adjacent. The Karnaugh map consists of a number of squares. Each one of the squares represents a minterm or a maxterm. Each square is called a cell. Two squares are said to be adjacent to each other if they are physically adjacent to each other or can be made adjacent by wrapping the map from left to right or top to bottom. Typical maps and their row and column designations are shown in Figure 5.2. The numbers in squares represent the decimal code of that cell. A function is represented by placing 1s in cells corresponding to the minterms present or 0s in the cells corresponding to the maxterms present in the function. 5-variable functions are represented on two 4-variable maps, and for 6-variable functions, four 4-variable maps are used. For number of variables  $n$  exceeding 6, it becomes incredibly cumbersome to use the Karnaugh maps.

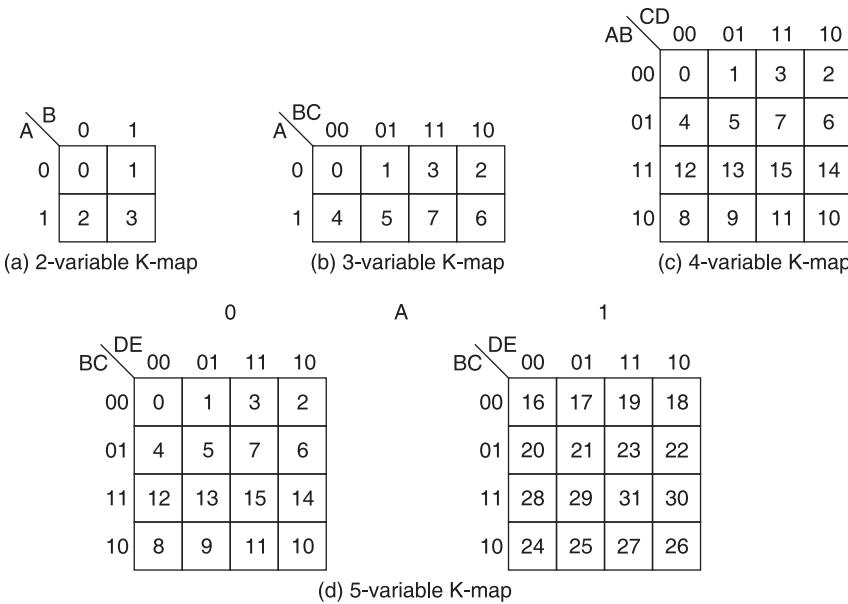
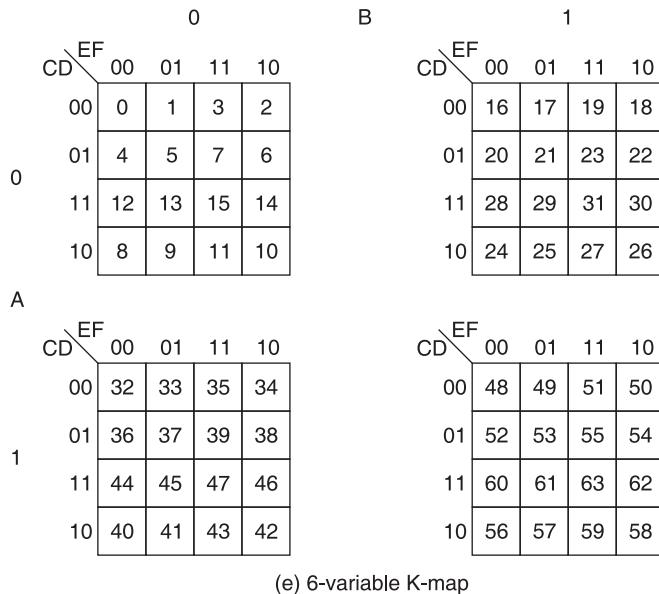


Figure 5.2 Some Karnaugh maps with their row and column designations (Contd.).



(e) 6-variable K-map

**Figure 5.2** Some Karnaugh maps with their row and column designations.

The possible minterms and maxterms of a 2-variable function  $f(A, B)$  are:

$$\begin{array}{llll} m_0 = \bar{A}\bar{B}, & m_1 = \bar{A}B, & m_2 = A\bar{B}, & m_3 = AB. \\ M_0 = (A + B), & M_1 = A + \bar{B}, & M_2 = \bar{A} + B, & M_3 = \bar{A} + \bar{B}. \end{array}$$

The possible minterms and maxterms of a 3-variable function  $f(A, B, C)$  are:

$$\begin{array}{llll} m_0 = \bar{A}\bar{B}\bar{C}, & m_1 = \bar{A}\bar{B}C, & m_2 = \bar{A}B\bar{C}, & m_3 = \bar{A}BC, \\ m_4 = A\bar{B}\bar{C}, & m_5 = A\bar{B}C, & m_6 = AB\bar{C}, & m_7 = ABC. \\ M_0 = A + B + C, & M_1 = A + B + \bar{C}, & M_2 = A + \bar{B} + C, & M_3 = A + \bar{B} + \bar{C}, \\ M_4 = \bar{A} + B + C, & M_5 = \bar{A} + B + \bar{C}, & M_6 = \bar{A} + \bar{B} + C, & M_7 = \bar{A} + \bar{B} + \bar{C}. \end{array}$$

The possible minterms and maxterms of a 4-variable function  $f(A, B, C, D)$  are:

$$\begin{array}{llll} m_0 = \bar{A}\bar{B}\bar{C}\bar{D}, & m_1 = \bar{A}\bar{B}\bar{C}D, & m_2 = \bar{A}\bar{B}CD, \dots & m_{14} = ABC\bar{D}, \\ m_{15} = ABCD. \\ M_0 = A + B + C + D, & M_1 = A + B + C + \bar{D}, & M_2 = A + B + \bar{C} + D, \dots \\ M_{14} = \bar{A} + \bar{B} + \bar{C} + D, & M_{15} = \bar{A} + \bar{B} + \bar{C} + \bar{D}. \end{array}$$

The possible minterms and maxterms of a 5-variable function  $f(A, B, C, D, E)$  are:

$$\begin{array}{llll} m_0 = \bar{A}\bar{B}\bar{C}\bar{D}\bar{E}, & m_1 = \bar{A}\bar{B}\bar{C}\bar{D}E, & m_2 = \bar{A}\bar{B}\bar{C}DE, \dots & m_{30} = ABCDE, \\ m_{31} = ABCDE. \\ M_0 = A + B + C + D + E, & M_1 = A + B + C + D + \bar{E}, \\ M_2 = A + B + C + \bar{D} + E, \dots & M_{30} = \bar{A} + \bar{B} + \bar{C} + \bar{D} + E, \\ M_{31} = \bar{A} + \bar{B} + \bar{C} + \bar{D} + \bar{E}. \end{array}$$

## 198 FUNDAMENTALS OF DIGITAL CIRCUITS

The possible minterms and maxterms of a 6-variable function  $f(A, B, C, D, E, F)$  are:

$$m_0 = \overline{ABC}\overline{D}\overline{E}\overline{F}, \quad m_1 = \overline{ABC}\overline{D}\overline{E}F, \quad m_2 = \overline{ABC}\overline{D}E\overline{F}, \dots \quad m_{62} = ABCDE\overline{F},$$

$$m_{63} = ABCDEF$$

$$M_0 = A + B + C + D + E + F,$$

$$M_1 = A + B + C + D + E + \overline{F},$$

$$M_2 = A + B + C + D + \overline{E} + F, \dots$$

$$M_{62} = \overline{A} + \overline{B} + \overline{C} + \overline{D} + \overline{E} + F,$$

$$M_{63} = \overline{A} + \overline{B} + \overline{C} + \overline{D} + \overline{E} + \overline{F}.$$

### 5.8 EXPANSION OF A BOOLEAN EXPRESSION IN SOP FORM TO THE STANDARD SOP FORM

The following steps are followed for the expansion of a Boolean expression in SOP form to the standard SOP form:

1. Write down all the terms.
2. If one or more variables are missing in any product term, expand that term by multiplying it with the sum of each one of the missing variable and its complement.
3. Drop out the redundant terms.

Also, the given expression can be directly written in terms of its minterms by using the following procedure:

1. Write down all the terms.
2. Put Xs in terms where variables must be inserted to form a minterm.
3. Replace the non-complemented variables by 1s and the complemented variables by 0s, and use all combinations of Xs in terms of 0s and 1s to generate minterms.
4. Drop out all the redundant terms.

### 5.9 EXPANSION OF A BOOLEAN EXPRESSION IN POS FORM TO STANDARD POS FORM

The expansion of a Boolean expression in POS form to the standard POS form is conducted as follows:

1. Write down all the terms.
2. If one or more variables are missing in any sum term, expand that term by adding the products of each of the missing variable and its complement.
3. Drop out the redundant terms.

The given expression can also be written in terms of maxterms by using the following procedure:

1. Write down all the terms.
2. Put Xs in terms wherever variables must be inserted to form a maxterm.
3. Replace the complemented variables by 1s and the non-complemented variables by 0s and use all combinations of Xs in terms of 0s and 1s to generate maxterms.
4. Drop out the redundant terms.

### 5.9.1 Conversion between Canonical Forms

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

$$f(A, B, C) = \sum m(0, 2, 4, 6, 7)$$

This has a complement that can be expressed as

$$\overline{f(A, B, C)} = \sum m(1, 3, 5) = m_1 + m_3 + m_5$$

Now if we take the complement of  $\overline{f}$  by De Morgan's theorem we obtain  $f$  in a different form.

$$f = \overline{(m_1 + m_3 + m_5)} = \overline{m_1} \cdot \overline{m_3} \cdot \overline{m_5} = M_1 M_3 M_5 = \prod M(1, 3, 5)$$

In fact,  $\overline{m_j} = M_j$ .

That is, the maxterm with subscript  $j$  is a complement of the minterm with the same subscript  $j$  and vice versa. To convert one canonical form to another, interchange the symbols  $\Sigma$  and  $\Pi$ , and list those numbers missing from the original form.

**EXAMPLE 5.15** Expand  $\overline{A} + \overline{B}$  to minterms and maxterms.

**Solution**

The given expression is a two-variable function. In the first term  $\overline{A}$ , the variable  $B$  is missing; so, multiply it by  $(B + \overline{B})$ . In the second term  $\overline{B}$ , the variable  $A$  is missing; so, multiply it by  $(A + \overline{A})$ . Therefore,

$$\begin{aligned}\overline{A} + \overline{B} &= \overline{A}(B + \overline{B}) + \overline{B}(A + \overline{A}) \\ &= \overline{AB} + \overline{A}\overline{B} + \overline{BA} + \overline{B}\overline{A} \\ &= \overline{AB} + \overline{A}\overline{B} + A\overline{B} + B\overline{A} \\ &= \overline{AB} + \overline{A}\overline{B} + AB \\ &= 01 + 00 + 10 \\ &= m_1 + m_0 + m_2 \\ &= \sum m(0, 1, 2)\end{aligned}$$

The minterm  $m_3$  is missing in the SOP form. Therefore, the maxterm  $M_3$  will be present in the POS form. Hence the POS form is  $M_3$ , i.e.  $\overline{A} + \overline{B}$ . Also,

$$\begin{aligned}\overline{A} + \overline{B} &= \overline{A} \cdot X + X \cdot \overline{B} \\ &= 0X + X0 \\ &= 00 + 01 + 00 + 10 \\ &= 00 + 01 + 10 \\ &= \sum m(0, 1, 2)\end{aligned}$$

**EXAMPLE 5.16** Expand  $A + B\bar{C} + A\bar{B}\bar{D} + ABCD$  to minterms and maxterms.

**Solution**

The given expression is a four-variable function. In the first term A, the variables B, C, and D are missing. So, multiply it by  $(B + \bar{B})(C + \bar{C})(D + \bar{D})$ . In the second term  $B\bar{C}$ , the variables A and D are missing. So, multiply it by  $(A + \bar{A})(D + \bar{D})$ . In the third term,  $A\bar{B}\bar{D}$ , the variable C is missing. So, multiply it by  $(C + \bar{C})$ . In the fourth term  $ABCD$ , all the variables are present. So, leave it as it is. Therefore,

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

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

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

$$\begin{aligned} \text{or } A + B\bar{C} + A\bar{B}\bar{D} + ABCD &= ABCD + ABC\bar{D} + AB\bar{C}D + AB\bar{C}\bar{D} + A\bar{B}CD + A\bar{B}C\bar{D} \\ &\quad + A\bar{B}\bar{C}D + A\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}B\bar{C}\bar{D} \\ &= m_{15} + m_{14} + m_{13} + m_{12} + m_{11} + m_{10} + m_9 + m_8 + m_5 + m_4 \\ &= \Sigma m(4, 5, 8, 9, 10, 11, 12, 13, 14, 15) \end{aligned}$$

In the SOP form, the minterms 0, 1, 2, 3, 6, and 7 are missing. So in the POS form, the maxterms 0, 1, 2, 3, 6, and 7 will be present. Therefore, the POS form is

$$\prod M(0, 1, 2, 3, 6, 7)$$

Also,

$$\begin{aligned} A &= AXXX = 1XXX \\ &= 1000 + 1001 + 1010 + 1011 + 1100 + 1101 + 1110 + 1111 \\ &= m_8 + m_9 + m_{10} + m_{11} + m_{12} + m_{13} + m_{14} + m_{15} \end{aligned}$$

$$\begin{aligned} B\bar{C} &= XB\bar{C}X = X10X \\ &= 0100 + 0101 + 1100 + 1101 \\ &= m_4 + m_5 + m_{12} + m_{13} \end{aligned}$$

$$A\bar{B}\bar{D} = AB\bar{D} = 11X0 = 1100 + 1110 = m_{12} + m_{14}$$

Dropping out the redundant terms, we get

$$A + B\bar{C} + A\bar{B}\bar{D} + ABCD = \Sigma m(4, 5, 8, 9, 10, 11, 12, 13, 14, 15)$$

**EXAMPLE 5.17** Expand  $A(\bar{B} + A)B$  to maxterms and minterms.

**Solution**

The given expression is a two-variable function in the POS form. The variable B is missing in the first term A. So, add  $B\bar{B}$  to it. The second term contains all the variables. So, leave it as it is. The variable A is missing in the third term B. So, add  $A\bar{A}$  to it. Therefore,

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

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

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

$$\begin{aligned}
 &= (00)(01)(10) \\
 &= M_0 \cdot M_1 \cdot M_2 \\
 &= \prod M(0, 1, 2)
 \end{aligned}$$

The maxterm  $M_3$  is missing in the POS form. So, the SOP form will contain only the minterm  $m_3$ .

Also,

$$\begin{aligned}
 A \rightarrow 0X &= (00)(01) = M_0 \cdot M_1 \\
 (A + \bar{B}) \rightarrow (01) &= M_1 \\
 B \rightarrow X0 &= (00)(10) = M_0 \cdot M_2
 \end{aligned}$$

Therefore,

$$A(A + \bar{B})B = \prod M(0, 1, 2)$$

**EXAMPLE 5.18** Expand  $A(\bar{A} + B)(\bar{A} + B + \bar{C})$  to maxterms and minterms.

**Solution**

The given expression is a three-variable function in the POS form. The variables B and C are missing in the first term A. So, add  $B\bar{B}$  and  $C\bar{C}$  to it. The variable C is missing in the second term  $(\bar{A} + B)$ . So, add  $C\bar{C}$  to it. The third term  $(\bar{A} + B + \bar{C})$  contains all the three variables. So, leave it as it is. Therefore,

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

Therefore,

$$\begin{aligned}
 A(\bar{A} + B)(\bar{A} + B + \bar{C}) &= (A + B + C)(A + B + \bar{C})(A + \bar{B} + C)(A + \bar{B} + \bar{C})(\bar{A} + B + C)(\bar{A} + B + \bar{C}) \\
 &= (000)(001)(010)(011)(100)(101) \\
 &= M_0 \cdot M_1 \cdot M_2 \cdot M_3 \cdot M_4 \cdot M_5 \\
 &= \prod M(0, 1, 2, 3, 4, 5)
 \end{aligned}$$

The maxterms  $M_6$  and  $M_7$  are missing in the POS form. So, the SOP form will contain the minterms 6 and 7. Therefore, the given expression in the SOP form is  $\Sigma m(6, 7)$ .

Also,

$$\begin{aligned}
 A \rightarrow 0XX &= (000)(001)(010)(011) \\
 &= M_0 \cdot M_1 \cdot M_2 \cdot M_3 \\
 \bar{A} + B \rightarrow 10X &= (100)(101) = M_4 \cdot M_5 \\
 \bar{A} + B + \bar{C} &= 101 = M_5
 \end{aligned}$$

Therefore,

$$A(\bar{A} + B)(\bar{A} + B + \bar{C}) = \prod M(0, 1, 2, 3, 4, 5)$$

**EXAMPLE 5.19** Write the algebraic terms of a four-variable expression having the following minterms.

- (a)  $m_0$       (b)  $m_5$       (c)  $m_9$       (d)  $m_{14}$

**Solution**

Given minterm	$m_0$	$m_5$	$m_9$	$m_{14}$
Binary form	0000	0101	1001	1110
Product term	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$A\bar{B}\bar{C}\bar{D}$	$AB\bar{C}\bar{D}$

**EXAMPLE 5.20** Write the algebraic terms of a four-variable expression having the following maxterms.

- (a)  $M_3$       (b)  $M_9$       (c)  $M_{11}$       (d)  $M_{14}$

**Solution**

Given maxterm	$M_3$	$M_9$	$M_{11}$	$M_{14}$
Binary form	0011	1001	1011	1110
Sum term	$A + B + \bar{C} + \bar{D}$	$\bar{A} + B + C + \bar{D}$	$\bar{A} + B + \bar{C} + \bar{D}$	$\bar{A} + \bar{B} + \bar{C} + D$

## 5.10 COMPUTATION OF TOTAL GATE INPUTS

The total number of gate inputs required to realize a Boolean expression is computed as follows.

If the expression is in the SOP form, count the number of AND inputs and the number of AND gates feeding the OR gate. If the expression is in the POS form, count the number of OR inputs and the number of OR gates feeding the AND gate. If it is in hybrid form, count the gate inputs and the gates feeding other gates.

The cost of implementing a circuit is roughly proportional to the number of gate inputs required.

**EXAMPLE 5.21** How many gate inputs are required to realize the following expressions?

- (a)  $f_1 = ABC + A\bar{B}CD + E\bar{F} + AD$   
(b)  $f_2 = A(B + C + \bar{D})(\bar{B} + C + \bar{E})(A + \bar{B} + C + E)$

**Solution**

(a) Write the expression	$ABC + A\bar{B}CD + E\bar{F} + AD$
Count the AND inputs	$3 + 4 + 2 + 2 = 11$
Count the AND gates feeding the OR gate	$1 + 1 + 1 + 1 = 4$
Total gate inputs	$= 15$
(b) Write the expression	$A \cdot (B + C + \bar{D}) \cdot (\bar{B} + C + \bar{E}) \cdot (A + \bar{B} + C + E)$
Count the OR inputs	$0 + 3 + 3 + 4 = 10$
Count the OR gates feeding the AND gate	$1 + 1 + 1 + 1 = 4$
Total gate inputs	$= 14$

## 5.11 BOOLEAN EXPRESSIONS AND LOGIC DIAGRAMS

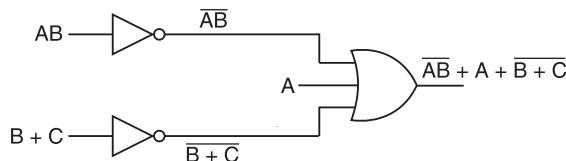
Boolean expressions can be realized as hardware using logic gates. Conversely, hardware can be translated into Boolean expressions for the analysis of existing circuits.

### 5.11.1 Converting Boolean Expressions to Logic

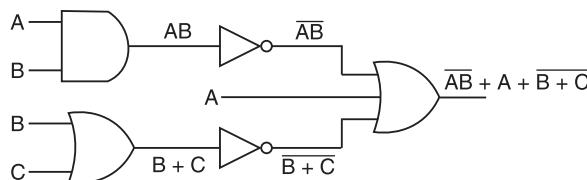
The easiest way to convert a Boolean expression to a logic circuit is to start with the output and work towards the input. Assume that the expression  $\overline{AB} + A + \overline{B+C}$  is to be realized using AOI logic. Start with the final expression  $\overline{AB} + A + \overline{B+C}$ . Since it is a summation of three terms, it must be the output of a three-input OR gate. So, draw an OR gate with three inputs as shown below.



$\overline{AB}$  must be the output of an inverter whose input is  $AB$ , and  $\overline{B+C}$  must be the output of an inverter whose input is  $B+C$ . So, we introduce two inverters as shown below.

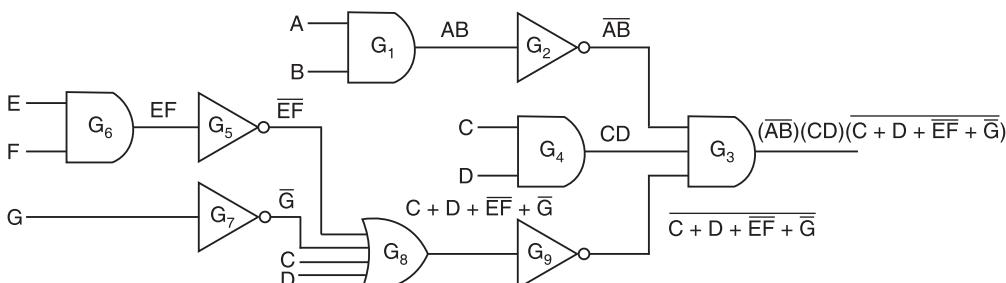


Now  $AB$  must be the output of a two-input AND gate whose inputs are  $A$  and  $B$ . And  $B+C$  must be the output of a two-input OR gate whose inputs are  $B$  and  $C$ . So, we introduce an AND gate and an OR gate as shown below.



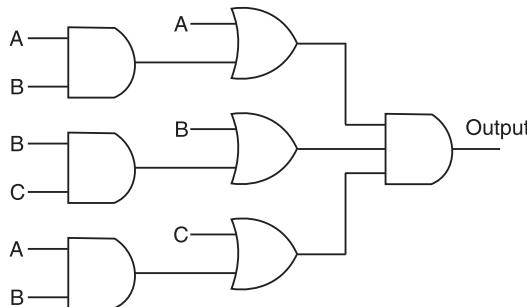
### 5.11.2 Converting Logic to Boolean Expressions

To convert logic to algebra, start with the input signals and develop the terms of the Boolean expression until the output is reached. Consider the logic diagram shown below.



First label all the inputs. The signals A and B feed the AND gate  $G_1$ . The output of this gate is, therefore,  $AB$ . This is the input to NOT gate  $G_2$ . The output of the NOT gate  $G_2$  will, therefore, be  $\overline{AB}$ . C and D are the inputs to AND gate  $G_4$ . The output of this gate is, therefore,  $CD$ . E and F are the inputs to AND gate  $G_6$ . The output of this gate is, therefore,  $EF$ . This signal EF is the input to inverter  $G_5$ . Its output is, therefore,  $\overline{EF}$ . The input to inverter  $G_7$  is G. So, its output is  $\overline{G}$ . Now C, D,  $\overline{EF}$ , and  $\overline{G}$  are the inputs to OR gate  $G_8$ . The output of the OR gate  $G_8$  will, therefore, be  $(C + D + \overline{EF} + \overline{G})$ . This is the input to inverter  $G_9$ . The output of  $G_9$  is, therefore,  $(C + D + \overline{EF} + \overline{G})$ . The inputs to the AND gate  $G_3$  are,  $\overline{AB}$ , CD and  $(C + D + \overline{EF} + \overline{G})$ . The output of  $G_3$ , which is also the output of the logic circuit is, therefore, equal to  $(\overline{AB}) \cdot (CD) \cdot (C + D + \overline{EF} + \overline{G})$ .

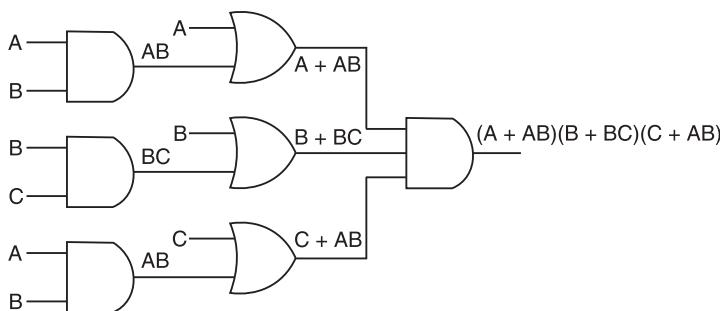
**EXAMPLE 5.22** Write the Boolean expression for the logic diagram given below and simplify it as much as possible and draw the logic diagram that implements the simplified expression.



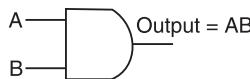
### Solution

Starting from the input side and writing the expressions for the outputs of the individual gates, we can easily show that the

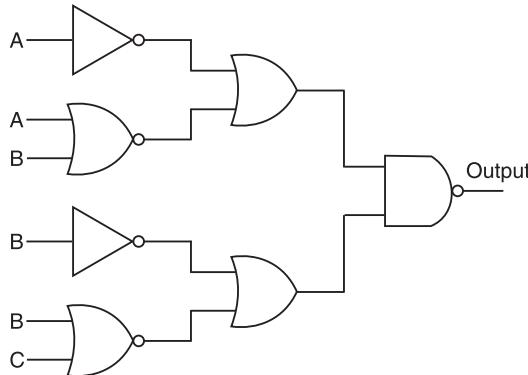
$$\begin{aligned}
 \text{Output} &= (A + AB)(B + BC)(C + AB) \\
 &= A(1 + B)B(1 + C)(C + AB) \\
 &= AB(C + AB) \\
 &= ABC + AB \\
 &= AB(1 + C) \\
 &= AB
 \end{aligned}$$



The logic diagram to realize the simplified expression is just an AND gate shown below.



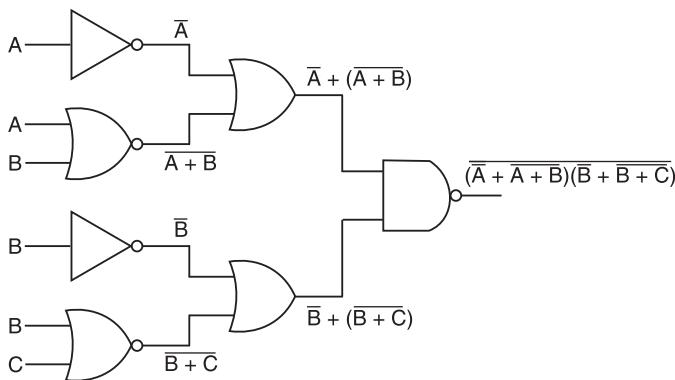
**EXAMPLE 5.23** Draw the simplest possible logic diagram that implements the output of the logic diagram shown below.



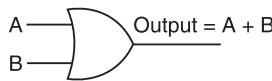
### Solution

Starting from the input side and writing the expressions for the outputs of the individual gates as shown in the diagram below, we have

$$\begin{aligned}
 \text{Output} &= \overline{(\bar{A} + A + B)} \bar{B} + \overline{B + C} \\
 &= \overline{\bar{A} + A + B} + \overline{B + C} \\
 &= \bar{A} \cdot \overline{A + B} + \bar{B} \cdot \overline{B + C} \\
 &= A \cdot (A + B) + B \cdot (B + C) \\
 &= AA + AB + B + BC \\
 &= A + AB + B + BC \\
 &= A(1 + B) + B(1 + C) \\
 &= A + B
 \end{aligned}$$

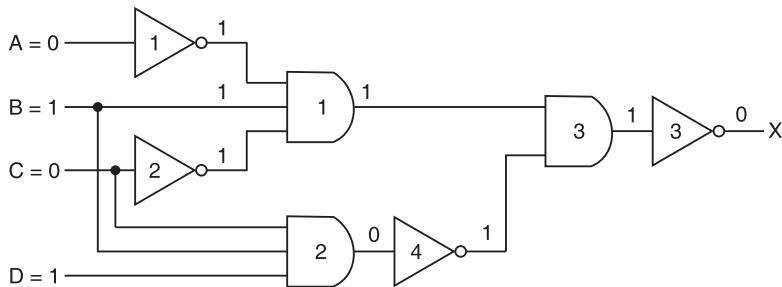


The logic diagram to implement the simplified expression is shown below.



## 5.12 DETERMINATION OF OUTPUT LEVEL FROM THE DIAGRAM

The output logic level for the given input levels can be determined by obtaining the Boolean expression for the output of the circuit and then substituting the values of the inputs in that expression. This can also be determined directly from the circuit diagram by writing down the output level of each gate starting from the input side till the final output is reached. This technique is often used for troubleshooting or testing of a logic system. This procedure is self-evident from the illustration given below.

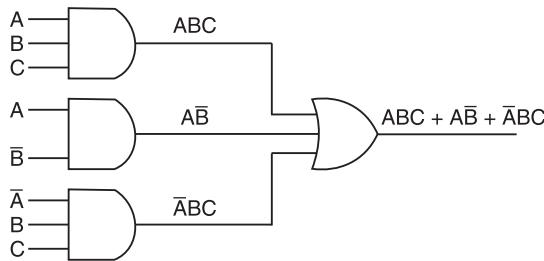


The same result can be obtained by writing the output expression and substituting the values of the inputs in it. Thus,

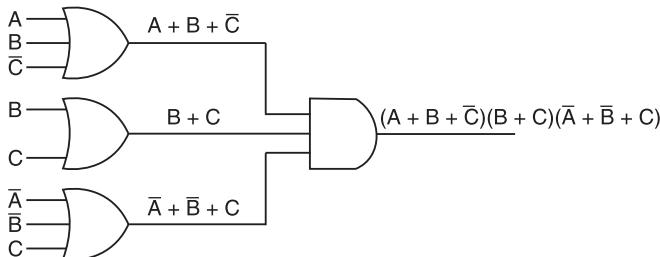
$$\begin{aligned} X &= \overline{(\bar{A}\bar{B}\bar{C})(\bar{B}\bar{C}\bar{D})} \\ &= \overline{(\bar{0} \cdot 1 \cdot \bar{0})(1 \cdot \bar{0} \cdot 1)} \\ &= \overline{1 \cdot \bar{0}} \\ &= 0 \end{aligned}$$

## 5.13 CONVERTING AND/OR/INVERT LOGIC TO NAND/NOR LOGIC

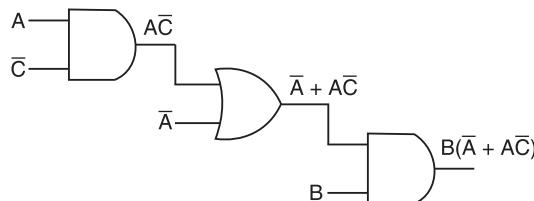
In the design of digital circuits, the minimal Boolean expressions are usually obtained in SOP (sum-of-products) form or POS (product-of-sums) form. Sometimes the minimal expressions may also be expressed in hybrid form. The SOP form is implemented using a group of AND gates whose outputs are ORed and the POS form is implemented using a group of OR gates whose outputs are ANDed. The hybrid form is a combination of both the SOP and POS forms. Sometimes the complement of a function is to be implemented. So, the designed circuit can be implemented using AND and OR gates only called the A/O logic, or using AND/OR/NOT gates called the AOI logic. In the realization of Boolean expressions, the variable and its complement are assumed to be available. For example, the SOP expression  $ABC + \bar{A}\bar{B} + \bar{A}\bar{B}C$  can be implemented in A/O logic as shown below.



The POS expression  $(A + B + \bar{C})(B + C)(\bar{A} + \bar{B} + C)$  can be implemented using OR and AND gates as shown below.



The expression  $AB\bar{C} + A\bar{B} [= B(\bar{A} + A\bar{C})]$  can be implemented in hybrid form as shown below.



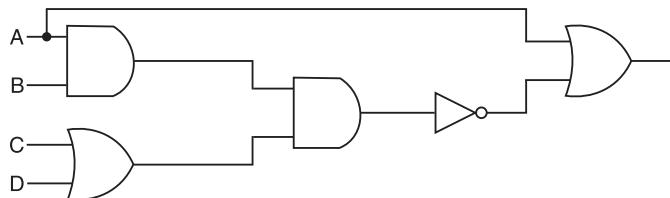
Hybrid logic reduces the number of gate inputs required for realization (from 7 to 6 in this case), but results in multilevel logic. Different inputs pass through different numbers of gates to reach the output. It leads to non-uniform propagation delay between input and output and may give rise to logic race. The SOP and POS realizations give rise to two-level logic. The two-level logic provides uniform time delay between input and output, because each input signal has to pass through two gates to reach the output. So, it does not suffer from the problem of logic race.

Since NAND logic and NOR logic are universal logic systems, digital circuits which are first computed and converted to AOI logic may then be converted to either NAND logic or NOR logic depending on the choice. The procedure is given as follows:

1. Draw the circuit in AOI logic.
2. If NAND hardware is chosen, add a circle at the output of each AND gate and at the inputs to all the OR gates.
3. If NOR hardware is chosen, add a circle at the output of each OR gate and at the inputs to all the AND gates.
4. Add or subtract an inverter on each line that received a circle in steps 2 or 3 so that the polarity of signals on those lines remains unchanged from that of the original diagram.

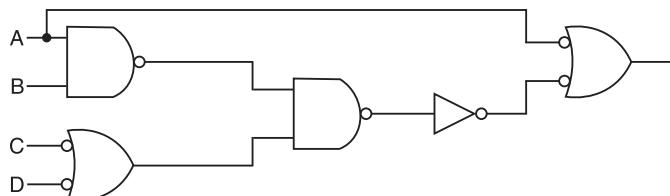
5. Replace bubbled OR by NAND and bubbled AND by NOR.
6. Eliminate double inversions.

**EXAMPLE 5.24** Convert the following AOI logic circuit to (a) NAND logic, and (b) NOR logic.

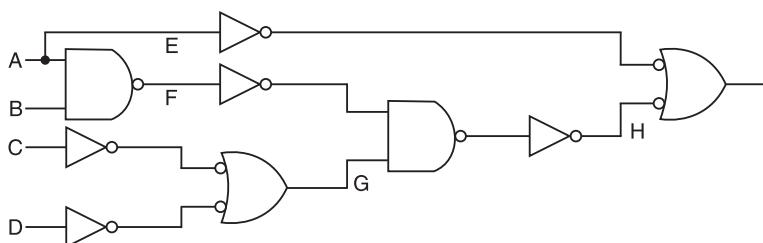


**Solution**

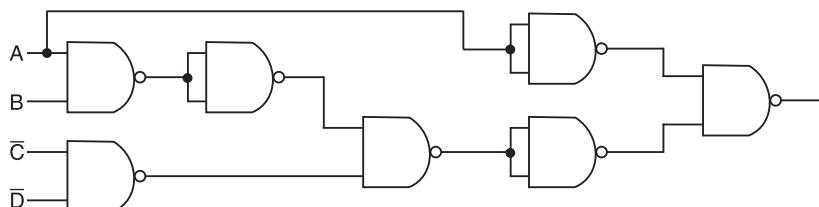
(a) NAND logic: Put a circle at the output of each AND gate and at the inputs to all OR gates as shown below.



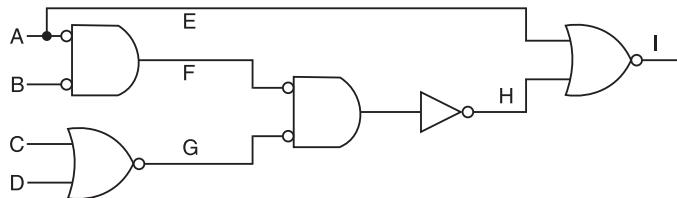
Add an inverter to each of the lines E, F, C, D that received only one circle in the previous step so that the polarity of these lines remains unchanged. Inverters in lines C and D can be removed, if C and D are replaced by  $\bar{C}$  and  $\bar{D}$ . Line H received two circles. So, no change is required.



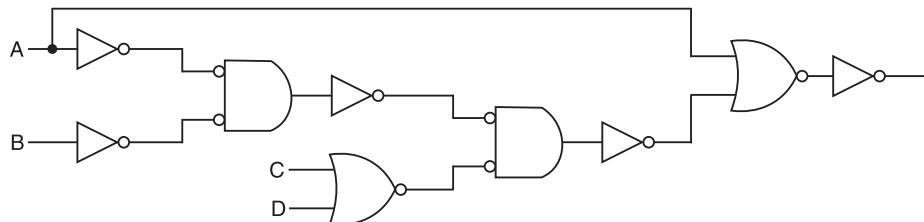
Replace bubbled OR gates and NOT gates by NAND gates. Using only NAND gates, the logic circuit can now be drawn as shown below.



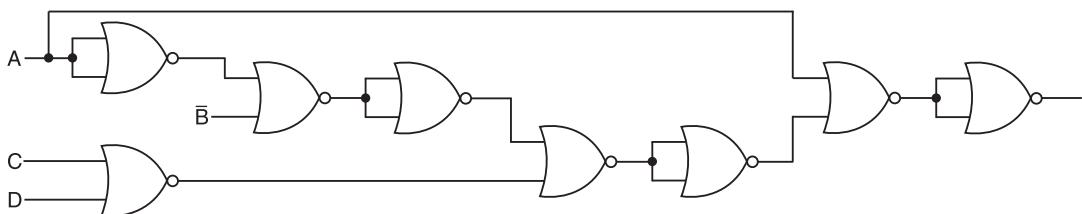
(b) NOR logic: Put a circle at the output of each OR gate and at the inputs to all AND gates as shown below.



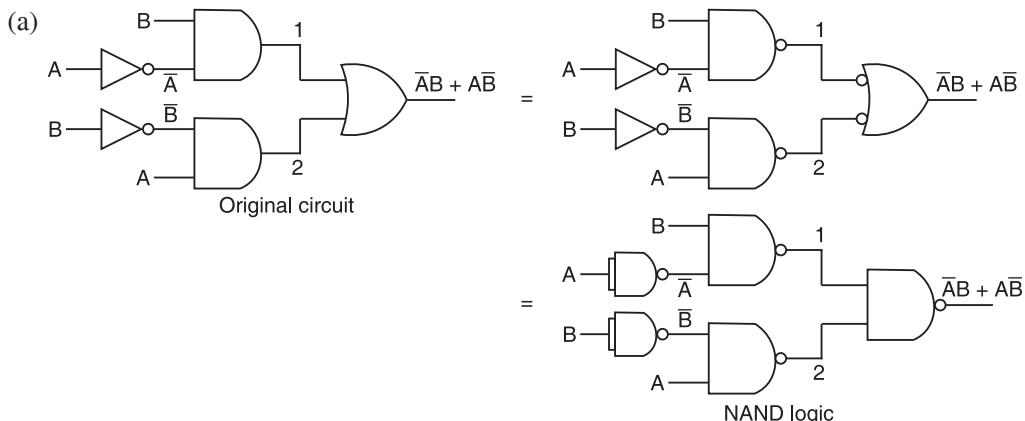
Add an inverter in each of the lines A, B, F, and I that received only one circle in the previous step, so that the polarity of these lines remains unchanged. Line G received two circles. So, no change is required.

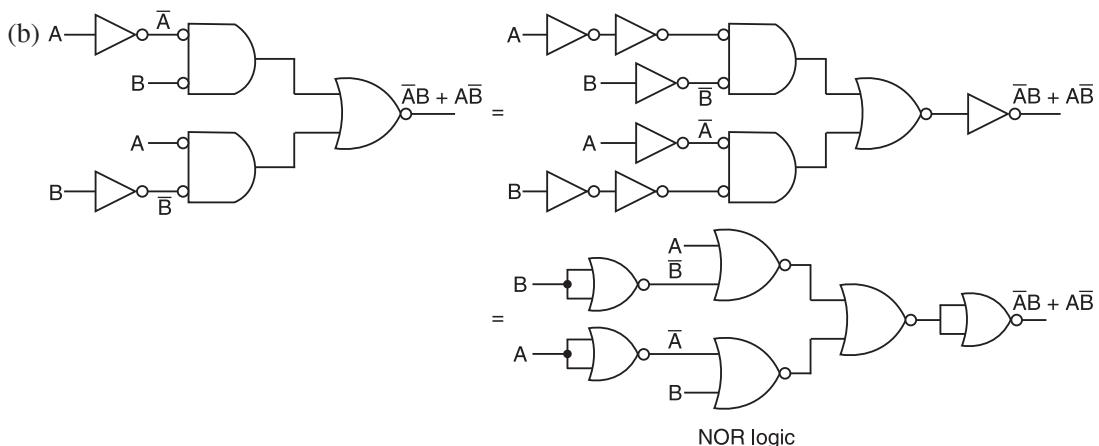


Replace bubbled AND gates and NOT gates by NOR gates. Using only NOR gates, the logic circuit can now be drawn as shown below. Note that the inverter in line B has been removed assuming that  $\bar{B}$  is available.

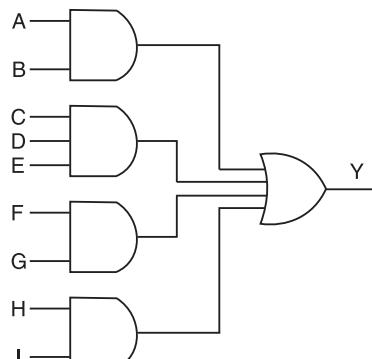


Now study the conversion of the following AOI circuit to (a) NAND logic, and (b) NOR logic.

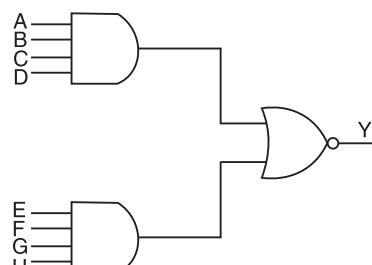




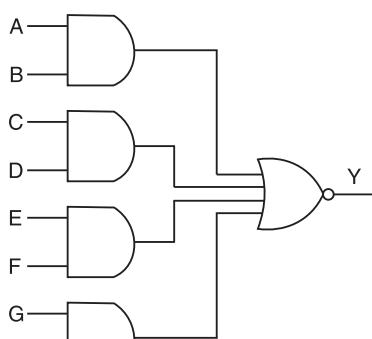
The A/O gates and AOI gates are available in IC form. A circuit having  $n$  AND gates is said to be  $n$ -wide. The A/O gates in which an additional variable or a combination of variables can be included in the logic operation are called *expandable gates*. Some AOI gates and expandable A/O gates available in IC form are shown in Figure 5.3.



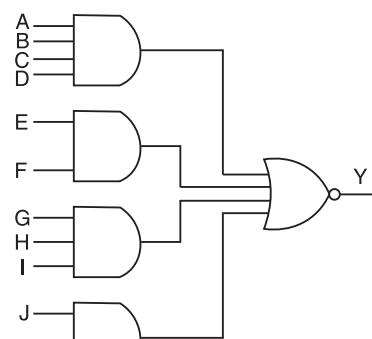
(a) Expandable 4-wide A/O gate  
 $Y = AB + CDE + FG + HI$



(b) 2-wide 4-input AOI gate  
 $Y = \overline{ABCD} + \overline{EFGH}$



(c) 4-wide, 2-input AOI gate  
 $Y = \overline{AB} + \overline{CD} + \overline{EF} + \overline{GH}$



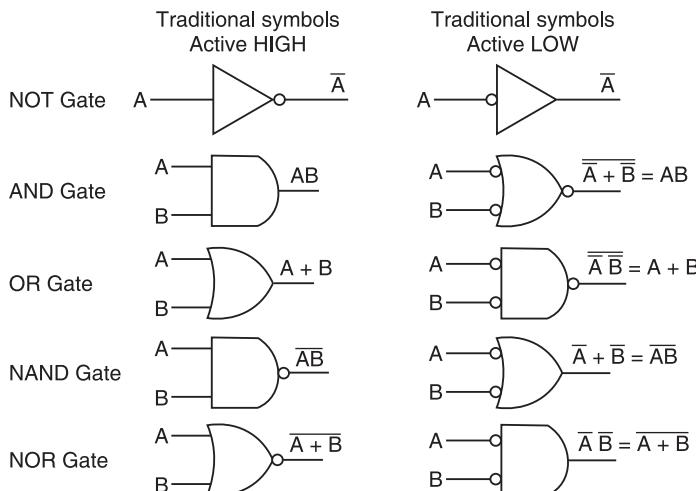
(d) 4-2-3-2 AOI gate  
 $Y = \overline{ABCD} + \overline{EF} + \overline{GHI} + \overline{JK}$

**Figure 5.3** Some AOI gates and expandable A/O gates available in IC form.

### 5.13.1 Active-LOW Notation

The logic gates and circuits discussed so far are called active-HIGH gates and circuits. It means that the action represented or initiated by a variable occurs when it is equal to a 1. In some digital circuits, action occurs when one of several inputs is a 0. Such an input is said to be an active-LOW input. On logic diagrams, placing the inversion bubble at the point where the input signal is connected, shows an active-LOW input. We may regard the bubble itself as an inverter. Thus, we can think of a 0 occurring on the external signal line as being inverted and producing a 1 internal to the device and vice versa.

The logic symbols of active-HIGH logic gates and their active-LOW versions, are shown in Figure 5.4. Note that the NAND gate is equivalent to an active-LOW input OR gate. The NOR gate is equivalent to an active-LOW input AND gate. The AND gate is equivalent to active-LOW NOR gate and the OR gate is equivalent to active-LOW NAND gate. The active-LOW symbols (alternate symbols) can be obtained from active-HIGH symbols (standard symbols) by inverting each of the inputs and outputs and changing the gates—OR to AND and AND to OR.



**Figure 5.4** Logic symbols of active-HIGH logic gates and their active-LOW versions.

**Asserted levels:** The logic signals can be active-LOW signals or active-HIGH signals. Normally the active-HIGH signals are represented by variables with no bar over them such as CLOCK, A, MEM, etc., whereas the active-LOW signals are represented by variables with bar over them such as  $\bar{X}$ , CLR, MEM, etc. The bar simply emphasizes that a particular signal is active-LOW. When a signal is in its active state, it is said to be *asserted*. When it is not in its active state, i.e. when it is inactive, it is said to be *unasserted*. The terms ‘asserted’ and ‘unasserted’ are synonymous with ‘active’ and ‘inactive’, respectively.

**Negative logic:** The assertion level refers to the signal level necessary to cause an event to occur. Till now, we have assumed that logic 1 is +5 V and logic 0 is 0 V. In this notation, events occur when inputs are +5 V, i.e. the assertion level is a 1. This is called *positive logic*, because a 1 is more positive than a 0. In some systems, it is convenient to define the ground level as a 1 and +5 V as a 0. That means that the assertion level is a 0. That is, the level required to do something is a

0. This is called *negative logic*, because a 1 is more negative than a 0. In a negative logic system, the more positive of the two voltage levels is represented by a logic 0. An AND gate in the positive logic system becomes an OR gate in the negative logic system and vice versa.

Active-LOW bubbles can be very useful when analyzing logic diagrams, because they can be placed in such a way that they effectively cancel out one another. This eliminates the necessity for writing numerous inversion bars over compound logic expressions.

The circuit shown in Figure 5.5 implements the expression  $D(\bar{C} + A + B)$ . We can use Boolean algebra to show that this expression is equivalent to  $DC(A + B)$ .

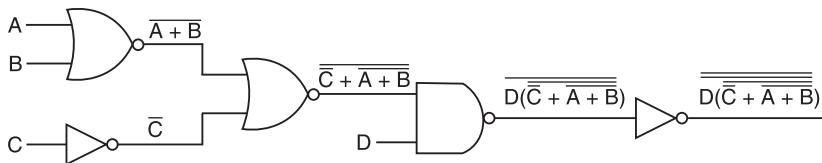
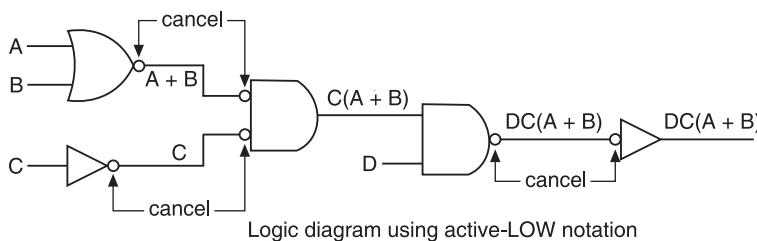


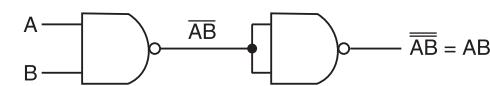
Figure 5.5 Conventional logic diagram.

The same implementation with active-LOW equivalents by replacing one NOR gate and one inverter is shown below. Since consequent inversion bubbles cancel out as shown, it is readily apparent in this diagram that the output is  $DC(A + B)$ .



**NAND and NOR gates as universal gates:** We know that AND, OR, and NOT gates are the basic building blocks of a digital computer. They are called the basic gates. Any digital circuit of any complexity can be built using only these three gates. A universal gate is a gate which alone can be used to build any logic circuit. So, to show that the NAND gate and the NOR gate are universal gates, we have to show that all the three basic logic gates can be realized using only NAND gates or using only NOR gates. The diagrams given in Figure 5.6 show the realization of AND, OR, and NOT functions using either only NAND gates or only NOR gates.

(a) Realization of AND function using only NAND gates  
 $AB = \overline{\overline{AB}} = \overline{(AB)}$



(b) Realization of OR function using only NAND gates  
 $A + B = \overline{\overline{A + B}} = \overline{\overline{A} \cdot \overline{B}}$

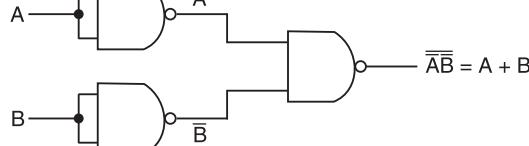
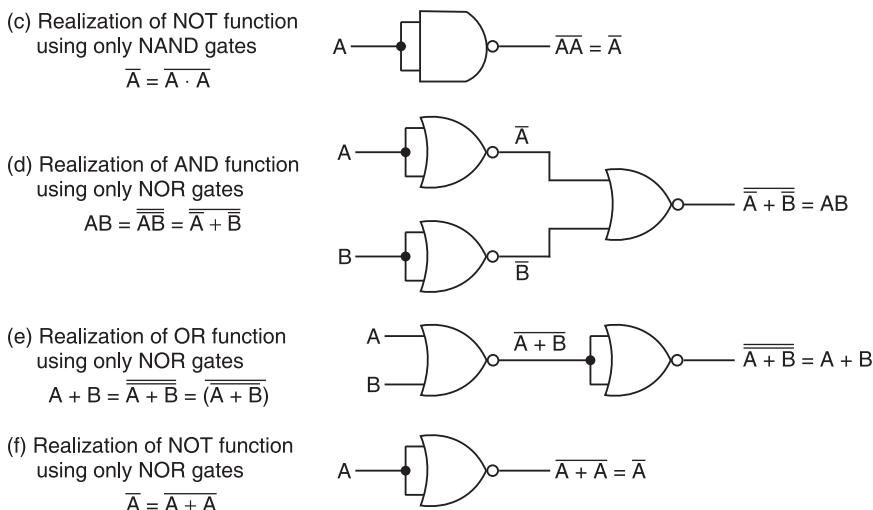


Figure 5.6 Diagrams showing the realization of AND, OR and NOT functions using either only NAND gates or only NOR gates (Contd.)...



**Figure 5.6** Diagrams showing the realization of AND, OR and NOT functions using either only NAND gates or only NOR gates.

**EXAMPLE 5.25** Find the values of the two-valued variables A, B, C, and D by solving the set of simultaneous equations.

$$\bar{A} + AB = 0$$

$$AB = AC$$

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

**Solution**

Given

$$\bar{A} + AB = 0$$

or

$$(\bar{A} + A)(\bar{A} + B) = 0$$

or

$$(1)(\bar{A} + B) = 0$$

or

$$\bar{A} + B = 0$$

i.e.  $\bar{A}$  must be 0 and B must be 0. Therefore, A = 1 and B = 0.

Also

$$AB = AC$$

or

$$1 \cdot 0 = 1 \cdot C$$

Therefore

$$C = 0$$

Also,

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

or

$$1 \cdot 0 + 1 \cdot 1 + 0 \cdot D = 1 \cdot D$$

or

$$1 = D$$

Therefore, the values of A, B, C, and D are A = 1, B = 0, C = 0, and D = 1.

**EXAMPLE 5.26** Prove that

(a) If  $A \oplus B = 0$ , then  $A = B$

(b)  $A \oplus B = \bar{A} \oplus \bar{B}$

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

(d)  $0 \oplus A = A$

(e)  $1 \oplus A = \bar{A}$

(f)  $A \oplus A = 0$

(g)  $A \oplus \bar{A} = 1$

**Solution**

(a) The X-OR gate is an anti-coincidence gate. Its output is 0 when both the inputs are equal.

Therefore, if  $A \oplus B = 0$ , then A must be equal to B.

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

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

∴  $A \oplus B = \bar{A} \oplus \bar{B}$

(c)  $\overline{A \oplus B} = \bar{\bar{A}} \cdot B + \bar{A} \cdot \bar{B} = AB + \bar{A}\bar{B} = \overline{A \oplus B}$

$$A \oplus \bar{B} = \bar{A} \cdot \bar{B} + A \cdot \bar{\bar{B}} = \bar{A}\bar{B} + AB = \overline{A \oplus B}$$

∴  $\overline{A \oplus B} = \bar{A} \oplus B = A \oplus \bar{B}$

(d)  $0 \oplus A = 0 \cdot \bar{A} + \bar{0} \cdot A = 0 + 1 \cdot A = 0 + A = A$

(e)  $1 \oplus A = 1 \cdot \bar{A} + \bar{1} \cdot A = \bar{A} + 0 \cdot A = \bar{A} + 0 = \bar{A}$

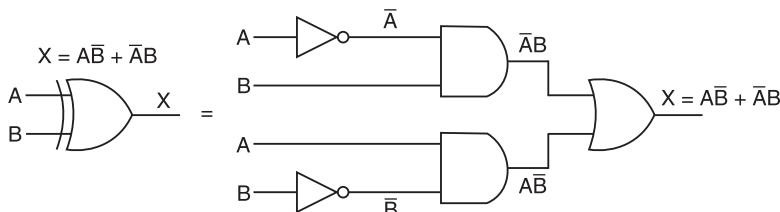
(f)  $A \oplus A = A \cdot \bar{A} + \bar{A} \cdot A = 0 + 0 = 0$

(g)  $A \oplus \bar{A} = A \cdot \bar{A} + \bar{A} \cdot \bar{A} = A \cdot A + \bar{A} \cdot \bar{A} = A + \bar{A} = 1$

**EXAMPLE 5.27** Realize the X-OR function using (a) AOI logic, (b) NAND logic, and (c) NOR logic.

**Solution**

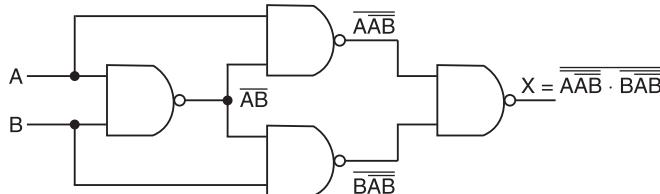
(a) Using AOI logic:



(b) Using NAND logic:

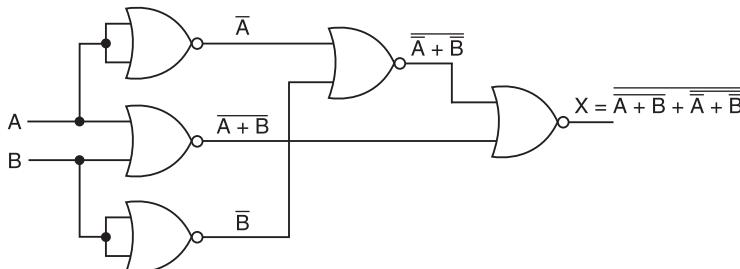
$$\begin{aligned} X &= A\bar{B} + \bar{A}B \\ &= \bar{A}\bar{B} + A\bar{B} + \bar{A}B + BB \\ &= A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B}) \end{aligned}$$

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



(c) Using NOR logic:

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



## 5.14 MISCELLANEOUS EXAMPLES

**EXAMPLE 5.28** Obtain the duals of the following functions:

- |   |  |
|---|--|
| (a) $\overline{AB} + \overline{ABC} + \overline{ABCD} + \overline{ABCDE}$                     | (b) $ABEF + AB\overline{E}\overline{F} + \overline{AB}BEF$   |
| (c) $\overline{XYZ} + \overline{XY}\overline{Z} + X\overline{Y}\overline{Z} + X\overline{YZ}$ | (d) $\overline{XYZ} + X\overline{Y}\overline{Z} + XYZ + XY\overline{Z}$  |
| (e) $\overline{XZ} + \overline{XY} + X\overline{Y}\overline{Z} + YZ$                          | (f) $\overline{X}\overline{Y}\overline{Z} + \overline{XY}\overline{Z} + X\overline{Y}\overline{Z} + X\overline{YZ} + XY\overline{Z}$ |
| (g) $\overline{ABC} + \overline{ABC} + \overline{ABC} + ABC$                                  | (h) $AB + \overline{AC} + \overline{ABC}$  |

**Solution**

To obtain the dual of an expression, change the ORs to ANDs, ANDs to ORs, 0s to 1s, 1s to 0s but do not complement the variables. Based on this rule the duals of the above functions are as follows.

- (a)  $(\overline{A} + B)(\overline{A} + B + \overline{C})(\overline{A} + B + C + D)(\overline{A} + B + \overline{C} + \overline{D} + E)$

- (b)  $(A + B + E + F)(A + B + \bar{E} + \bar{F})(\bar{A} + \bar{B} + E + F)$   
 (c)  $(\bar{X} + Y + Z)(\bar{X} + Y + \bar{Z})(X + \bar{Y} + \bar{Z})(X + \bar{Y} + Z)$   
 (d)  $(\bar{X} + Y + Z)(X + \bar{Y} + \bar{Z})(X + Y + Z)(X + Y + \bar{Z})$   
 (e)  $(\bar{X} + Z)(\bar{X} + Y)(X + \bar{Y} + Z)(Y + Z)$   
 (f)  $(\bar{X} + \bar{Y} + \bar{Z})(\bar{X} + Y + \bar{Z})(X + \bar{Y} + \bar{Z})(X + \bar{Y} + Z)(X + Y + \bar{Z})$   
 (g)  $(\bar{A} + \bar{B} + \bar{C})(\bar{A} + B + \bar{C})(A + \bar{B} + \bar{C})(A + B + \bar{C})$   
 (h)  $(A + B)(A + \bar{C})(A + \bar{B} + C)$

**EXAMPLE 5.29** Find the complements of the following expressions.

- |   |  |
|---|--|
| (a) $AB + A(B + C) + \bar{B}(B + D)$  | (b) $A + B + \bar{A}\bar{B}C$              |
| (c) $\bar{A}B + A\bar{B}C + \bar{A}BCD + \bar{A}\bar{B}CDE$                 | (d) $ABEF + AB\bar{E}\bar{F} + \bar{A}BEF$ |
| (e) $\bar{B}\bar{C}D + (\bar{B} + C + D) + \bar{B}\bar{C}\bar{D}E$          | (f) $AB + \bar{A}\bar{C} + A\bar{B}C$      |
| (g) $(A\bar{B} + A\bar{C})(BC + B\bar{C})(ABC)$                             | (h) $A\bar{B}C + \bar{A}BC + ABC$          |
| (i) $(\bar{A}\bar{B}\bar{C})(\bar{A} + B + C)$                              | (j) $A + \bar{B}C (A + B + \bar{C})$       |
| (k) $\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C} + ABC$ |  |

**Solution**

To obtain the complement of an expression change the ANDs to ORs, ORs to ANDs, 0s to 1s, 1s to 0s and complement each variable. Based on this rule the complements of the above functions are as follows.

- (a)  $(\bar{A} + \bar{B}) \cdot (\bar{A} + \bar{B} \cdot \bar{C}) \cdot (\bar{B} + \bar{B}\bar{D}) = (\bar{A} + \bar{B}) \cdot (\bar{A} + \bar{B} \cdot \bar{C}) \cdot (B + \bar{B}\bar{D})$   
 (b)  $\bar{A} \cdot \bar{B} \cdot (\bar{A} + \bar{B} + \bar{C}) = (\bar{A}) \cdot (\bar{B}) \cdot (A + B + \bar{C})$   
 (c)  $(\bar{A} + \bar{B}) \cdot (\bar{A} + BC) \cdot (\bar{A} + \bar{B} + \bar{C} + \bar{D}) \cdot (\bar{A} + \bar{B}\bar{C} + \bar{D} + \bar{E})$   
 $= (A + \bar{B}) \cdot (\bar{A} + BC) \cdot (A + \bar{B} + \bar{C} + \bar{D}) \cdot (A + BC + D + \bar{E})$   
 (d)  $(\bar{A} + \bar{B} + \bar{E} + \bar{F}) \cdot (\bar{A} + \bar{B} + \bar{E} + \bar{F}) \cdot (\bar{A} + \bar{B} + \bar{E} + \bar{F})$   
 $= (\bar{A} + \bar{B} + \bar{E} + \bar{F}) \cdot (\bar{A} + \bar{B} + E + F) \cdot (A + B + \bar{E} + \bar{F})$   
 (e)  $(\bar{B} + \bar{C} + \bar{D}) \cdot (\bar{B} + \bar{C} + \bar{D}) \cdot (\bar{B} + \bar{C} + \bar{D} + \bar{E})$   
 $= (B + C + \bar{D}) \cdot (B + C + D) \cdot (B + C + D + \bar{E})$   
 (f)  $(\bar{A} + \bar{B}) \cdot (AC) \cdot (\bar{A} + B + \bar{C})$   
 (g)  $(\bar{A} + \bar{B}) \cdot (\bar{A} + \bar{C}) + (\bar{B} + \bar{C}) \cdot (\bar{B} + \bar{C}) + (\bar{A} + \bar{B} + \bar{C})$   
 $= (\bar{A} + B) \cdot (\bar{A} + C) + (\bar{B} + \bar{C}) \cdot (\bar{B} + C) + (\bar{A} + \bar{B} + \bar{C})$   
 (h)  $(\bar{A} + \bar{B} + \bar{C}) \cdot (\bar{A} + \bar{B} + \bar{C}) \cdot (\bar{A} + \bar{B} + \bar{C})$   
 (i)  $(\bar{A}\bar{B}\bar{C}) + (\bar{A} + B + C) = (ABC) + (A + B + C)$   
 (j)  $\bar{A} \cdot (\bar{B} + \bar{C}) + (\bar{A}\bar{B}\bar{C}) = \bar{A} \cdot (B + \bar{C}) + (\bar{A}\bar{B}C)$

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

**EXAMPLE 5.30** Find the complement and the dual of the function given below and then reduce it to a minimum number of literals in each case.  $f = [(\bar{a}\bar{b})a][(\bar{a}\bar{b})b]$ .

**Solution**

$$\text{Complement of } [(\bar{a}\bar{b})a][(\bar{a}\bar{b})b] = \bar{\bar{ab}} + \bar{a} + \bar{\bar{ab}} + \bar{b} = ab + \bar{a} + ab + \bar{b} \\ (\text{Reduction}) ab + \bar{a} + ab + \bar{b} = \bar{a} + \bar{b} + ab = (\bar{a} + a)(\bar{a} + b) + \bar{b} \\ = \bar{a} + b + \bar{b} = \bar{a} + 1 = 1$$

$$\text{Dual of } [(\bar{a}\bar{b})a][(\bar{a}\bar{b})b] = (\bar{ab} + a) + (\bar{ab} + b) \\ (\text{Reduction}) (\bar{ab} + a) + (\bar{ab} + b) = \bar{ab} + a + b = a + b + \bar{ab} = a + b + \bar{a} + \bar{b} \\ = a + \bar{a} + b + \bar{b} = 1 + 1 = 1$$

**EXAMPLE 5.31** Simplify the following Boolean expressions to a minimum number of literals.

- |   |   |
|---|---|
| (a) $\bar{x}\bar{y} + xy + \bar{x}y$  | (b) $x\bar{y} + \bar{y}\bar{z} + \bar{x}\bar{z}$        |
| (c) $(x+y)(x+\bar{y})$  | (d) $\bar{x}y + xy + x\bar{z} + x\bar{y}\bar{z}$        |
| (e) $(A+B)(\bar{A}+C)(\bar{B}+D)(C\bar{D})$                                   | (f) $\bar{A}\bar{C} + ABC + A\bar{C}$ to three literals |
| (g) $(\bar{x}\bar{y} + z) + z + xy + wz$ to three literals                    |   |
| (h) $\bar{A}B(\bar{D} + \bar{C}D) + B(A + \bar{A}CD)$ to one literal.         |   |
| (i) $(\bar{A} + C)(\bar{A} + \bar{C})(\bar{A} + B + \bar{C}D)$ to one literal |   |
| (j) $AB + A(B + C) + \bar{B}(B + D)$  | (k) $A + B + \bar{A}\bar{B}C$                           |
| (l) $\bar{A}B + \bar{A}\bar{B}\bar{C} + \bar{A}BCD + \bar{A}B\bar{C}DE$       | (m) $ABEF + AB\bar{E}F + \bar{A}BEF$                    |
| (n) $ABCD + A + ABD + (\bar{D})(\bar{A}\bar{B}\bar{C})$                       | (o) $x[y + z(\overline{xy + xz})]$                      |
| (p) $\bar{x}\bar{z} + \bar{y}\bar{z} + y\bar{z} + xyz$                        |   |

**Solution**

The simplification of the above Boolean expressions is as follows:

- (a)  $\bar{x}\bar{y} + xy + \bar{x}y = \bar{x}\bar{y} + y(x + \bar{x}) = \bar{x}\bar{y} + y = (y + \bar{y})(y + \bar{x}) = \bar{x} + y$
- (b)  $x\bar{y} + \bar{y}\bar{z} + \bar{x}\bar{z} = x\bar{y} + \bar{x}\bar{z} + \bar{y}\bar{z}(x + \bar{x}) = x\bar{y} + \bar{x}\bar{z} + \bar{y}\bar{z}x + \bar{y}\bar{z}\bar{x} \\ = x\bar{y}(1 + \bar{z}) + \bar{x}\bar{z}(1 + \bar{y}) = x\bar{y} + \bar{x}\bar{z}$
- (c)  $(x+y)(x+\bar{y}) = xx + xy + x\bar{y} + y\bar{y} = x + xy + x\bar{y} + 0 = x(1 + y + \bar{y}) + 0 = x$
- (d)  $\bar{x}y + xy + x\bar{z} + x\bar{y}\bar{z} = y(x + \bar{x}) + x\bar{z}(1 + \bar{y}) = y + x\bar{z}$
- (e)  $(A+B)(\bar{A}+C)(\bar{B}+D)(C\bar{D}) = (A\bar{A} + AC + \bar{A}B + BC)(\bar{B}C\bar{D} + DC\bar{D}) \\ = (AC + \bar{A}B + BC)\bar{B}C\bar{D} = A\bar{B}C\bar{D}$
- (f)  $\bar{A}\bar{C} + ABC + A\bar{C} = \bar{C}(\bar{A} + A) + ABC = \bar{C} + ABC = (\bar{C} + C)(\bar{C} + AB) = \bar{C} + AB$
- (g)  $(\bar{x}\bar{y} + z) + z + xy + wz = \bar{x}\bar{y} + \bar{z} + z(1 + w) + xy = (x + y)\bar{z} + z + xy \\ = (z + \bar{z})(z + x + y) + xy \\ = x + y + z + xy = x + y + z$

$$\begin{aligned}
 (h) \bar{A}\bar{B}(\bar{D} + \bar{C}\bar{D}) + B(A + \bar{A}CD) &= \bar{A}\bar{B}\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D} + AB + \bar{A}BCD \\
 &= \bar{A}\bar{B}D(C + \bar{C}) + \bar{A}\bar{B}\bar{D} + AB = \bar{A}\bar{B}D + \bar{A}\bar{B}\bar{D} + AB \\
 &= \bar{A}\bar{B}(D + \bar{D}) + AB = AB + \bar{A}\bar{B} = B(A + \bar{A}) = B
 \end{aligned}$$

$$\begin{aligned}
 (i) (\bar{A} + C)(\bar{A} + \bar{C})(\bar{A} + B + \bar{C}\bar{D}) &= (\bar{A} + \bar{A}C + \bar{A}\bar{C} + C\bar{C})(\bar{A} + B + \bar{C}\bar{D}) \\
 &= \bar{A}(1 + C + \bar{C})(\bar{A} + B + \bar{C}\bar{D}) = \bar{A}(\bar{A} + B + \bar{C}\bar{D}) \\
 &= \bar{A} + \bar{A}B + \bar{A}\bar{C}\bar{D} = \bar{A}(1 + B + \bar{C}\bar{D}) = \bar{A}
 \end{aligned}$$

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

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

$$(l) \bar{A}B + \bar{A}\bar{B}\bar{C} + \bar{A}BCD + \bar{A}\bar{B}\bar{C}\bar{D}\bar{E} = \bar{A}B(1 + \bar{C} + CD + \bar{C}\bar{D}\bar{E}) = \bar{A}B$$

$$\begin{aligned}
 (m) ABEF + ABE\bar{F} + \bar{A}BEF &= AB(EF + \bar{E}\bar{F}) + \bar{A}BEF = AB + \bar{A}BEF \\
 &= (AB + \bar{A}B)(AB + EF) = AB + EF
 \end{aligned}$$

$$\begin{aligned}
 (n) ABC\bar{D} + A + ABD + (\bar{D})(\bar{A}\bar{B}\bar{C}) &= ABC\bar{D} + A + ABD + (\bar{A}\bar{B}\bar{C}\bar{D}) \\
 &= A(1 + \bar{B}\bar{D} + B\bar{C}\bar{D}) + \bar{A}\bar{B}\bar{C}\bar{D} = A + \bar{A}\bar{B}\bar{C}\bar{D} \\
 &= (A + \bar{A})(A + \bar{B}\bar{C}\bar{D}) = A + \bar{B}\bar{C}\bar{D}
 \end{aligned}$$

$$\begin{aligned}
 (o) x[y + z(\overline{xy} + \overline{xz})] &= x[y + z(\overline{xy} \cdot \overline{xz})] = xy + xz \cdot \overline{xy} \cdot \overline{xz} = xy + 0 \\
 &= xy
 \end{aligned}$$

$$\begin{aligned}
 (p) \bar{x}\bar{z} + \bar{y}\bar{z} + y\bar{z} + xyz &= \bar{x}\bar{z} + \bar{z}(y + \bar{y}) + xyz = \bar{x}\bar{z} + \bar{z} + xyz \\
 &= \bar{z}(1 + \bar{x}) + xyz = \bar{z} + xyz = (\bar{z} + z)(\bar{z} + xy) = \bar{z} + xy
 \end{aligned}$$

**EXAMPLE 5.32** Given  $\bar{A}\bar{B} + \bar{A}B = C$ , find  $\bar{A}C + \bar{A}\bar{C}$ .

**Solution**

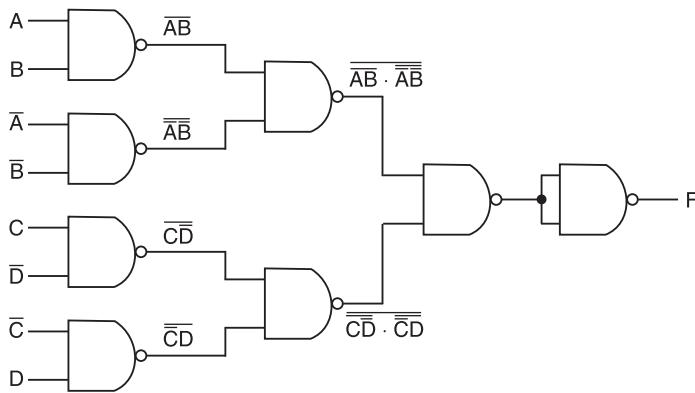
$$\begin{aligned}
 \bar{A}C + \bar{A}\bar{C} &= \overline{\bar{A}(\bar{A}\bar{B} + \bar{A}B)} + \bar{A}(\bar{A}\bar{B} + \bar{A}B) \\
 &= \overline{\bar{A}(\bar{A} + \bar{B} + \bar{A}B)} + \bar{A}(\bar{A} + \bar{B} + \bar{A}B) \\
 &= \bar{A}\bar{B} + \bar{A}[\bar{A}(1 + B) + \bar{B}] \\
 &= \bar{A} +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

**EXAMPLE 5.33** Draw the logic diagram using only two input NAND gates to implement the following expression  $F = (AB + \bar{A}\bar{B})(\bar{C}\bar{D} + \bar{C}\bar{D})$ .

**Solution**

To implement an expression using only NAND gates, the expression must have only product terms. The given expression in terms of product terms and the corresponding logic diagram are shown below.

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

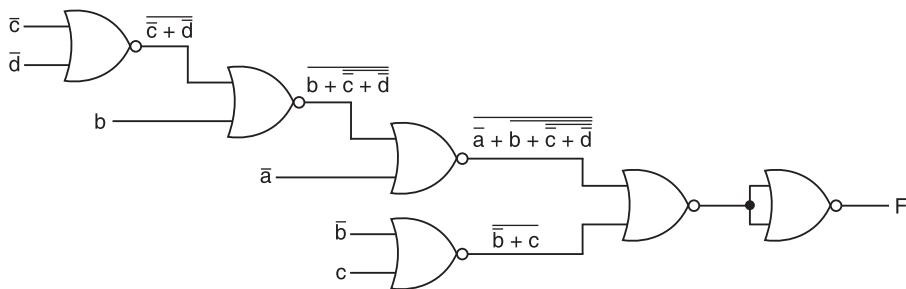
**EXAMPLE 5.34**

- (a) Implement the following function using only NOR gates  $F = a(b + cd) + b\bar{c}$ .  
 (b) Implement the following function using only NAND gates  $G = (a + \bar{b}) \cdot (cd + \bar{e})$   
 (c) Give the minimum two-level SOP realization of the following switching function using only NAND gates  $F = \sum m(0, 3, 4, 5, 7)$ .

**Solution**

- (a) To implement the expression using only NOR gates, the expression should have only sum terms. The given expression in terms of sum terms is given below and its implementation using NOR gates is shown in Figure 5.7.

$$F = a(b + cd) + b\bar{c} = a \cdot (b + cd) + (\bar{b}\bar{c}) = \bar{a} + b + \bar{c} + \bar{d} + \bar{b} + c$$

**Figure 5.7** Example 5.34a.

- (b) To implement the expression using only NAND gates the expression should be in terms of only product terms. The given expression in terms of product terms is given below and its implementation using only NAND gates is shown in Figure 5.8.

$$G = (a + \bar{b}) \cdot (cd + \bar{e}) = (a + \bar{b}) \cdot (cd + \bar{e}) = (\bar{a}b)(\bar{c}d \cdot e)$$

- (c) The minimal expression is

$$\begin{aligned} F = \sum m(0, 3, 4, 5, 7) &= \bar{x}\bar{y}\bar{z} + \bar{x}yz + x\bar{y}\bar{z} + x\bar{y}z + xyz \\ &= x\bar{y}(z + \bar{z}) + \bar{y}\bar{z}(x + \bar{x}) + yz(x + \bar{x}) \\ &= x\bar{y} + yz + \bar{y}\bar{z} \\ &= \bar{x}\bar{y} \cdot yz \cdot \bar{y}\bar{z} \end{aligned}$$

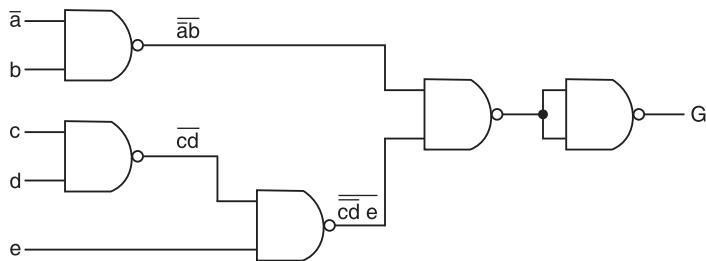


Figure 5.8 Example 5.34b.

The realization using only NAND gates is shown in Figure 5.9.

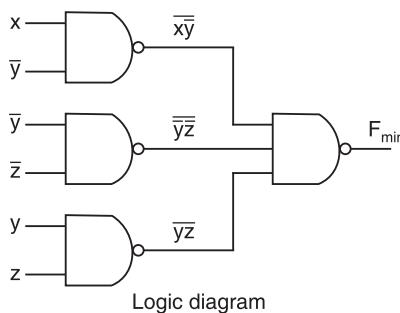


Figure 5.9 Example 5.34c.

**EXAMPLE 5.35** Derive the Boolean expression for a two-input Ex-OR gate to realize with two input NAND gates without using complemented variables and draw the circuit.

### Solution

The Boolean expression for a two-input ( $A, B$ ) Ex-OR gate is  $F = A\bar{B} + \bar{A}B$ . The derivation of the expression to realize with two input NAND gates without using complemented variables is given below. Its realization is shown in Figure 5.10.

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

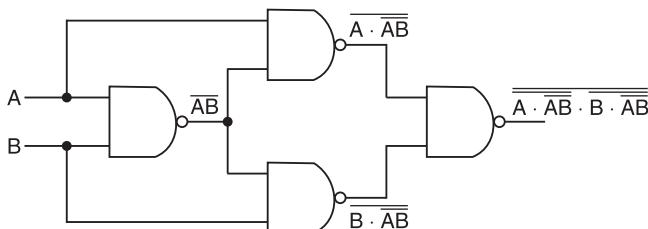


Figure 5.10 Example 5.35.

**EXAMPLE 5.36** Derive the Boolean expression for a two-input Ex-NOR gate to realize with two input NOR gates, without using complemented variables and draw the circuit.

**Solution**

We know that the Boolean expression for a two-input Ex-NOR gate is the complement of the Boolean expression for a two input Ex-OR gate. The derivation of the expression to realize with two input NOR gates without using complemented variables is given below. Its realization is shown in Figure 5.11.

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

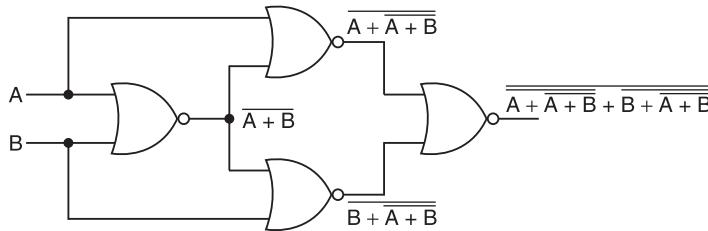


Figure 5.11 Example 5.36.

**EXAMPLE 5.37** Redraw the circuit given in Figure 5.12 after simplification.

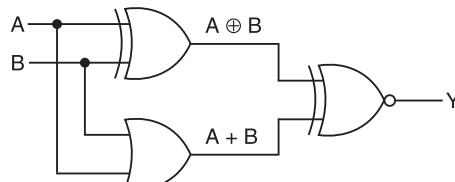
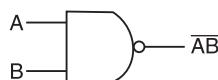


Figure 5.12 Example 5.37.

**Solution**

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

So, the simplified circuit is just a two-input NAND gate.



**EXAMPLE 5.38** Redraw the circuit given in Figure 5.13 after simplification.

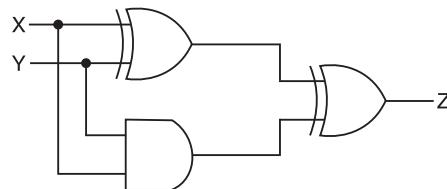


Figure 5.13 Example 5.38.

**Solution**

From the logic diagram, we see that

$$\begin{aligned}
 Z &= (X \oplus Y) \oplus (XY) \\
 &= (\overline{X \oplus Y}) \cdot XY + (X \oplus Y) \cdot \overline{XY} \\
 &= (\overline{X}\overline{Y} + XY)XY + (X\overline{Y} + \overline{X}Y)(\overline{X} + \overline{Y}) \\
 &= XY + X\overline{Y} + \overline{X}Y = X(Y + \overline{Y}) + \overline{X}Y = X + \overline{X}Y \\
 &= (X + \overline{X})(X + Y) = X + Y
 \end{aligned}$$

So, the simplified circuit is a two input OR gate.



**EXAMPLE 5.39** Give three possible ways to express the function

$$F = \overline{ABD} + \overline{ABC}\overline{D} + \overline{ABD} + AB\overline{CD}$$

**Solution**

$$\begin{aligned}
 F &= \overline{ABD}(1 + \overline{C}) + \overline{ABD} + AB\overline{CD} \\
 &= \overline{ABD} + \overline{ABD} + AB\overline{CD} \\
 &= \overline{A}(\overline{BD} + BD) + A(B\overline{CD}) \\
 &= \overline{A}(\overline{BD} + BD) + A(B\overline{CD}) + (\overline{BD} + BD)(B\overline{CD}) \\
 &= \overline{A}(\overline{BD} + BD) + A(B\overline{CD}) + B\overline{CD} \\
 &= \overline{A}(\overline{BD} + BD) + B\overline{CD} \quad (8 \text{ literals}) \\
 &= \overline{A}(B \odot D) + B\overline{CD} \quad (6 \text{ literals}) \\
 &= \overline{A}(B \oplus D) + B\overline{CD} \quad (6 \text{ literals})
 \end{aligned}$$

$$\begin{aligned}
 F &= \overline{ABD} + \overline{ABD} + AB\overline{CD} \\
 &= BD(\overline{A} + A\overline{C}) + (\overline{ABD}) \\
 &= BD(\overline{A} + A)(\overline{A} + \overline{C}) + \overline{ABD} \\
 &= BD(\overline{A} + \overline{C}) + \overline{ABD} \quad (7 \text{ literals}) \\
 F &= BD(\overline{AC}) + \overline{ABD} \quad (7 \text{ literals})
 \end{aligned}$$

**EXAMPLE 5.40** Simplify the following expressions and implement them with NAND gate circuits.

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

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

**Solution**

The simplification of the given expressions is given below. Their implementation with NAND gate circuits is shown in Figure 5.13.

$$\begin{aligned} (a) F &= A\bar{B} + ABD + A\bar{B}\bar{D} + \bar{A}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C} \\ &= A\bar{B} + AB + \bar{A}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C} = A(\bar{B} + B) + \bar{A}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C} \\ &= A + \bar{A}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C} = (A + \bar{A})(A + \bar{C}\bar{D}) + \bar{A}\bar{B}\bar{C} = A + \bar{C}\bar{D} + \bar{A}\bar{B}\bar{C} \\ &= (A + \bar{A})(A + \bar{B}\bar{C}) + \bar{C}\bar{D} = (A + \bar{B}\bar{C}) + \bar{C}\bar{D} = \bar{A} \cdot \bar{B}\bar{C} \cdot \bar{C}\bar{D} \end{aligned}$$

$$\begin{aligned} (b) G &= BD + B\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} = B(D + \bar{C}\bar{D}) + A\bar{B}\bar{C}\bar{D} = B(D + C)(D + \bar{D}) + A\bar{B}\bar{C}\bar{D} \\ &= BD + BC + A\bar{B}\bar{C}\bar{D} = \overline{BD} \cdot \overline{BC} \cdot \overline{ABC\bar{D}} \end{aligned}$$

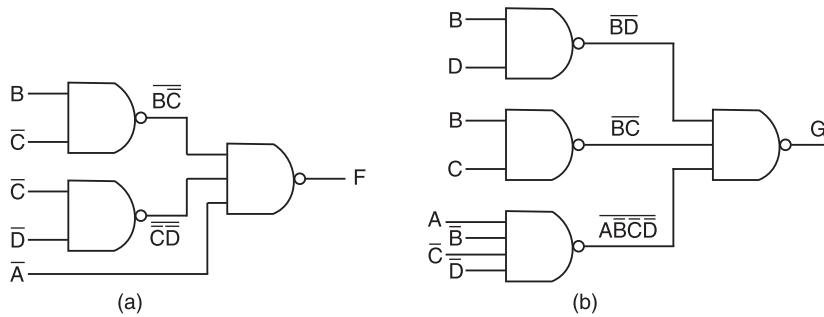


Figure 5.13 Example 5.40.

**EXAMPLE 5.41** Express the following functions as sum of minterms and product of maxterms:

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

$$(b) F(x, y, z) = (xy + z)(xz + y)$$

**Solution**

Considering that the missing minterms in SOP form become the maxterms in POS form and vice versa, the given expressions are expressed as sum of minterms and product of maxterms as shown below.

$$\begin{aligned} (a) F &= \bar{B}D + \bar{A}D + BD = X0X1 + 0XX1 + X1X1 \\ &= 0001, 0011, 1001, 1011, 0001, 0011, 0101, 0111, 0101, 0111, 1101, 1111 \\ &= \Sigma m(1, 3, 9, 11, 1, 3, 5, 7, 5, 7, 13, 15) \\ &= \Sigma m(1, 3, 5, 7, 9, 11, 13, 15) \\ &= \Pi M(0, 2, 4, 6, 8, 10, 12, 14) \end{aligned}$$

$$\begin{aligned}
 (b) \quad F &= (xy + z)(xz + y) = (x + z)(y + z)(x + y)(y + z) \\
 &= (x + y)(x + z)(y + z) \\
 &= (00X)(0X0)(X00) \\
 &= (000)(001)(000)(010)(000)(100) \\
 &= \prod M(0, 1, 0, 2, 0, 4) = \prod M(0, 1, 2, 4) \\
 &= \sum m(3, 5, 6, 7)
 \end{aligned}$$

**EXAMPLE 5.42** Express the Boolean function  $F = AB + \bar{A}C$  in a product of maxterm form.

**Solution**

The given expression is in SOP form. So convert it first into POS form and then to standard POS form.

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

The function has three variables: A, B and C. Each OR term is missing one variable; therefore

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

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

$$F = (A + B + C)(A + \bar{B} + C)(\bar{A} + B + C)(\bar{A} + B + \bar{C}) = M_0 M_2 M_4 M_5 = \prod M(0, 2, 4, 5)$$

The product symbol,  $\prod$  denotes ANDing of maxterms.

### SHORT QUESTIONS AND ANSWERS

1. What is a literal?  
A. A logical variable in complemented or uncomplemented form is known as a literal.
2. What is a binary variable?  
A. Binary variable is a variable that can have only two values 0 and 1.
3. What is switching theory?  
A. Switching theory is the mathematical theory of logic circuits.
4. What is Boolean algebra?  
A. Boolean algebra is a system of mathematical logic which uses the letters of English alphabet to represent variables. Any single variable or a function of the variables can have a value of either 0 or 1. In Boolean algebra there is no subtraction or division. Only logical addition and logical multiplication are performed. There are no fractions or negative numbers. It is algebra of binary variables.
5. What are the basic operations in Boolean algebra?  
A. The basic operations in Boolean algebra are as follows.
  - (a) *AND operation*. It is the same as logical multiplication. It is denoted by ‘·’ or  $\cap$  or  $\wedge$  or no symbol at all.

(b) *OR operation*. It is the same as logical addition. It is denoted by + or  $\cup$  or  $\vee$ .

(c) *NOT operation*. It is the same as inversion or complementation. It is denoted by a bar or prime.

**6.** What are NAND, NOR, X-OR, and X-NOR operations in Boolean algebra?

A. In Boolean algebra the NAND operation is a combination of AND and NOT operations. It is complementation of AND operation. The NOR operation is a combination of OR and NOT operations. It is complementation of OR operation. The X-OR operation is called the modulo-2 addition operation. The X-NOR operation is a combination of X-OR and NOT operations.

**7.** What do you mean by an Axiom?

A. Axioms or postulates of Boolean algebra are a set of logical expressions that we accept without proof and upon which we can build a set of useful theorems.

**8.** State De Morgan's theorem.

A. De Morgan's theorem states that

(a) the complement of a sum of variables is equal to the product of their individual complements and

(b) the complement of a product of variables is equal to the sum of their individual complements.

**9.** What is the use of De Morgan's theorem?

A. De Morgan's theorem allows removal of variables from under a NOT sign. It allows transformation from SOP form to POS form and vice versa.

**10.** How do you De Morganize a Boolean expression?

A. To De Morganize an expression, complement the entire function, change all the ANDs to ORs, all the ORs to ANDs, 0s to 1s and 1s to 0s, and then complement each of the individual variables.

**11.** How do you obtain the dual of an expression?

A. The dual of an expression is obtained by changing the ANDs to ORs, ORs to ANDs, 0s to 1s, and 1s to 0s. The variables are not complemented.

**12.** What does a logical operation in a Boolean expression represent?

A. Every logic operation in the Boolean expression represents a corresponding element of hardware.

**13.** What are the two basic forms of Boolean expressions?

A. The two basic forms of Boolean expressions are (a) SOP form and (b) POS form.

**14.** What is hybrid form?

A. The hybrid form of realization is a combination of both SOP and POS forms.

**15.** What is AOI logic?

A. AOI logic is one in which the circuits are realized using AND, OR, and NOT gates only.

**16.** What is universal logic?

A. Universal logic is one in which the circuits are realized using either only NAND gates or only NOR gates.

**17.** What are expandable gates?

A. A/O gates in which an additional variable or a combination of variables can be included in the logic operation are called expandable gates.

**18.** What do you mean by an  $n$ -wide A/O gate?

A. A circuit having  $n$  AND gates feeding an OR gate is called an  $n$ -wide A/O gate.

**19.** What do you mean by an active-LOW gate?

A. Active-LOW gates and circuits are those in which the action represented or initiated by a variable occurs when it is equal to a 0.

20. What do you mean by an active-HIGH gate?  
 A. Active-HIGH gates and circuits are those in which the action represented or initiated by a variable occurs when it is equal to a 1.
21. On logic diagrams, what does a bubble at the input indicate?  
 A. On logic diagrams, an inversion bubble at the point where the input is connected to a gate indicates an active-LOW input.
22. What does an OR gate in positive logic system become in the negative logic system?  
 A. An OR gate in the positive logic system becomes an AND gate in the negative logic system and vice versa.
23. Write the Boolean algebraic laws.  
 A. The laws of Boolean algebra are:
  - (a) Laws of complementation  $\bar{0} = 1$ ;  $\bar{1} = 0$ ;  $\bar{\bar{A}} = A$   
 $A = 0$  only when  $\bar{A} = 1$ ;  $A = 1$  only when  $\bar{A} = 0$
  - (b) AND laws  $0 \cdot 0 = 0$ ;  $0 \cdot 1 = 0$ ;  $1 \cdot 0 = 0$ ;  $1 \cdot 1 = 1$   
 $A \cdot 0 = 0$ ;  $A \cdot 1 = A$ ;  $A \cdot A = A$ ;  $A \cdot \bar{A} = 0$
  - (c) OR laws  $0 + 0 = 0$ ;  $0 + 1 = 1$ ;  $1 + 0 = 1$ ;  $1 + 1 = 1$   
 $A + 0 = A$ ;  $A + 1 = 1$ ;  $A + A = A$ ;  $A + \bar{A} = 1$
  - (d) Commutative laws  $A + B = B + A$ ;  $AB = BA$
  - (e) Associative laws  $(A + B) + C = A + (B + C)$ ;  $(AB) \cdot C = A \cdot (BC)$
  - (f) Distributive laws  $A \cdot (B + C) = AB + AC$ ;  $A + BC = (A + B) \cdot (A + C)$
  - (g) Redundant literal rule (RLR)  $A + \bar{A}B = A + B$ ;  $A(\bar{A} + B) = AB$
  - (h) Idempotence laws  $A \cdot A = A$ ;  $A + A = A$
  - (i) Absorption laws  $A + AB = A$ ;  $A(A + B) = A$
  - (j) Consensus Theorem (included factor theorem)  $AB + \bar{A}C + BC = AB + \bar{A}C$
  - (k) Transposition theorem  $AB + \bar{A}C = (A + C)(\bar{A} + B)$
  - (l) De Morgan's theorem  $\bar{A + B} = \bar{A} \cdot \bar{B}$ ;  $\bar{AB} = \bar{A} + \bar{B}$
  - (m) Shannon's expansion theorem  

$$f(A, B, C, \dots) = A \cdot f(1, B, C, \dots) + \bar{A} \cdot f(0, B, C, \dots)$$

$$f(A, B, C, \dots) = [A + f(0, B, C, \dots)] \cdot [A + f(1, B, C, \dots)]$$
24. When do you say that a signal is asserted?  
 A. When a signal is in its active state, it is said to be asserted. When it is not in its active state, i.e. when it is inactive, it is said to be unasserted.
25. What do you mean by assertion level?  
 A. The assertion level refers to the signal level necessary to cause an event to occur.
26. What are the merits and demerits of hybrid logic?  
 A. Hybrid logic reduces the number of gate inputs required for realization, but results in multi level logic. Different inputs pass through different number of gates to reach the output. It leads to non-uniform propagation delay between input and output and may give rise to logic race.
27. What is the advantage of SOP and POS forms of realization?  
 A. The SOP and POS realizations give rise to two-level logic. The two level logic provides uniform time delay between input and output, because each input signal has to pass through two gates to reach the output. So it does not suffer from the problem of logic race.



## REVIEW QUESTIONS

1. State and prove (a) commutative, (b) associative, (c) distributive, (d) Redundant literal rule, (e) idempotence, and (f) absorption laws of Boolean algebra.
2. State and prove (a) consensus theorem, (b) transposition theorem, and (c) De Morgan's theorem.
3. What are the steps followed in the reduction of Boolean expressions?
4. How do you convert AOI logic to (a) NAND logic and (b) NOR logic.
5. Show that both NAND gate and NOR gate are universal gates.
6. Realize X-OR operation using (a) Only NAND gates, (b) only NOR gates, and (c) AOI logic.
7. Draw logic diagrams to realize the following expressions (a)  $A \oplus B \oplus C \oplus D$  and (b)  $A \odot B \odot C \odot D$ .
8. How do you find the complement of an expression?
9. How do you find the dual of an expression?
10. How do you Demorganize an expression?

## FILL IN THE BLANKS

1. The basic operations in Boolean algebra are (a) \_\_\_\_\_, (b) \_\_\_\_\_, and (c) \_\_\_\_\_.
2. Every logical operation in a Boolean expression represents a \_\_\_\_\_.
3. The hybrid form of realization is a combination of both \_\_\_\_\_ and \_\_\_\_\_ forms.
4. On logic diagrams, a bubble at the input indicates \_\_\_\_\_.
5. An OR gate in positive logic system becomes \_\_\_\_\_ in negative logic system.
6. An AND gate in positive logic system becomes \_\_\_\_\_ in negative logic system.
7. The commutative laws say that \_\_\_\_\_.
8. The associative laws say that \_\_\_\_\_.
9. The distributive laws say that \_\_\_\_\_.
10. The redundant literal rule says that \_\_\_\_\_.
11. The idempotence laws say that \_\_\_\_\_.
12. The absorption laws say that \_\_\_\_\_.
13. The consensus theorem states that \_\_\_\_\_.
14. The consensus theorem is also called \_\_\_\_\_ theorem.
15. The transposition theorem states that \_\_\_\_\_.
16. The De Morgan's theorem states that \_\_\_\_\_.
17. The Shannon's Expansion theorem states that \_\_\_\_\_.
18. The interconnection of gates to perform a variety of logical operations is called \_\_\_\_\_.
19. The terms asserted and unasserted are synonymous with \_\_\_\_\_ and \_\_\_\_\_ respectively.
20. The assertion level refers to the \_\_\_\_\_ level necessary to cause an event to occur.
21. Hybrid logic is a \_\_\_\_\_ logic. It leads to \_\_\_\_\_ propagation delay and may give rise to \_\_\_\_\_.
22. SOP and POS realizations give rise to \_\_\_\_\_ logic. It leads to \_\_\_\_\_ time delay between input and output. So it does not suffer from the problem of \_\_\_\_\_.
23. A logic expression form most suitable for realization using only NAND gates is \_\_\_\_\_.
24. A logic expression form most suitable for realization using only NOR gates is \_\_\_\_\_.

### OBJECTIVE TYPE QUESTIONS

1.  $A + AB + ABC + ABCD + ABCDE + \dots =$ 
  - 1
  - $A$
  - $A + AB$
  - $AB$
2.  $A + \bar{A}B + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C}D + \dots =$ 
  - $A + B + C + \dots$
  - $\bar{A} + \bar{B} + \bar{C} + \bar{D} + \dots$
  - 1
  - 0
3. De Morgan's theorem states that
 

(a) $\overline{A + B} = \bar{A}\bar{B}$ and $\overline{AB} = \bar{A} + \bar{B}$	(b) $\overline{A + B} = \bar{A} + \bar{B}$ and $\overline{AB} = \bar{A}\bar{B}$
(c) $\overline{A + B} = A + B$ and $\overline{AB} = AB$	(d) $\overline{A + B} = \bar{A}\bar{B}$ and $\overline{AB} = \bar{A}\bar{B}$
4. The logic expression  $(A + B)(\bar{A} + \bar{B})$  can be implemented by giving the inputs A and B to a two-input
  - NOR gate
  - NAND gate
  - X-OR gate
  - X-NOR gate
5. The logic expression  $(\bar{A} + B)(A + \bar{B})$  can be implemented by giving the inputs A and B to a two-input
  - NOR gate
  - NAND gate
  - X-OR gate
  - X-NOR gate
6. Which of the following Boolean algebraic expressions is incorrect?
 

(a) $A + \bar{A}B = A + B$	(b) $A + AB = B$
(c) $(A + B)(A + C) = A + BC$	(d) $(A + \bar{B})(A + B) = A$
7. The simplified form of the Boolean expression  $(X + Y + XY)(X + Z)$  is
  - $X + Y + Z$
  - $XY + YZ$
  - $X + YZ$
  - $XZ + Y$
8. The simplified form of the Boolean expression  $(X + \bar{Y} + Z)(X + \bar{Y} + \bar{Z})(X + Y + Z)$  is
  - $\bar{X}Y + Z$
  - $X + \bar{Y}Z$
  - $\bar{X}Y + \bar{Z}$
  - $XY + \bar{Z}$
9.  $A + B = B + A$ ;  $AB = BA$  represent which laws?
  - Commutative
  - Associative
  - Distributive
  - Idempotence
10.  $(A + B) + C = A + (B + C)$ ;  $(AB)C = A(BC)$  represent which laws?
  - Commutative
  - Associative
  - Distributive
  - Idempotence
11.  $A(B + C) = AB + AC$ ;  $A + BC = (A + B)(A + C)$  represent which laws?
  - Commutative
  - Associative
  - Distributive
  - Idempotence
12.  $A + AB = A$ ;  $A(A + B) = A$  represent which laws?
  - Idempotence
  - Absorption
  - Associative
  - Commutative
13.  $AB + \bar{A}C + BC = AB + \bar{A}C$  represents which theorem?
  - Consensus
  - Transposition
  - De Morgan's
  - none of these
14.  $AB + \bar{A}C = (A + C)(\bar{A} + B)$  represents which theorem?
  - Consensus
  - Transposition
  - De Morgan
  - Included factor
15. The dual of a Boolean expression is obtained by
  - interchanging all 0s and 1s
  - interchanging all 0s and 1s, all + and ‘.’ signs
  - interchanging all 0s and 1s, all + and ‘.’ signs and complementing all the variables
  - interchanging all + and ‘.’ signs and complementing all the variables

## PROBLEMS

**5.6** Reduce the following Boolean expressions.

- $AB + A(B + C) + \bar{B}(B + D)$
- $ABEF + A\bar{B}\bar{F} + \bar{A}\bar{B}EF$
- $\bar{A}\bar{B} + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}CD + \bar{A}\bar{B}\bar{C}\bar{D}E$
- $\bar{B}\bar{C}\bar{D} + (\bar{B} + C + D) + \bar{B}\bar{C}\bar{D}E$
- $(WX + W\bar{Y})(X + W) + WX(\bar{X} + \bar{Y})$
- $\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$

- $(X + Y + Z)(\bar{X} + \bar{Y} + \bar{Z})X$
- $ABC[AB + \bar{C}(BC + AC)]$
- $A + B + \bar{A}\bar{B}C$
- $(\bar{A}\bar{B}C)(\bar{A}\bar{B}) + BC$
- $AB + \bar{A}\bar{C} + A\bar{B}C(AB + C)$

**5.7** Prove that

- $\overline{\overline{AB} + \bar{A} + AB} = 0$
- $\overline{\overline{AB} + ABC + A(B + \bar{A})} = 0$
- $A\bar{B}(C + BD) + \bar{A}\bar{B} = \bar{B}C + \bar{A}\bar{B}$
- $ABCD + AB(\overline{CD}) + (\overline{AB})CD = AB + CD$
- $(A\bar{B} + A\bar{C})(BC + B\bar{C})(ABC) = 0$
- $A[B + C(\overline{AB} + \overline{AC})] = AB$
- $A + \bar{B}C(A + \bar{B}\bar{C}) = A$
- $\overline{\overline{ABC} + \overline{AB} + BC} = \overline{AB}$
- $(b) \overline{AB} + \overline{AC} + A\bar{B}C(AB + C) = 1$
- $AB + A(B + C) + B(B + C) = B + AC$
- $\overline{\overline{A}\bar{B}C + (A + B + \bar{C})} + \overline{A}\bar{B}\bar{C}D = \overline{A}\bar{B}(C + D)$
- $(h) (A + \bar{A})(AB + A\bar{B}\bar{C}) = AB$
- $A\bar{B}C + \bar{A}\bar{B}C + ABC = C(A + B)$
- $(l) (\overline{A + \bar{B}\bar{C}})(A\bar{B} + \overline{ABC}) = \overline{ABC}$
- $(n) \overline{\overline{ABC}(A + B + C)} = \overline{A}\bar{B}\bar{C}$

**5.8** Apply De Morgan's theorem to each of the following expressions.

- $\overline{P(Q + R)}$
- $\overline{(P + \bar{Q})(\bar{R} + S)}$
- $\overline{\overline{(A + B)}(\overline{C + D})} \overline{\overline{(E + F)}(\overline{G + H})}$
- $\overline{(A + B + \bar{C} + D)(0 + \bar{A}\bar{B}\bar{C}\bar{D})}$

**5.9** Without reducing, convert the following expressions to NAND logic.

- $(A + B)(C + D)$
- $(A + C)(ABC + ACD)$
- $(A + \bar{B}\bar{C})D$
- $AB + CD (A\bar{B} + CD)$

**5.10** Without reducing, convert the following expressions to NAND logic.

- $A + BC + ABC$
- $(XY + Z)(XY + P)$
- $(1 + A)(ABC)$

**5.11** Without reducing, convert the following expressions to NOR logic.

- $X + Y + XY$
- $(X\bar{Y} + X + \overline{X + Y})$
- $(1 + A)(AC)$

**5.12** Without reducing, implement the following expressions in AOI logic and then convert them into (1) NAND logic and (2) NOR logic.

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

**5.13** Prove that

- If  $A + B = A + C$  and  $\bar{A} + B = \bar{A} + C$ , then  $B = C$ .
- If  $A + B = A + C$  and  $AB = AC$ , then  $B = C$ .

**5.14** Given  $A\bar{B} + \bar{A}B = C$ , show that  $A\bar{C} + \bar{A}C = B$ .

# 6

## MINIMIZATION OF SWITCHING FUNCTIONS

### 6.1 INTRODUCTION

We have seen how Boolean expressions can be simplified algebraically, but being not a systematic method we can never be sure whether the minimal expression obtained is the real minimal or not. The effectiveness of algebraic simplification depends on our familiarity with, and ability to apply Boolean algebraic rules, laws and theorems. The Karnaugh map (K-map) method, on the other hand, is a systematic method of simplifying the Boolean expressions. *The K-map is a chart or a graph, composed of an arrangement of adjacent cells, each representing a particular combination of variables in sum or product form.* Like a truth table, it is a means of showing the relationship between the logic inputs and the desired output. Although a K-map can be used for problems involving any number of variables, it becomes tedious for problems involving five or more variables. Usually it is limited to six variables. An  $n$  variable function can have  $2^n$  possible combinations of product terms in SOP form, or  $2^n$  possible combinations of sum terms in POS form. Since the K-map is a graphical representation of Boolean expressions, a two-variable K-map will have  $2^2 = 4$  cells or squares, a three variable map will have  $2^3 = 8$  cells or squares, and a four variable map will have  $2^4 = 16$  cells, and so on.

Any Boolean expression can be expressed in a *standard or canonical or expanded sum (OR) of products (AND) form—SOP form*—or in a *standard or canonical or expanded product (AND) of sums (OR) form—POS form*. A standard SOP form is one in which a number of product terms, each one of which contains all the variables of the function either in complemented or non-complemented form, are summed together. A standard POS form is one in which a number of sum terms, each one of which contains all the variables of the function either in complemented or non-complemented form, are multiplied together. Each of the product terms in the standard SOP form is called a *minterm* and each of the sum terms in the standard POS form is called a *maxterm*.

For simplicity, the minterms and maxterms are usually represented as binary words in terms of 0s and 1s, instead of actual variables. For minterms, the binary words are formed by representing each non-complemented variable by a 1 and each complemented variable by a 0, and the decimal equivalent of this binary word is expressed as a subscript of lower case  $m$ , i.e.  $m_0, m_2, m_5, m_7$ , etc. For maxterms, the binary words are formed by representing each non-complemented variable by a 0 and each complemented variable by a 1, and the decimal equivalent of this binary word is expressed as a subscript of the upper-case letter  $M$ , i.e.  $M_0, M_1$ , etc. Any given function which is not in the standard form, can always be converted to standard form by *unreducing*, that is, *expanding* the function.

A standard SOP form can always be converted to a standard POS form, by treating the missing minterms of the SOP form as the maxterms of the POS form. Similarly, a standard POS form can always be converted to a standard SOP form, by treating the missing maxterms of the POS form as the minterms of the corresponding SOP form.

## 6.2 TWO-VARIABLE K-MAP

A two-variable expression can have  $2^2 = 4$  possible combinations of the input variables A and B. Each of these combinations,  $\bar{A}\bar{B}$ ,  $\bar{A}B$ ,  $A\bar{B}$ , and  $AB$  (in the SOP form) is called a minterm. Instead of representing the minterms in terms of the input variables, using the shorthand notation the minterms may be represented in terms of their decimal designations— $m_0$  for  $\bar{A}\bar{B}$ ,  $m_1$  for  $\bar{A}B$ ,  $m_2$  for  $A\bar{B}$ , and  $m_3$  for  $AB$ , assuming that A represents the MSB. The letter  $m$  stands for minterm and the subscript represents the decimal designation of the minterm.

The presence (absence) of a minterm in the expression indicates that the output of the logic circuit assumes logic 1 (logic 0) level for that combination of input variables.

Consider the expression  $f = \bar{A}\bar{B} + \bar{A}B + AB$ . It can be expressed using minterms as

$$F = m_0 + m_2 + m_3 = \Sigma m (0, 2, 3)$$

and should be read as the sum of minterms 0, 2, and 3. It can also be represented in terms of a truth table as shown in Table 6.1.

**Table 6.1**

<b>Minterm</b>	<b>Inputs</b>		<b>Output <i>f</i></b>
	<b>A</b>	<b>B</b>	
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	1

The first column indicates the minterm designation, the second column indicates the input combinations, and the last column indicates the presence or absence of that minterm in the output expression. A 1 in the output column indicates that the output contains that particular minterm in its sum and a 0 in that column indicates that the particular minterm does not appear in the expression for output. Such information about the two-variable expression can also be indicated by a two-variable K-map.

### 6.2.1 Mapping of SOP Expressions

A two-variable K-map has  $2^2 = 4$  squares. These squares are called *cells*. Each square on the K-map represents a unique minterm. The minterm designations of the squares are shown in Figure 6.1. A 1 placed in any square indicates that the corresponding minterm is included in the output expression, and a 0 or no entry in any square indicates that the corresponding minterm does not appear in the expression for output.

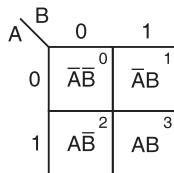


Figure 6.1 The minterms of a two-variable K-map.

The mapping of the expression  $\Sigma m(0, 2, 3)$  is shown in Figure 6.2.

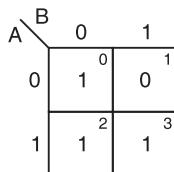


Figure 6.2 K-map of  $\Sigma m(0, 2, 3)$ .

**EXAMPLE 6.1** Map the expression  $f = \bar{A}B + A\bar{B}$ .

**Solution**

The given expression in minterms is

$$f = m_1 + m_2 = \Sigma m(1, 2)$$

The K-map is shown in Figure 6.3.

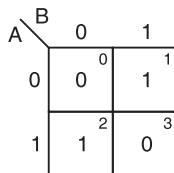


Figure 6.3 Example 6.1: K-map of  $\Sigma m(1, 2)$ .

### 6.2.2 Minimization of SOP Expressions

To minimize a Boolean expression given in the SOP form by using the K-map, we have to look for adjacent squares having 1s, that is, minterms adjacent to each other, and combine them to form larger squares to eliminate some variables. Two squares are said to be adjacent to each other, if their minterms differ in only one variable. For example, in a two-variable K-map,  $m_0$  and  $m_1$ , i.e.  $\bar{A}\bar{B}$  and  $\bar{A}B$  differ only in variable B ( $\bar{A}$  is common to both of them). So, they may be combined to form a 2-square to eliminate the variable B.

Similarly, minterms  $m_0 (\bar{A}\bar{B})$  and  $m_2 (A\bar{B})$ ;  $m_1 (\bar{A}B)$  and  $m_3 (AB)$ ; and  $m_2 (A\bar{B})$  and  $m_3 (AB)$  are adjacent to each other. However, minterms  $m_0 (\bar{A}B)$  and  $m_3 (AB)$ , and  $m_1 (\bar{A}B)$  and  $m_2 (A\bar{B})$  are not adjacent to each other, because they differ in more than one variable.

The necessary (but not sufficient) condition for adjacency of minterms is that their decimal designations must differ by a power of 2. A minterm can be combined with any number of minterms adjacent to it to form larger squares.

Two minterms, which are adjacent to each other, can be combined to form a bigger square called a 2-square or a pair. This eliminates one variable—the variable that is not common to both the minterms. For example in Figure 6.4,

$m_0$  and  $m_1$  can be combined to yield,

$$f_1 = m_0 + m_1 = \bar{A}\bar{B} + \bar{A}B = \bar{A}(B + \bar{B}) = \bar{A}$$

$m_0$  and  $m_2$  can be combined to yield,

$$f_2 = m_0 + m_2 = \bar{A}\bar{B} + A\bar{B} = \bar{B}(A + \bar{A}) = \bar{B}$$

$m_1$  and  $m_3$  can be combined to yield,

$$f_3 = m_1 + m_3 = \bar{A}B + AB = B(A + \bar{A}) = B$$

$m_2$  and  $m_3$  can be combined to yield,

$$f_4 = m_2 + m_3 = A\bar{B} + AB = A(B + \bar{B}) = A$$

$m_0, m_1, m_2$ , and  $m_3$  can be combined to yield,  $f_5 = m_0 + m_1 + m_2 + m_3$

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

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

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

$$= 1$$

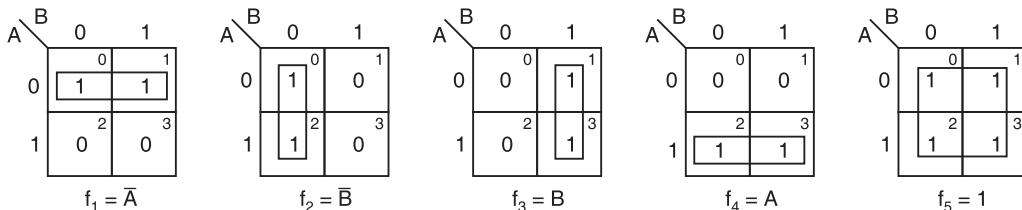


Figure 6.4 The possible minterm groupings in a two-variable K-map.

Two 2-squares adjacent to each other can be combined to form a 4-square. A 4-square eliminates 2 variables. A 4-square is called a *quad*.

To read the squares on the map after minimization, consider only those variables which remain constant throughout the square, and ignore the variables which are varying. Write the non-complemented variable if the variable is remaining constant as a 1, and the complemented variable if the variable is remaining constant as a 0, and write the variables as a product term. In Figure 6.4,  $f_1$  is read as  $\bar{A}$ , because, along the square, A remains constant as a 0, that is, as  $\bar{A}$ , whereas B is changing from 0 to 1.  $f_3$  is read as B, because, along the square, B remains constant as a 1, whereas A is changing from 0 to 1.  $f_5$  is read as a 1, because, no variable remains constant throughout the square, which means that the output is a 1 for any combination of inputs.

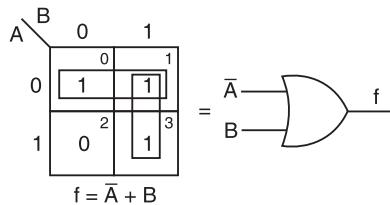
**EXAMPLE 6.2** Reduce the expression  $f = \bar{A}\bar{B} + \bar{A}B + AB$  using mapping.

**Solution**

Expressed in terms of minterms, the given expression is

$$f = m_0 + m_1 + m_3 = \Sigma m (0, 1, 3)$$

Figure 6.5 shows the K-map for  $f$  and its reduction. In one 2-square, A is constant as a 0 but B varies from a 0 to a 1, and in the other 2-square, B is constant as a 1 but A varies from a 0 to a 1. So, the reduced expression is  $\bar{A} + B$ . It requires two gate inputs for realization as shown in the figure.



**Figure 6.5** Example 6.2: K-map in SOP form, and logic diagram.

The main criterion in the design of a digital circuit is that its cost should be as low as possible. To design a circuit with the least cost, the expression used to realize that circuit must be minimal. Since the cost is roughly proportional to the number of gate inputs in the circuit, an expression is considered minimal only if it corresponds to the least possible number of gate inputs. There is no guarantee that the minimal expression obtained from the K-map in the SOP form is the real minimal. To obtain the real minimal expression, we obtain the minimal expressions for any problem in both the SOP and POS forms by using the K-maps and then take the minimal of these two minimals.

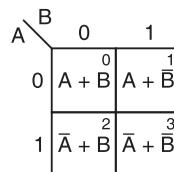
We know that the 1s on the K-map indicate the presence of minterms in the output expression, whereas the 0s indicate the absence of minterms. Since the absence of a minterm in the SOP expression means the presence of the corresponding maxterm in the POS expression of the same problem, when a SOP expression is plotted on the K-map, 0s or no entries on the K-map represent the maxterms. To obtain the minimal expression in the POS form, consider the 0s on the K-map and follow the procedure used for combining 1s. Also, since the absence of a maxterm in the POS expression means the presence of the corresponding minterm in the SOP expression of the same problem, when a POS expression is plotted on the K-map, 1s or no entries on the K-map represent the minterms.

### 6.2.3 Mapping of POS Expressions

Each sum term in the standard POS expression is called a *maxterm*. A function in two variables ( $A, B$ ) has four possible maxterms,  $A + B$ ,  $A + \bar{B}$ ,  $\bar{A} + B$ , and  $\bar{A} + \bar{B}$ . They are represented as  $M_0$ ,  $M_1$ ,  $M_2$ , and  $M_3$  respectively. The upper-case letter M stands for maxterm and its subscript denotes the decimal designation of that maxterm obtained by treating the non-complemented variable as a 0 and the complemented variable as a 1 and putting them side by side for reading the decimal equivalent of the binary number so formed.

For mapping a POS expression on to the K-map, 0s are placed in the squares corresponding to the maxterms which are present in the expression and 1s are placed (or no entries are made) in

the squares corresponding to the maxterms which are not present in the expression. The decimal designation of the squares for maxterms is the same as that for the minterms. A two-variable K-map and the associated maxterms are shown in Figure 6.6.

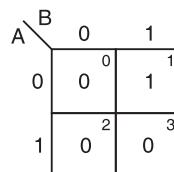


**Figure 6.6** The maxterms of a two-variable K-map.

**EXAMPLE 6.3** Plot the expression  $f = (A + B)(\bar{A} + B)(\bar{A} + \bar{B})$  on the K-map.

**Solution**

The given expression in terms of maxterms is  $f = \prod M (0, 2, 3)$ . The corresponding K-map is shown in Figure 6.7.

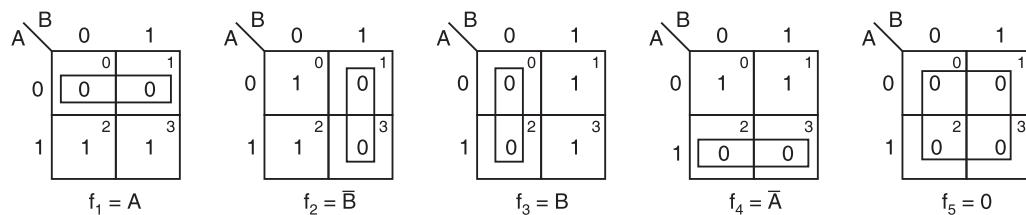


**Figure 6.7** Example 6.3: K-Map for  $\prod M (0, 2, 3)$ .

#### 6.2.4 Minimization of POS Expressions

To obtain the minimal expression in POS form, map the given POS expression on to the K-map and combine the adjacent 0s into as large squares as possible. Read the squares putting the complemented variable if its value remains constant as a 1 and the non-complemented variable if its value remains constant as a 0 along the entire square (ignoring the variables which do not remain constant throughout the square) and then write them as a sum term.

Various maxterm combinations and the corresponding reduced expressions are shown in Figure 6.8. In Figure 6.8,  $f_1$  is read as  $A$  because  $A$  remains constant as a 0 throughout the square, and  $B$  changes from a 0 to a 1.  $f_2$  is read as  $\bar{B}$  because  $B$  remains constant along the square as a 1 and  $A$  changes from a 0 to a 1.  $f_3$  is read as  $B$  because both the variables are changing along the square.



**Figure 6.8** The possible maxterm groupings in a two-variable K-map.

**EXAMPLE 6.4** Reduce the expression  $f = (A + B)(A + \bar{B})(\bar{A} + \bar{B})$  using mapping.

**Solution**

The given expression in terms of maxterms is  $f = \prod M(0, 1, 3)$ . Figure 6.9 shows the K-map for  $f$  and its reduction. It requires two gate inputs for realization of the reduced expression as shown in the figure.

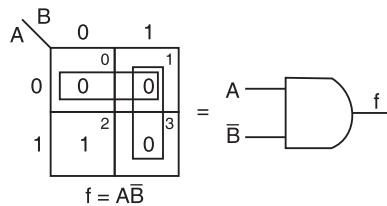


Figure 6.9 Example 6.4: K-map in POS form and logic diagram.

In the given expression, the maxterm  $M_2$  is absent. This is indicated by a 1 on the K-map. The corresponding SOP expression is  $\sum m_2$  or  $A\bar{B}$ . This realization is the same as that for the POS form.

### 6.3 THREE-VARIABLE K-MAP

A function in three variables ( $A, B, C$ ) expressed in the standard SOP form can have eight possible combinations:  $\bar{A}\bar{B}\bar{C}$ ,  $\bar{A}\bar{B}C$ ,  $\bar{A}B\bar{C}$ ,  $\bar{A}BC$ ,  $A\bar{B}\bar{C}$ ,  $A\bar{B}C$ ,  $AB\bar{C}$ , and  $ABC$ . Each one of these combinations designated by  $m_0, m_1, m_2, m_3, m_4, m_5, m_6$ , and  $m_7$ , respectively, is called a minterm.  $A$  is the MSB of the minterm designator and  $C$  is the LSB.

In the standard POS form, the eight possible combinations are:  $A + B + C$ ,  $A + B + \bar{C}$ ,  $A + \bar{B} + C$ ,  $A + \bar{B} + \bar{C}$ ,  $\bar{A} + B + C$ ,  $\bar{A} + B + \bar{C}$ ,  $\bar{A} + \bar{B} + C$ , and  $\bar{A} + \bar{B} + \bar{C}$ . Each one of these combinations designated by  $M_0, M_1, M_2, M_3, M_4, M_5, M_6$ , and  $M_7$ , respectively, is called a maxterm.  $A$  is the MSB of the maxterm designator and  $C$  is the LSB.

A three-variable K-map has, therefore,  $8 (= 2^3)$  squares or cells, and each square on the map represents a minterm or maxterm as shown in Figure 6.10a and Figure 6.10b. The small number on the top right corner of each cell indicates the minterm or maxterm designation.

		BC	00	01	11	10
		A	0	1	3	2
0	0	$\bar{A}\bar{B}\bar{C}$ ( $m_0$ )	$\bar{A}\bar{B}C$ ( $m_1$ )	$\bar{A}B\bar{C}$ ( $m_3$ )	$\bar{A}B\bar{C}$ ( $m_2$ )	
	1	$A\bar{B}\bar{C}$ ( $m_4$ )	$A\bar{B}C$ ( $m_5$ )	$ABC$ ( $m_7$ )	$AB\bar{C}$ ( $m_6$ )	

(a) Minterms

		BC	00	01	11	10
		A	0	1	3	2
0	0	$A + B + C$ ( $M_0$ )	$A + B + \bar{C}$ ( $M_1$ )	$A + \bar{B} + \bar{C}$ ( $M_3$ )	$A + \bar{B} + C$ ( $M_2$ )	
	1	$\bar{A} + B + C$ ( $M_4$ )	$\bar{A} + B + \bar{C}$ ( $M_5$ )	$\bar{A} + \bar{B} + \bar{C}$ ( $M_7$ )	$\bar{A} + \bar{B} + C$ ( $M_6$ )	

(b) Maxterms

Figure 6.10 The three-variable K-map.

The binary numbers along the top of the map indicate the condition of  $B$  and  $C$  for each column. The binary number along the left side of the map against each row indicates the condition of  $A$  for that row. For example, the binary number 01 on top of the second column in Figure 6.10a

indicates that the variable B appears in complemented form and the variable C in non-complemented form in all the minterms in that column. The binary number 0 on the left of the first row indicates that the variable A appears in complemented form in all the minterms in that row. Similarly, the binary number 01 on top of the second column in Figure 6.10b indicates that the variable B appears in non-complemented form and the variable C in complemented form in all the maxterms in that column. The binary number 0 on the left of the first row indicates that the variable A appears in non-complemented form in all the maxterms in that row. Observe that the binary numbers along the top of the K-map are not in normal binary order. They are, in fact, in the Gray code. This is to ensure that two physically adjacent squares are really adjacent, i.e. their minterms or maxterms differ by only one variable.

**EXAMPLE 6.5** Map the expression  $f = \bar{A}\bar{B}C + A\bar{B}C + \bar{A}\bar{B}\bar{C} + AB\bar{C} + ABC$ .

**Solution**

In the given expression, the minterms are:  $\bar{A}\bar{B}C = 001 = m_1$ ;  $A\bar{B}C = 101 = m_5$ ;  $\bar{A}\bar{B}\bar{C} = 010 = m_2$ ;  $AB\bar{C} = 110 = m_6$ ;  $ABC = 111 = m_7$ . So the expression is  $f = \sum m (1, 5, 2, 6, 7) = \sum m (1, 2, 5, 6, 7)$ . The corresponding K-map is shown in Figure 6.11.

		BC	00	01	11	10	
		A	0	0	1	0	1
			0	4	5	7	6
0	0		0	1	0	1	
1	1		0	1	1	1	

Figure 6.11 Example 6.5: K-map in SOP form.

**EXAMPLE 6.6** Map the expression  $f = (A + B + C)(\bar{A} + B + \bar{C})(\bar{A} + \bar{B} + \bar{C})(A + \bar{B} + \bar{C})(\bar{A} + \bar{B} + C)$ .

**Solution**

In the given expression the maxterms are:  $A + B + C = 000 = M_0$ ;  $\bar{A} + B + \bar{C} = 101 = M_5$ ;  $\bar{A} + \bar{B} + \bar{C} = 111 = M_7$ ;  $A + \bar{B} + \bar{C} = 011 = M_3$ ;  $\bar{A} + \bar{B} + C = 110 = M_6$ .

So the expression is  $f = \prod M (0, 5, 7, 3, 6) = \prod M (0, 3, 5, 6, 7)$ . The mapping of the expression is shown in Figure 6.12.

		BC	00	01	11	10	
		A	0	0	1	0	1
			0	1	0	0	0
0	0		0	1	0	1	
1	1		1	0	0	0	

Figure 6.12 Example 6.6: K-map in POS form.

### 6.3.1 Minimization of SOP and POS Expressions

For reducing the Boolean expressions in SOP (POS) form plotted on the K-map, look at the 1s (0s) present on the map. These represent the minterms (maxterms). Look for the minterms

(maxterms) adjacent to each other, in order to combine them into larger squares. Combining of adjacent squares in a K-map containing 1s (or 0s) for the purpose of simplification of a SOP (or POS) expression is called *looping*. Some of the minterms (maxterms) may have many adjacencies. Always start with the minterm (maxterm) with the least number of adjacencies and try to form as large a square as possible. The larger squares must form a geometric square or rectangle. They can be formed even by wrapping around, but cannot be formed by using diagonal configurations. Next consider the minterm (maxterms) with next to the least number of adjacencies and form as large a square as possible. Continue this till all the minterms (maxterms) are taken care of. A minterm (maxterm) can be combined any number of times if it helps in reduction, i.e. A minterm (maxterm) can be part of any number of squares if it is helpful in reduction. Read the minimal expression from the K-map, corresponding to the squares formed. There can be more than one minimal expression.

Two squares are said to be adjacent to each other (since the binary designations along the top of the map and those along the left side of the map are in Gray code), if they are physically adjacent to each other, or can be made adjacent to each other by wrapping around. For squares to be combinable into bigger squares it is essential but not sufficient that their minterm designations must differ by a power of two.

From the above, we can outline the generalized procedure to simplify the Boolean expressions as follows:

1. Plot the K-map and place 1s (0s) corresponding to the minterms (maxterms) of the SOP (POS) expression.
2. Check the K-map for 1s (0s) which are not adjacent to any other 1 (0). They are isolated minterms (maxterms). They are to be read as they are because they cannot be combined even into a 2-square.
3. Check for those 1s (0s) which are adjacent to only one other 1 (0) and make them pairs (2 squares).
4. Check for quads (4 squares) and octets (8 squares) of adjacent 1s (0s) even if they contain some 1s (0s) which have already been combined. They must geometrically form a square or a rectangle.
5. Check for any 1s (0s) that have not been combined yet and combine them into bigger squares if possible.
6. Form the minimal expression by summing (multiplying) the product (sum) terms of all the groups.

### 6.3.2 Reading the K-maps

While reading the reduced K-map in SOP (POS) form, the variable which remains constant as 0 along the square is written as the complemented (non-complemented) variable and the one which remains constant as 1 along the square is written as non-complemented (complemented) variable and the term as a product (sum) term. All the product (sum) terms are added (multiplied).

Some possible combinations of minterms and the corresponding minimal expressions read from the K-maps are shown in Figure 6.13. Here  $f_6$  is read as 1, because along the 8-square no variable remains constant.  $f_5$  is read as  $\bar{A}$ , because, along the 4-square formed by  $m_0, m_1, m_2$ , and

$m_3$ , the variables B and C are changing, and A remains constant as a 0. Algebraically, we have

$$\begin{aligned} f_5 &= m_0 + m_1 + m_2 + m_3 = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC \\ &= \bar{A}\bar{B}(\bar{C} + C) + \bar{A}B(C + \bar{C}) \\ &= \bar{A}\bar{B} + \bar{A}B = \bar{A}(\bar{B} + B) = \bar{A} \end{aligned}$$

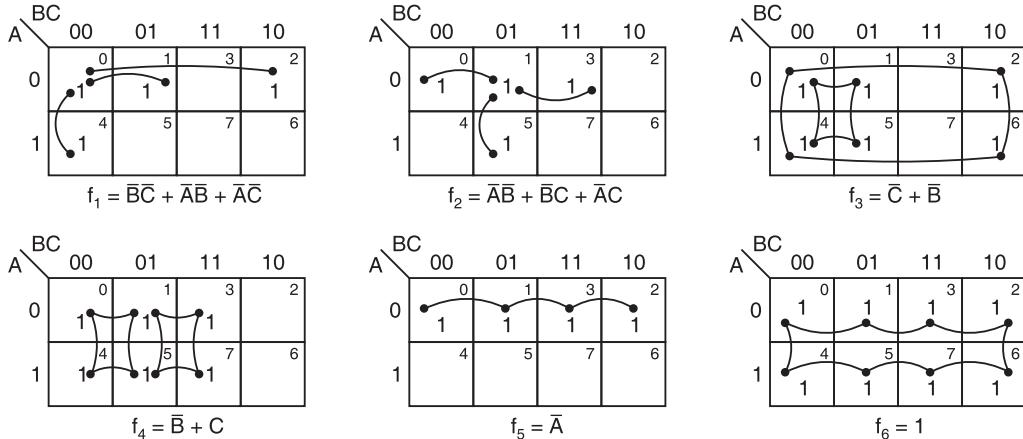


Figure 6.13 Some possible combinations of minterms in a three-variable K-map (SOP form).

$f_3$  is read as  $\bar{C} + \bar{B}$ , because in the 4-square formed by  $m_0, m_2, m_6$ , and  $m_4$ , the variables A and B are changing, whereas the variable C remains constant as a 0. So it is read as  $\bar{C}$ . In the 4-square formed by  $m_0, m_1, m_4$  and  $m_5$ , A and C are changing but B remains constant as a 0. So it is read as  $\bar{B}$ . So, the resultant expression for  $f_3$  is the sum of these two, i.e.  $\bar{C} + \bar{B}$ .

$f_1$  is read as  $\bar{B}\bar{C} + \bar{A}\bar{B} + \bar{A}\bar{C}$ , because in the 2-square formed by  $m_0$  and  $m_4$ , A is changing from a 0 to a 1, whereas B and C remain constant as a 0. So, it is read as  $\bar{B}\bar{C}$ . In the 2-square formed by  $m_0$  and  $m_1$ , C is changing from a 0 to a 1, whereas A and B remain constant as a 0. So, it is read as  $\bar{A}\bar{B}$ . In the 2-square formed by  $m_0$  and  $m_2$ , B is changing from a 0 to a 1 whereas A and C remain constant as a 0. So, it is read as  $\bar{A}\bar{C}$ . Therefore, the resultant SOP expression is,  $\bar{B}\bar{C} + \bar{A}\bar{B} + \bar{A}\bar{C}$ .

Some possible maxterm groupings and the corresponding minimal POS expressions read from the K-map are shown in Figure 6.14.

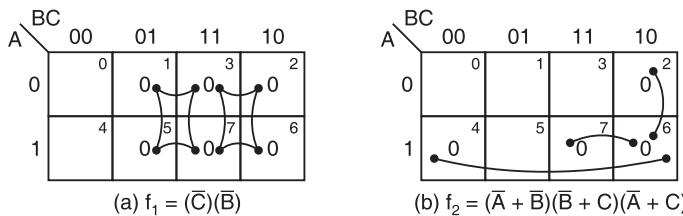


Figure 6.14 Some possible combinations of maxterms in a three-variable K-map (POS form).

In Figure 6.14a, along the 4-square formed by  $M_1, M_3, M_7$  and  $M_5$ , A and B are changing from a 0 to a 1, whereas C remains constant as a 1. So it is read as  $\bar{C}$ . Along the 4-square formed

by  $M_3$ ,  $M_2$ ,  $M_7$  and  $M_6$ , variables A and C are changing from a 0 to a 1, but B remains constant as a 1. So it is read as  $\bar{B}$ . The minimal expression is the product of these two terms, i.e.  $f_1 = (\bar{C})(\bar{B})$ .

In Figure 6.14b, along the 2-square formed by  $M_4$  and  $M_6$ , variable B is changing from a 0 to a 1, while variable A remains constant as a 1 and variable C remains constant as a 0. So, read it as  $\bar{A} + C$ . Similarly, the 2-square formed by  $M_7$  and  $M_6$  is read as  $\bar{A} + \bar{B}$ , while the 2-square formed by  $M_2$  and  $M_6$  is read as  $\bar{B} + C$ . The minimal expression is the product of these three sum terms, i.e.  $f_2 = (\bar{A} + C)(\bar{A} + \bar{B})(\bar{B} + C)$ .

**EXAMPLE 6.7** Reduce the expression  $f = \sum m(0, 2, 3, 4, 5, 6)$  using mapping and implement it in AOI logic as well as in NAND logic.

### Solution

The SOP K-map and its reduction, and the implementation of the minimal expression using AOI logic and the corresponding NAND logic are shown in Figures 6.15a, b, and c respectively.

In the SOP K-map shown in Figure 6.15a, the reduction is done as per the following steps:

1.  $m_5$  has only one adjacency  $m_4$ , so combine  $m_5$  and  $m_4$  into a 2-square. Along this 2-square A remains constant as 1 and B remains constant as 0 but C varies from 0 to 1. So read it as  $A\bar{B}$ .
2.  $m_3$  has only one adjacency  $m_2$ . So combine  $m_3$  and  $m_2$  into a 2-square. Along this 2-square A remains constant as 0 and B remains constant as 1 but C varies from 1 to 0. So read it as  $\bar{A}B$ .
3.  $m_6$  can form a 2-square with  $m_2$  and  $m_4$  can form a 2-square with  $m_0$ , but observe that by wrapping the map from left to right  $m_0$ ,  $m_4$ ,  $m_2$ ,  $m_6$  can form a 4-square. Out of these  $m_2$  and  $m_4$  have already been combined but they can be utilized again. So make it. Along this 4-square, A is changing from 0 to 1 and B is also changing from 0 to 1 but C is remaining constant as 0. So read it as  $\bar{C}$ .
4. Write all the product terms in SOP form. So the minimal SOP expression is

$$f_{\min} = A\bar{B} + \bar{A}B + \bar{C} = \overline{\bar{A}\bar{B} + \bar{A}B + C} = \overline{\bar{A}\bar{B}} \cdot \overline{\bar{A}B} \cdot \overline{C} = \overline{AB} \cdot \overline{A}\bar{B} \cdot C$$

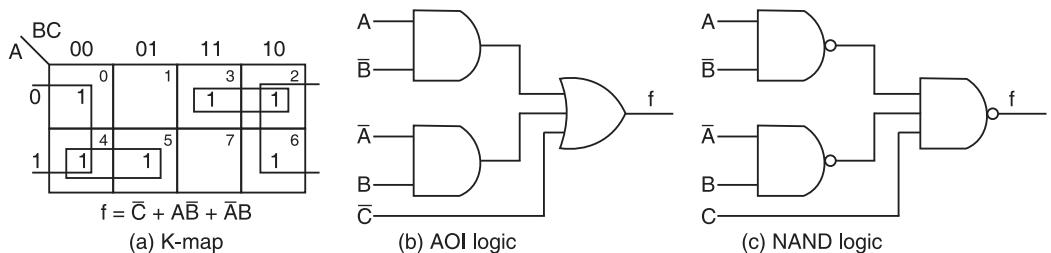


Figure 6.15 Example 6.7: K-map in SOP form, and logic diagrams.

**EXAMPLE 6.8** Reduce the expression  $f = \prod M(0, 1, 2, 3, 4, 7)$  using mapping and implement it in AOI logic as well as in NOR logic.

**Solution**

The K-map and its reduction, and the implementation of the minimal expression using AOI logic and the corresponding NOR logic are shown in Figures 6.16a, b, and c respectively.

In the POS K-map shown in Figure 6.16a the reduction is done as per the following steps:

1.  $M_4$  has only one adjacency  $M_0$ . So combine  $M_4$  and  $M_0$  into a 2-square. Along this 2-square, A varies from 0 to 1 but B and C remain constant as 0. So read it as  $(B + C)$ .
2.  $M_7$  has only one adjacency  $M_3$ . So combine  $M_7$  and  $M_3$  into a 2-square. Along this 2-square, A varies from 0 to 1, but B and C remain constant as 1. So read it as  $(\bar{B} + \bar{C})$ .
3.  $M_0$ ,  $M_1$ ,  $M_3$ , and  $M_2$  form a geometric rectangle. So make it a 4-square. Along this 4-square, B and C vary from 0 to 1, but A remains constant as 0. So read this 4-square as A.
4. Write all the sum terms in POS form. So the minimal POS expression is

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

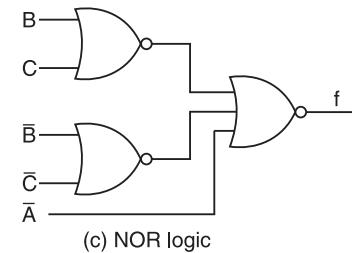
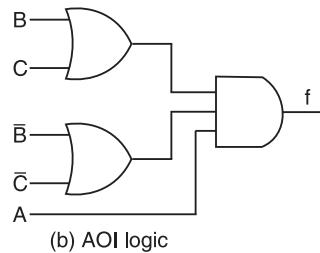
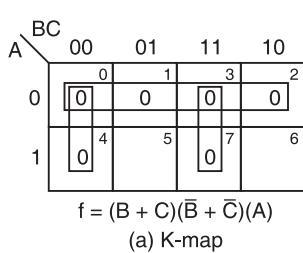


Figure 6.16 Example 6.8.

**EXAMPLE 6.9** Obtain the real minimal expression for  $f = \sum m(1, 2, 4, 6, 7)$  and implement it using universal gates.

**Solution**

In the given SOP expression, minterms  $m_0$ ,  $m_3$ , and  $m_5$  are missing. They, therefore, become the maxterms for the POS expression. So in POS form  $f = \prod M(0, 3, 5)$ .

To obtain the real minimal expression, obtain the minimal expressions in both the SOP and POS forms and then take the minimal of those two minimals. The K-maps in SOP and POS forms, their minimization, and the minimal expressions obtained from them are shown in Figures 6.17a and b respectively.

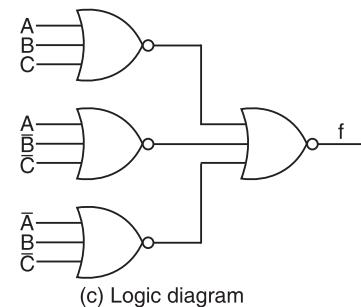
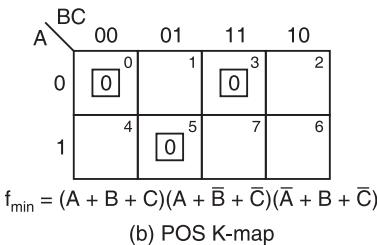
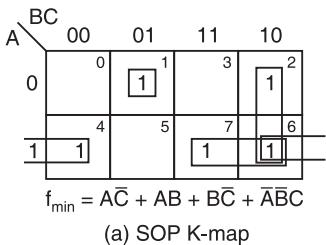


Figure 6.17 Example 6.9.

In the SOP K-map shown in Figure 6.17a, the minimization is done as per the following steps:

1.  $m_1$  has no adjacency. So it cannot be a part of a bigger square. So read it as it is, i.e as  $\bar{A}\bar{B}C$ .
2.  $m_4$  has only one adjacency  $m_6$ . So make a 2-square with  $m_4$  and  $m_6$  and read it as  $A\bar{C}$ .
3.  $m_2$  has only one adjacency  $m_6$ . So make a 2-square with  $m_2$  and  $m_6$  and read it as  $B\bar{C}$ .
4.  $m_7$  has only one adjacency  $m_6$ . So make a 2-square with  $m_7$  and  $m_6$  and read it as  $AB$  (observe that  $m_6$  has been combined thrice).
5. Write all the product terms in SOP form.

So the minimal SOP expression is

$$f_{\min} = \bar{A}\bar{B}C + A\bar{C} + B\bar{C} + AB$$

In the POS K-map no two maxterms are adjacent to each other. So no minimization is possible and they are to be read as they are.  $M_0$  is read as  $(A + B + C)$ .  $M_3$  is read as  $(A + \bar{B} + \bar{C})$ .  $M_5$  is read as  $(\bar{A} + B + \bar{C})$ . So the POS expression is  $f = (A + B + C)(A + \bar{B} + \bar{C})(\bar{A} + B + \bar{C})$ . The SOP form requires 13 gate inputs, whereas the POS form requires 12. So, the POS form is preferred. Thus, the real minimal expression is

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

The logic diagram corresponding to the minimal expression using NOR gates is shown in Figure 6.17c.

**EXAMPLE 6.10** Show the truth table for each of the following functions and find its simplest POS form

- (a)  $f(X, Y, Z) = XY + XZ$
- (b)  $f(X, Y, Z) = \bar{X} + Y\bar{Z}$

#### Solution

The truth tables for the functions, the POS expressions obtained from them, the corresponding K-maps, their simplification and the simplest POS forms are shown in Figures 6.18a and b respectively. In the truth table, 0s represent maxterms and 1s represent minterms. The simplest POS form can be found by using either K-map or algebraically. The minimal expressions from K-map are

$$(i) f_{\min} = X(Y + Z) \quad (ii) f_{\min} = (\bar{X} + Y)(\bar{X} + \bar{Z})$$

Algebraically we have

$$(i) f_{\min} = XY + XZ = X(Y + Z) \quad (ii) f_{\min} = \bar{X} + Y\bar{Z} = (\bar{X} + Y)(\bar{X} + \bar{Z})$$

X	Y	Z	XY	XZ	XY + XZ
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	1	1	0	0	0 + 0 = 0
1	0	0	0	0	0 + 0 = 0
1	0	1	0	1	0 + 1 = 1
1	1	0	1	0	1 + 0 = 1
1	1	1	1	1	1 + 1 = 1

(i) Truth table

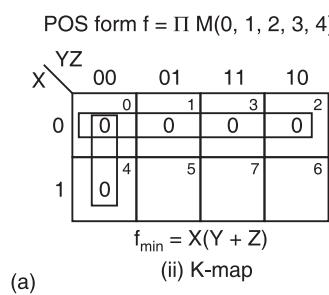


Figure 6.18 Example 6.10 (Contd.)

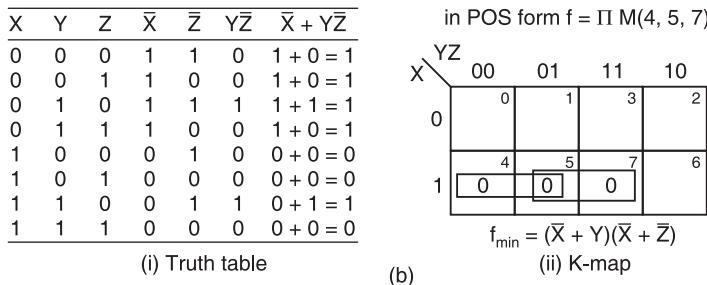


Figure 6.18 Example 6.10.

## 6.4 FOUR-VARIABLE K-MAP

A four-variable (A, B, C, D) expression can have  $2^4 = 16$  possible combinations of input variables such as  $\bar{A}\bar{B}\bar{C}\bar{D}$ ,  $\bar{A}\bar{B}\bar{C}D$ , ..., ABCD with minterm designations  $m_0, m_1, \dots, m_{15}$ , respectively, in the SOP form and as  $A + B + C + D$ ,  $A + B + C + \bar{D}$ , ...,  $\bar{A} + \bar{B} + \bar{C} + \bar{D}$  with maxterm designations  $M_0, M_1, \dots, M_{15}$ , respectively, in the POS form. A four-variable K-map has  $2^4 = 16$  squares or cells and each square on the map represents either a minterm or a maxterm as shown in Figure 6.19.

The binary number designations of the rows and columns are in the Gray code. Here 11 follows 01 and 10 follows 11. This is called *adjacency ordering*. The binary numbers along the top of the map indicate the conditions of C and D along any column and binary numbers along the left side indicate the conditions of A and B along any row. The numbers in the top right corners of the squares indicate the minterm or maxterm designations as usual.

		CD		AB	00	01	11	10
		00	0	$\bar{A} + B + C + D$ ( $M_0$ )	$\bar{A} + B + C + \bar{D}$ ( $M_1$ )	$\bar{A} + B + \bar{C} + \bar{D}$ ( $M_3$ )	$\bar{A} + B + \bar{C} + D$ ( $M_2$ )	
		01	1	$\bar{A} + \bar{B} + C + D$ ( $M_4$ )	$\bar{A} + \bar{B} + C + \bar{D}$ ( $M_5$ )	$\bar{A} + \bar{B} + \bar{C} + \bar{D}$ ( $M_7$ )	$\bar{A} + \bar{B} + \bar{C} + D$ ( $M_6$ )	
		11	2	$\bar{A} + \bar{B} + \bar{C} + \bar{D}$ ( $M_8$ )	$\bar{A} + \bar{B} + \bar{C} + D$ ( $M_9$ )	$\bar{A} + \bar{B} + C + \bar{D}$ ( $M_{13}$ )	$\bar{A} + \bar{B} + C + D$ ( $M_{15}$ )	
		10	3	$A + B + C + D$ ( $M_12$ )	$A + B + C + \bar{D}$ ( $M_{14}$ )	$A + B + \bar{C} + \bar{D}$ ( $M_{11}$ )	$A + B + \bar{C} + D$ ( $M_{10}$ )	
		00	4	$A + \bar{B} + C + D$ ( $M_5$ )	$A + \bar{B} + C + \bar{D}$ ( $M_6$ )	$A + \bar{B} + \bar{C} + \bar{D}$ ( $M_7$ )	$A + \bar{B} + \bar{C} + D$ ( $M_8$ )	
		01	5	$A + B + \bar{C} + D$ ( $M_4$ )	$A + B + \bar{C} + \bar{D}$ ( $M_3$ )	$A + B + C + \bar{D}$ ( $M_2$ )	$A + B + C + D$ ( $M_1$ )	
		11	6	$A + B + \bar{C} + \bar{D}$ ( $M_6$ )	$A + B + C + \bar{D}$ ( $M_7$ )	$A + B + C + D$ ( $M_8$ )	$A + B + \bar{C} + \bar{D}$ ( $M_9$ )	
		10	7	$A + \bar{B} + \bar{C} + \bar{D}$ ( $M_7$ )	$A + \bar{B} + \bar{C} + D$ ( $M_6$ )	$A + \bar{B} + C + \bar{D}$ ( $M_5$ )	$A + \bar{B} + C + D$ ( $M_4$ )	

SOP form
POS form

Figure 6.19 The minterms and maxterms of a four-variable K-map.

Squares which are physically adjacent to each other or which can be made adjacent by wrapping the map around from left to right or top to bottom can be combined to form bigger squares. The bigger squares (2 squares, 4 squares, 8 squares, etc.) must form either a geometric square or rectangle. For the minterms or maxterms to be combinable into bigger squares, it is necessary but not sufficient that their binary designations differ by a power of 2.

Some possible 2-squares are:  $m_0, m_1; m_0, m_2; m_0, m_4; m_0, m_8; m_{13}, m_{12}; m_{13}, m_5; m_{13}, m_9; m_{13}, m_{15}; m_{10}, m_{11}; m_{10}, m_8; m_{10}, m_{14}; m_{10}, m_2$ ; etc.

Some possible 4-squares are:  $m_0, m_1, m_3, m_2; m_0, m_4, m_{12}, m_8; m_0, m_1, m_4, m_5; m_0, m_4, m_2, m_6; m_0, m_1, m_8, m_9; m_5, m_7, m_{13}, m_{15}; m_0, m_2, m_8, m_{10}$ ; etc.

Some possible 8-squares are:  $m_0, m_1, m_3, m_2, m_8, m_9, m_{11}, m_{10}; m_0, m_4, m_{12}, m_8, m_2, m_6, m_{14}, m_{10}; m_4, m_5, m_7, m_6, m_{12}, m_{13}, m_{15}; m_1, m_3, m_5, m_7, m_{13}, m_{15}, m_9, m_{11}$ ; etc.

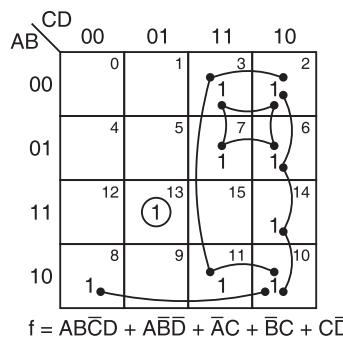
**EXAMPLE 6.11** Reduce using mapping the expression  $f = \sum m(2, 3, 6, 7, 8, 10, 11, 13, 14)$ .

### Solution

On the SOP K-map shown in Figure 6.20 the minimization is done as per the following steps:

1. Start with the minterm with the least number of adjacencies. The minterm  $m_{13}$  has no adjacency. Keep it as it is and read as  $AB\bar{C}\bar{D}$ .
2. The  $m_8$  has only one adjacency,  $m_{10}$ . Expand  $m_8$  into a 2-square with  $m_{10}$  and read the 2-square as  $A\bar{B}\bar{D}$ .
3. The  $m_7$  has two adjacencies,  $m_6$  and  $m_3$ . Observe that,  $m_7, m_6, m_2$ , and  $m_3$  form a geometric square. Hence  $m_7$  can be expanded into a 4-square with  $m_6, m_3$  and  $m_2$ . Read this 4-square as  $\bar{A}C$ .
4. The  $m_{11}$  has 2 adjacencies,  $m_{10}$  and  $m_3$ . Observe that,  $m_{11}, m_{10}, m_3$ , and  $m_2$  form a geometric square on wrapping the K-map from top to bottom. So expand  $m_{11}$  into a 4-square with  $m_{10}, m_3$  and  $m_2$ . Note that  $m_2$  and  $m_3$  have already become a part of the 4-square  $m_7, m_6, m_2$ , and  $m_3$ . But if  $m_{11}$  is expanded only into a 2-square with  $m_{10}$ , only one variable is eliminated. So  $m_2$  and  $m_3$  are used again to make another 4-square with  $m_{11}$  and  $m_{10}$  to eliminate two variables. Read this 4-square as  $\bar{B}C$ .
5. Now only  $m_{14}$  is left uncovered. It can form a 2-square with  $m_6$  or  $m_{10}$  but that eliminates only one variable. Don't do that. See whether it can be expanded into a larger square. Observe that,  $m_2, m_6, m_{14}$ , and  $m_{10}$  form a rectangle. So  $m_{14}$  can be expanded into a 4-square with  $m_2, m_6$ , and  $m_{10}$ . This eliminates two variables. Read this 4-square as  $C\bar{D}$ .
6. Write all the product terms in SOP form. Therefore, the reduced expression is

$$f_{\min} = AB\bar{C}\bar{D} + A\bar{B}\bar{D} + \bar{A}C + \bar{B}C + C\bar{D} \quad (18 \text{ inputs})$$



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

Figure 6.20 Example 6.11: K-map in SOP form.

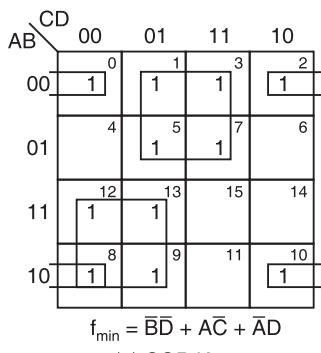
**EXAMPLE 6.12** Reduce using mapping the expression  $f = \sum m(0, 1, 2, 3, 5, 7, 8, 9, 10, 12, 13)$  and implement the real minimal expression in universal logic.

**Solution**

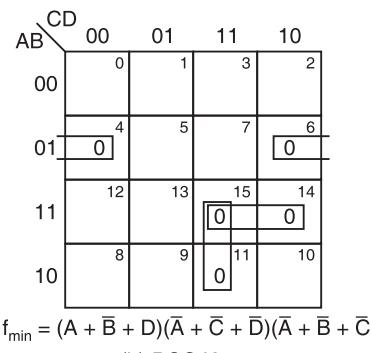
The given expression in the POS form is  $f = \prod M(4, 6, 11, 14, 15)$ . The K-maps for the SOP and POS forms, their reduction, and the reduced expressions obtained from them are shown in Figures 6.21a and b respectively. The SOP form requires 9 gate inputs, whereas the POS form requires 12 gate inputs. So the SOP form of realization is more economical. Now,

$$f_{\min} = \overline{BD} + A\bar{C} + \bar{A}D = \overline{BD} \cdot \overline{AC} \cdot \overline{AD}$$

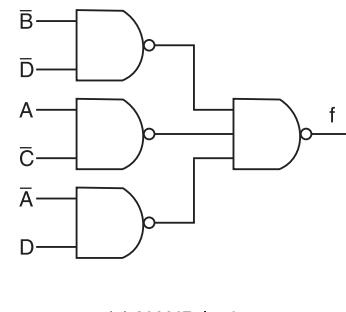
The implementation of the minimal expression using NAND logic is shown in Figure 6.21c.



(a) SOP K-map



(b) POS K-map



(c) NAND logic

**Figure 6.21** Example 6.12.

In the SOP K-map shown in Figure 6.21a, the reduction is done as per the following steps:

Looking at the map you feel like making one 4-square with  $m_0, m_1, m_3, m_2$  and another 4-square with  $m_1, m_5, m_{13}, m_9$  but do not do that. It will be a blunder. Proceed systematically as follows:

1. There are no isolated 1s.
2. There are no 1s which can be combined only into 2-squares. So make no 2-squares.
3.  $m_{10}$  can form a 4-square with  $m_0, m_2, m_8$ . Make it and read it as  $\overline{BD}$ .
4.  $m_{12}$  can form a 4-square with  $m_8, m_9, m_{13}$ . Make it and read it as  $A\bar{C}$ .
5.  $m_7$  can form a 4-square with  $m_5, m_1, m_3$ . Make it and read it as  $\bar{A}D$ .
6. Write all the product terms in SOP form.

So the minimal SOP expression is

$$f_{\min} = \overline{BD} + A\bar{C} + \bar{A}D$$

In the POS K-map shown in Figure 6.21b, the reduction is done as per the following steps:

1. There are no isolated 0s.
2.  $M_4$  can form a 2-square only with  $M_6$ . Make it and read it as  $(A + \bar{B} + D)$ .
3.  $M_{11}$  can form a 2-square only with  $M_{15}$ . Make it and read it as  $(\bar{A} + \bar{C} + \bar{D})$ .

4. Only  $M_{14}$  is left. It can form a 2-square with  $M_{15}$  or  $M_6$ . If you make it with  $M_{15}$ , read it as  $(\bar{A} + \bar{B} + \bar{C})$ .

5. Write all the sum terms in POS form.

So the minimal POS expression is

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

**EXAMPLE 6.13** Reduce using mapping the expression  $f = \prod M(2, 8, 9, 10, 11, 12, 14)$  and implement the real minimal expression in universal logic.

### Solution

The given expression in the SOP form is  $f = \sum m(0, 1, 3, 4, 5, 6, 7, 13, 15)$ . The K-maps for the SOP and POS forms, their reduction, and the reduced expressions obtained from them are shown in Figures 6.22a and b respectively. The SOP form requires 12 gate inputs, whereas the POS form requires only 10 gate inputs. So the POS form is more economical. The implementation of the minimal expressions using NOR gates is given in Figure 6.22c. Now

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

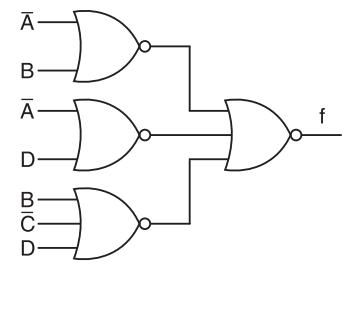
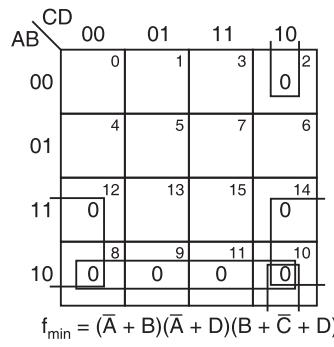
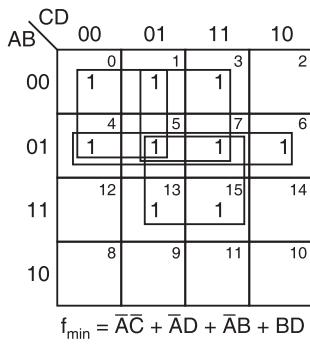


Figure 6.22 Example 6.13.

In the SOP K-map shown in Figure 6.22a, the reduction is done as per the following steps:

1. There are no isolated 1s.
2. There are no 1s which can be combined only into 2-squares. So make no 2-squares.
3.  $m_6$  can form a 4-square with  $m_7, m_5, m_4$ . Make it and read it as  $\bar{A}B$ .
4.  $m_{15}$  can form a 4-square with  $m_{13}, m_5, m_7$ . Make it and read it as  $BD$ .
5.  $m_0$  can form a 4-square with  $m_1, m_4, m_5$ . Make it and read it as  $\bar{A}\bar{C}$ .
6. Only  $m_3$  is left. It can make a 4-square with  $m_1, m_5, m_7$ . Make it and read it as  $\bar{A}\bar{D}$ .
7. Write all the product terms in SOP form.

So the minimal SOP expression is

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

In the POS K-map shown in Figure 6.22b, the reduction is done as per the following steps:

1. There are no isolated 0s.

2.  $M_2$  can form a 2-square with only  $M_{10}$ . Make it and read it as  $(B + \bar{C} + D)$ .
3.  $M_{12}$  and  $M_{14}$  can form a 4-square with  $M_8$ ,  $M_{10}$ . Make it and read it as  $(\bar{A} + D)$ .
4.  $M_9$ , and  $M_{11}$  can form a 4-square with  $M_8$  and  $M_{10}$ . Make it and read it as  $(\bar{A} + B)$ .
5. Write all the sum terms in POS form.

So the minimal POS expression is

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

**EXAMPLE 6.14** Reduce using mapping the following expression and implement the real minimal expression in universal logic.

$$f = \sum m(0, 2, 4, 6, 7, 8, 10, 12, 13, 15)$$

### Solution

The given expression in the POS form is  $f = \prod M(1, 3, 5, 9, 11, 14)$ . The K-maps for the SOP and POS forms, their minimization, and the minimal expressions obtained from them are shown in Figures 6.23a and b respectively. They are:

SOP minimal is

$$f_{\min} = \bar{C}\bar{D} + \bar{A}\bar{D} + \bar{B}\bar{D} + ABD + BCD$$

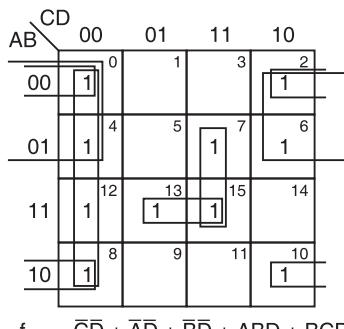
POS minimal is

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

The SOP form requires 17 gate inputs, whereas the POS form requires only 12 gate inputs. So the POS form is more economical. The implementation of the minimal expression using POS logic is shown in Figure 6.23c.

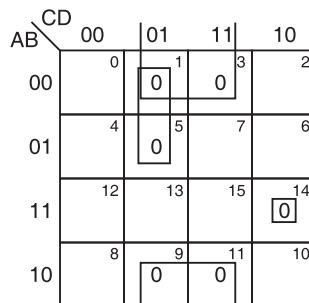
Now

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



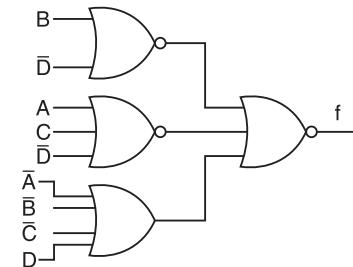
$$f_{\min} = \bar{C}\bar{D} + \bar{A}\bar{D} + \bar{B}\bar{D} + ABD + BCD$$

(a) SOP K-map



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

(b) POS K-map



(c) Logic diagram

Figure 6.23 Example 6.14.

### 6.4.1 Prime Implicants, Essential Prime Implicants, Redundant Prime Implicants and Selective Prime Implicants

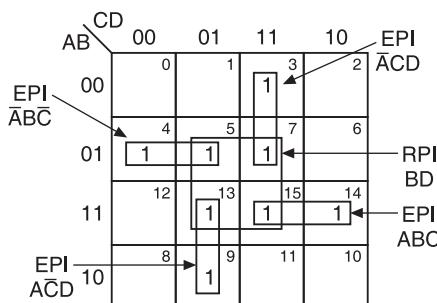
Each square or rectangle made up of the bunch of adjacent minterms is called a *subcube*. Each of these subcubes is called a *prime implicant* (PI). The prime implicant which contains at least one 1

which cannot be covered by any other prime implicant is called an *essential prime implicant* (EPI). The prime implicant whose each 1 is covered at least by one EPI is called a *redundant prime implicant* (RPI). A prime implicant which is neither an essential prime implicant nor a redundant prime implicant is called a *selective prime implicant* (SPI).

The function mapped in Figure 6.24 has a unique MSP comprising EPIs given by

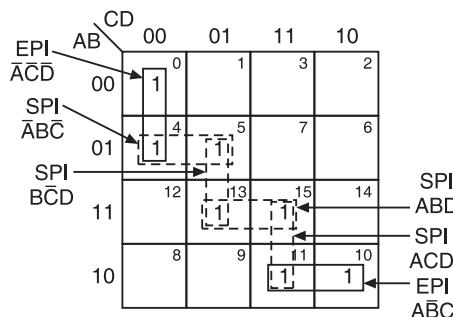
$$F(A, B, C, D) = \bar{A}CD + ABC + \bar{A}\bar{C}D + \bar{A}\bar{B}C$$

The RPI 'BD' may be included without changing the function but the resulting expression would not be in minimal sum of products (MSP) form.



**Figure 6.24** Essential and redundant prime implicants.

In Figure 6.25 representing  $F(A, B, C, D) = \sum m(0, 4, 5, 10, 11, 13, 15)$  SPIs are marked by dotted squares. This shows that the MSP form of a function need not be unique.



**Figure 6.25** Essential and selective prime implicants.

For the function mapped in Figure 6.25, the MSP form is obtained by including two EPIs and selecting a set of SPIs to cover the remaining uncovered minterms 5, 13, 15. These can be covered in the following ways.

- (a) (4, 5) and (13, 15) ...  $\bar{A}\bar{B}\bar{C} + ABD$
- (b) (5, 13) and (13, 15) ...  $B\bar{C}D + ABD$
- (c) (5, 13) and (15, 11) ...  $B\bar{C}D + ACD$

$$\therefore F(A, B, C, D) = \bar{A}\bar{C}D + \bar{A}\bar{B}C \dots \text{EPIs} + \bar{A}\bar{B}\bar{C} + ABD$$

or  $F(A, B, C, D) = \bar{A}\bar{C}D + \bar{A}\bar{B}C \dots \text{EPIs} + B\bar{C}D + ABD$

or  $F(A, B, C, D) = \overline{A}\overline{C}\overline{D} + A\overline{B}C \dots \text{EPIs} + B\overline{C}D + ACD$

Thus this function has three different MSP forms.

#### 6.4.2 False Prime Implicants, Essential False Prime Implicants, Redundant False Prime Implicants and Selective False Prime Implicants

The maxterms are called *false minterms*. The prime implicants obtained by using the maxterms are called *false prime implicants* (FPIs). The FPI which contains at least one 0 which cannot be covered by any other FPI is called an *essential false prime implicant* (EFPI).

The function

$$\begin{aligned} F(A, B, C, D) &= \Sigma m(0, 1, 2, 3, 4, 8, 12) \\ &= \Pi M(5, 6, 7, 9, 10, 11, 13, 14, 15) \\ F_{\min} &= (\overline{B} + \overline{C})(\overline{A} + \overline{C})(\overline{A} + \overline{D})(\overline{B} + \overline{D}) \end{aligned}$$

In the mapping shown in Figure 6.26 all the FPIs are EFPIs as each of them contains at least one 0 which cannot be covered by any other FPI.

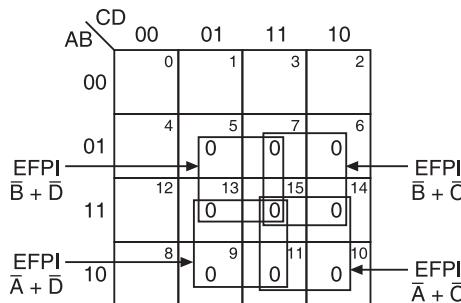


Figure 6.26 Essential false prime implicants.

Consider the mapping shown in Figure 6.27 for the function

$$F(A, B, C, D) = \Sigma m(3, 4, 5, 7, 9, 13, 14, 15) = \Pi M(0, 1, 2, 6, 8, 10, 11, 12)$$

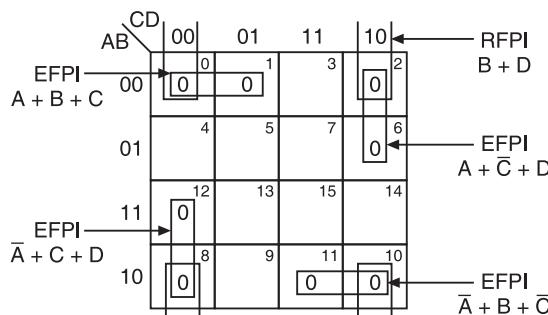


Figure 6.27 Essential and redundant false prime implicants.

The four FPIs in Figure 6.27 ( $A + B + C$ ), ( $A + \overline{C} + D$ ), ( $\overline{A} + C + D$ ), ( $\overline{A} + B + \overline{C}$ ) are all essential FPIs because each one of them contains at least one 0 which cannot be covered by any

other FPI. The four corner 0s form the largest cluster of adjacent 0s, which is an FPI whose 0s are covered by essential FPIs and hence is a redundant false prime implicant (RFPI).

Look at Figure 6.28 which maps

$$\begin{aligned} F(A, B, C, D) &= \Sigma m(0, 4, 5, 10, 11, 13, 15) \\ &= \Pi M(1, 2, 3, 6, 7, 8, 9, 12, 14) \end{aligned}$$

The function has in all seven FPIs marked in Figure 6.28. The FPI ( $A + \bar{C}$ ) is an essential FPI as it contains 0s at locations 2 and 7 which cannot be covered by any other FPI. The remaining six FPIs are all SFPIs. As the EFPi covers the 0s at locations 2, 3, 6, 7 we must now select a minimal set of SFPIs to cover the remaining five 0s at locations 1, 8, 9, 12, 14. The answer is not unique. One possible solution comprising EFPI and three SFPIs is

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

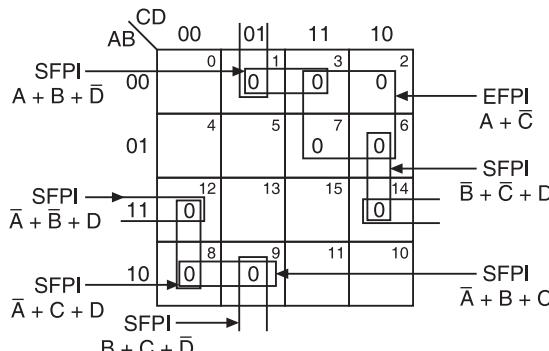


Figure 6.28 Selective false prime implicants.

**EXAMPLE 6.15** (a) Differentiate between a prime implicant and a non-prime implicant, and an essential prime implicant and a non-essential prime implicant.

(b) Reduce the following function using K-map and identify prime implicants and essential prime implicants:

$$F = \Sigma m(0, 1, 2, 3, 6, 7, 13, 15)$$

(c) Reduce the following function using K-map and identify the prime implicant and non-prime implicant.

$$F = \Sigma m(2, 3, 6, 7, 10, 11, 12)$$

### Solution

(a) A prime implicant is a square or rectangle made up of the bunch of adjacent minterms. It is also called a subcube. A non-prime implicant is a minterm which does not have any adjacent minterms, i.e. which cannot form a part of a bigger square. An essential prime implicant is a prime implicant which contains at least one 1 which cannot be covered by any other prime implicant. A non-essential prime implicant is a selective prime implicant. It is neither essential nor redundant. It may or may not appear in the minimal expression.

(b) The K-map for  $F = \Sigma m(0, 1, 2, 3, 6, 7, 13, 15)$  shown in Figure 6.29a has three essential prime implicants (EPIs), and one redundant prime implicant (RPI). The three essential prime implicants cover all the minterms. So the minimal expression is

$$f_{\min} = \bar{A}\bar{B} + \bar{A}C + ABD \text{ (10 gate inputs)}$$

(c) The K-map for  $F = \Sigma m(2, 3, 6, 7, 10, 11, 12)$  shown in Figure 6.29b has two essential prime implicants and one non-prime implicant. All the three together cover all the minterms. So the minimal expression is

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

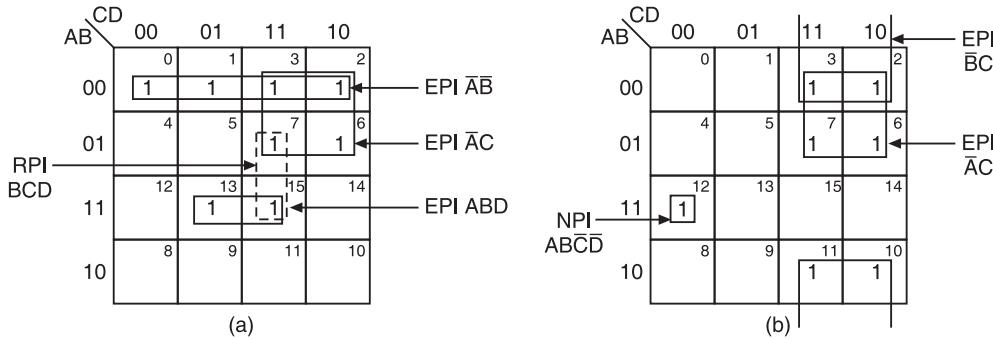


Figure 6.29 Example 6.15: K-maps in SOP form.

## 6.5 FIVE-VARIABLE K-MAP

A five-variable (A, B, C, D, E) expression can have  $2^5 = 32$  possible combinations of input variables such as  $\bar{A}\bar{B}\bar{C}\bar{D}\bar{E}$ ,  $\bar{A}\bar{B}\bar{C}\bar{D}E$ , ..., ABCDE, with minterm designations  $m_0, m_1, \dots, m_{31}$ , respectively, in SOP form and as  $A + B + C + D + E$ ,  $A + B + C + D + \bar{E}$ , ...,  $\bar{A} + \bar{B} + \bar{C} + \bar{D} + \bar{E}$ , with maxterm designations  $M_0, M_1, \dots, M_{31}$ , respectively, in POS form. The 32 squares of the K-map are divided into 2 blocks of 16 squares each. The left block represents minterms from  $m_0$  to  $m_{15}$  in which A is a 0, and the right block represents minterms from  $m_{16}$  to  $m_{31}$  in which A is 1. The five-variable K-map may contain 2-squares, 4-squares, 8-squares, 16-squares, or 32-square involving these two blocks. Squares are also considered adjacent in these two blocks, if when superimposing one block on top of another, the squares coincide with one another.

Some possible 2-squares in a five-variable map are  $m_0, m_{16}$ ;  $m_2, m_{18}$ ;  $m_5, m_{21}$ ;  $m_{15}, m_{31}$ ;  $m_{11}, m_{27}$ .

Some possible 4-squares are  $m_0, m_2, m_{16}, m_{18}$ ;  $m_0, m_1, m_{16}, m_{17}$ ;  $m_0, m_4, m_{16}, m_{20}$ ;  $m_{13}, m_{15}, m_{29}, m_{31}$ ;  $m_5, m_{13}, m_{21}, m_{29}$ .

Some possible 8-squares are  $m_0, m_1, m_3, m_2, m_{16}, m_{17}, m_{19}, m_{18}$ ;  $m_0, m_4, m_{12}, m_8, m_{16}, m_{20}, m_{28}, m_{24}$ ;  $m_5, m_7, m_{13}, m_{15}, m_{21}, m_{23}, m_{29}, m_{31}$ .

The squares are read by dropping out the variables which change. Some possible groupings shown in Figure 6.30 are read as follows.

- |   |  |
|---|--|
| (a) $m_0, m_{16} = \bar{B}\bar{C}\bar{D}\bar{E}$                          | $M_0, M_{16} = B + C + D + E$  |
| (b) $m_2, m_{18} = \bar{B}\bar{C}D\bar{E}$                                | $M_2, M_{18} = B + C + \bar{D} + E$  |
| (c) $m_4, m_6, m_{20}, m_{22} = \bar{B}C\bar{E}$                          | $M_4, M_6, M_{20}, M_{22} = B + \bar{C} + E$                                   |
| (d) $m_5, m_7, m_{13}, m_{15}, m_{21}, m_{23}, m_{29}, m_{31} = CE$       | $M_5, M_7, M_{13}, M_{15}, M_{21}, M_{23}, M_{29}, M_{31} = \bar{C} + \bar{E}$ |
| (e) $m_8, m_9, m_{10}, m_{11}, m_{24}, m_{25}, m_{26}, m_{27} = B\bar{C}$ | $M_8, M_9, M_{10}, M_{11}, M_{24}, M_{25}, M_{26}, M_{27} = \bar{B} + C$       |

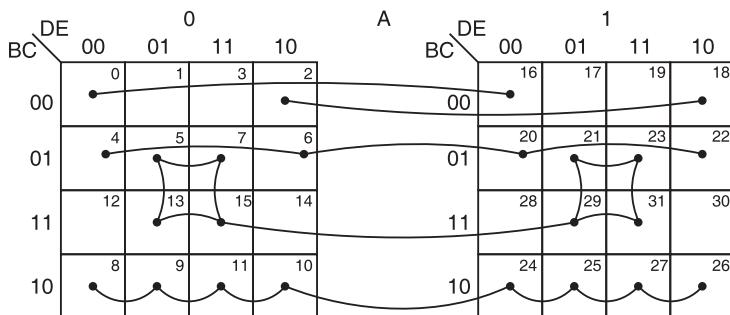


Figure 6.30 Some possible groupings in a five-variable K-map.

**EXAMPLE 6.16** Reduce the following expression in SOP and POS forms using mapping:

$$f = \sum m(0, 2, 3, 10, 11, 12, 13, 16, 17, 18, 19, 20, 21, 26, 27)$$

**Solution**

The given expression in POS form is

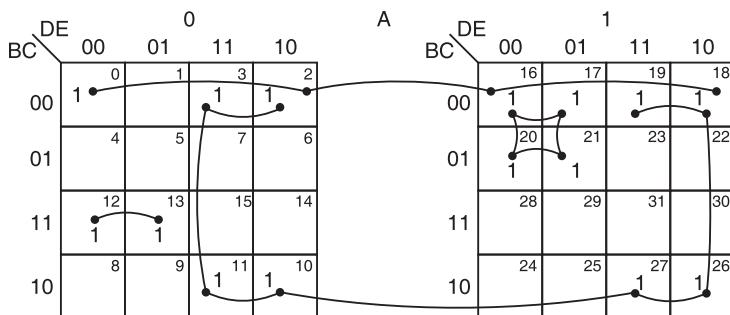
$$f = \prod M(1, 4, 5, 6, 7, 8, 9, 14, 15, 22, 23, 24, 25, 28, 29, 30, 31)$$

The real minimal expression is the minimal of the SOP and POS forms. In the SOP K-map shown in Figure 6.31, the reduction is done as per the following steps:

1. There are no isolated 1s.
2.  $m_{12}$  can go only with  $m_{13}$ . Form a 2-square which is read as  $\bar{A}BC\bar{D}$ .
3.  $m_0$  can go with  $m_2$ ,  $m_{16}$  and  $m_{18}$ . So, form a 4-square which is read as  $\bar{B}\bar{C}\bar{E}$ .
4.  $m_{20}$ ,  $m_{21}$ ,  $m_{17}$  and  $m_{16}$  form a 4-square which is read as  $A\bar{B}\bar{D}$ .
5.  $m_2$ ,  $m_3$ ,  $m_{18}$ ,  $m_{19}$ ,  $m_{10}$ ,  $m_{11}$ ,  $m_{26}$ , and  $m_{27}$  form an 8-square which is read as  $\bar{C}D$ .
6. Write all the product terms in SOP form.

So the minimal SOP expression is

$$f_{\min} = \bar{A}BC\bar{D} + \bar{B}\bar{C}\bar{E} + A\bar{B}\bar{D} + \bar{C}D \text{ (16 inputs)}$$



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

Figure 6.31 Example 6.16: K-map in SOP form.

In the POS K-map shown in Figure 6.32, the reduction is done as per the following steps:

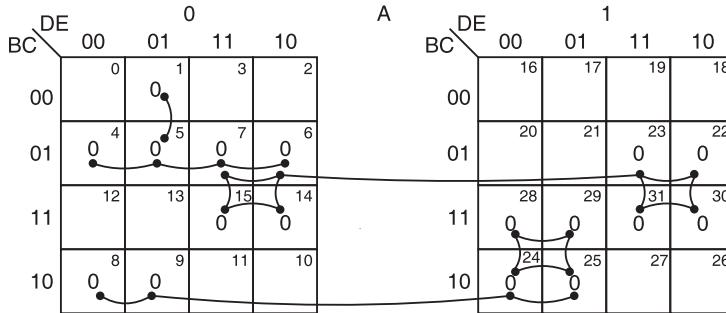
1. There are no isolated 0s.

2.  $M_1$  can go only with  $M_5$  or  $M_9$ . Make a 2-square with  $M_5$ , which is read as  $(A + B + D + \bar{E})$ .
3.  $M_4$  can go with  $M_5$ ,  $M_7$ , and  $M_6$  to form a 4-square, which is read as  $(A + B + \bar{C})$ .
4.  $M_8$  can go with  $M_9$ ,  $M_{24}$ , and  $M_{25}$  to form a 4-square, which is read as  $(\bar{B} + C + D)$ .
5.  $M_{28}$  can go with  $M_{29}$ ,  $M_{24}$ , and  $M_{25}$  to form a 4-square, which is read as  $(\bar{A} + \bar{B} + D)$ .
6.  $M_{30}$  can make a 4-square with  $M_{31}$ ,  $M_{29}$ , and  $M_{28}$  or with  $M_{31}$ ,  $M_{14}$ , and  $M_{15}$  or with  $M_{31}$ ,  $M_{22}$  and  $M_{23}$ . Don't do that. Note that it can make an 8-square with  $M_{31}$ ,  $M_{23}$ ,  $M_{22}$ ,  $M_6$ ,  $M_7$ ,  $M_{14}$  and  $M_{15}$ , which is read as  $(\bar{C} + \bar{D})$ .
7. Write all the sum terms in POS form. So the minimal POS expression is

$$F_{\min} = (A + B + D + \bar{E})(A + B + \bar{C})(\bar{B} + C + D)(\bar{A} + \bar{B} + D)(\bar{C} + \bar{D}) \quad (20 \text{ inputs})$$

The SOP form requires a less number of gate inputs. The real minimal expression is, therefore,

$$f_{\min} = \overline{ABCD} + \overline{BCE} + \overline{ABD} + \overline{CD}$$



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

**Figure 6.32** Example 6.16: K-map in POS form.

**EXAMPLE 6.17** Minimize in SOP and POS forms on the map the 5-variable function

$$F = \sum m(0, 1, 4, 5, 6, 13, 14, 15, 22, 24, 25, 28, 29, 30, 31)$$

Implement the minimal expression using NAND logic.

**Solution**

The given function in POS form is

$$F = \prod M(2, 3, 7, 8, 9, 10, 11, 12, 16, 17, 18, 19, 20, 21, 23, 26, 27)$$

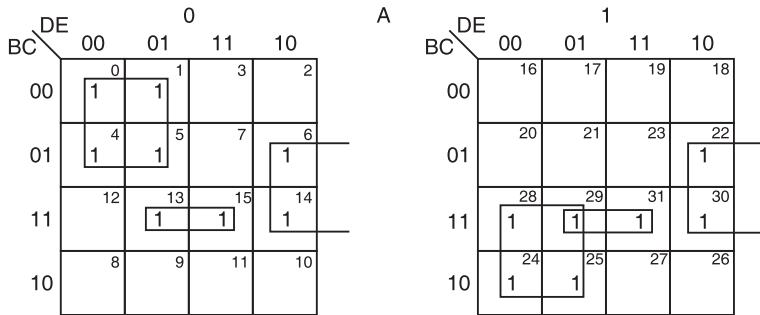
The K-maps in SOP and POS forms, their minimization, and the minimal expressions obtained from them are shown in Figures 6.33a and b respectively. The SOP form requires 16 gate inputs, whereas the POS form requires 20 gate inputs. So the SOP form gives the real minimal, and hence it is implemented in NAND logic as shown in Figure 6.33c.

SOP minimal is

$$f_{\min} = \overline{ABD} + CD\bar{E} + BCE + ABD = \overline{(\overline{ABD})} \overline{(\overline{CD\bar{E}})} \overline{(BCE)} \overline{(\overline{ABD})}$$

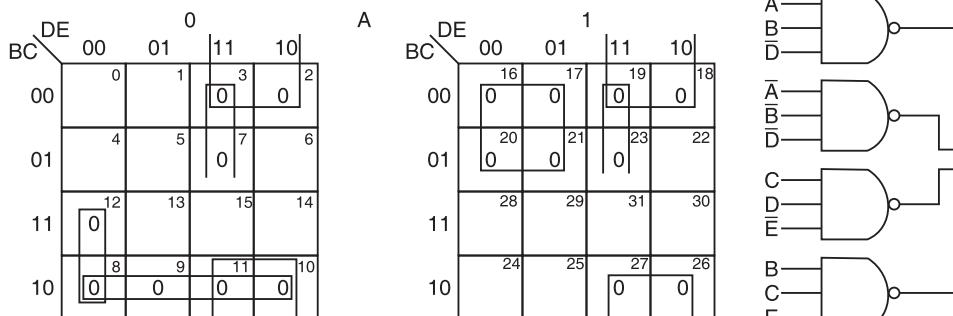
POS minimal is

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

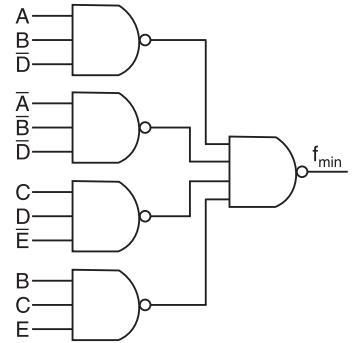


$$f_{min} = \overline{A}\overline{B}D + CD\overline{E} + BCE + AB\overline{D} = (\overline{A}\overline{B}D)(CD\overline{E})(BCE)(AB\overline{D})$$

(a) SOP K-map



(b) POS K-map



(c) Logic diagram

**Figure 6.33** Example 6.17.

**EXAMPLE 6.18** Simplify the Boolean function using K-map in SOP and POS forms:

$$F = \sum m(0, 1, 2, 4, 7, 8, 12, 14, 15, 16, 17, 18, 20, 24, 28, 30, 31)$$

**Solution**

The given function in POS form is

$$F = \prod M(3, 5, 6, 9, 10, 11, 13, 19, 21, 22, 23, 25, 26, 27, 29)$$

The K-maps in SOP and POS forms, their minimization and the minimal expressions obtained from them are shown in Figures 6.34a and b respectively. The SOP form requires 20 gate inputs and the POS form requires 26 gate inputs. So the SOP form gives the real minimal.

SOP minimal is

$$f_{min} = \overline{D}\overline{E} + BCD + \overline{B}\overline{C}\overline{E} + \overline{B}\overline{C}\overline{D} + \overline{A}CDE$$

POS minimal is

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

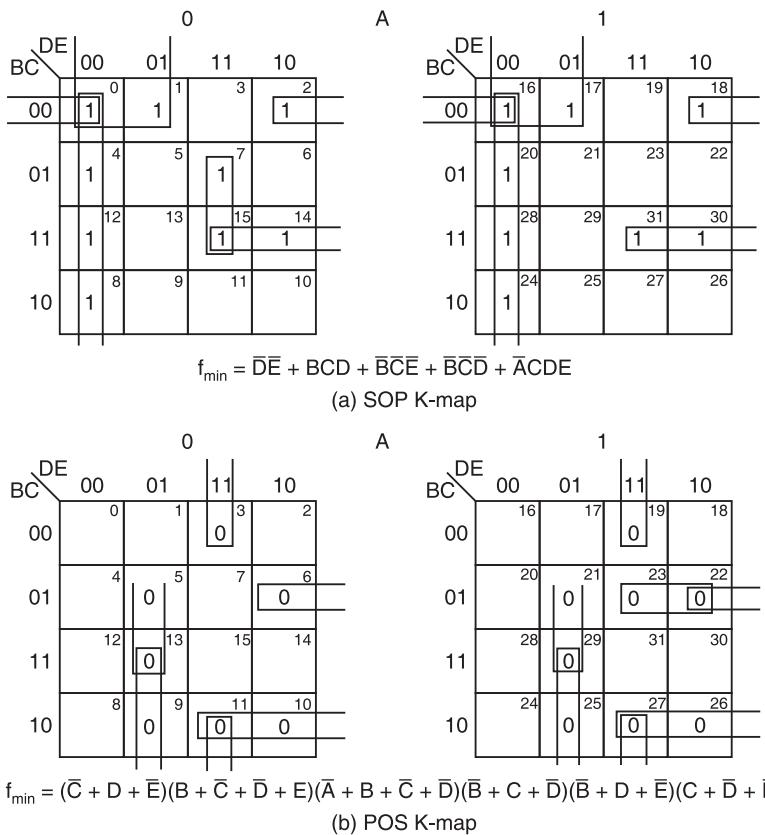


Figure 6.34 Example 6.18.

## 6.6 SIX-VARIABLE K-MAP

A six-variable ( $A, B, C, D, E, F$ ) expression can have  $2^6 = 64$  possible combinations of input variables such as  $\bar{A}\bar{B}\bar{C}\bar{D}\bar{E}\bar{F}$ ,  $\bar{A}\bar{B}\bar{C}\bar{D}\bar{E}F$ , ...,  $ABCDEF$ , with minterm designations  $m_0, m_1, \dots, m_{63}$ , respectively, in the SOP form and as  $(A + B + C + D + E + F)$ ,  $(A + B + C + D + E + \bar{F})$ , ...,  $(\bar{A} + \bar{B} + \bar{C} + \bar{D} + \bar{E} + \bar{F})$ , with maxterm designations  $M_0, M_1, \dots, M_{63}$ , respectively, in the POS form. The 64 squares of the K-map are divided into four blocks of 16 squares each. Each square on the map represents a minterm or a maxterm. The values of  $A$  and  $B$  remain constant for all minterms (maxterms) in each block. The top left block represents minterms ( $m_0$  to  $m_{15}$ ) in which  $A$  is a 0 and  $B$  is a 0. The top right block represents minterms ( $m_{16}$  to  $m_{31}$ ) in which  $A$  is a 0 and  $B$  is a 1. The bottom left block represents minterms ( $m_{32}$  to  $m_{47}$ ) in which  $A$  is a 1 and  $B$  is a 0. The bottom right block represents minterms ( $m_{48}$  to  $m_{63}$ ) in which  $A$  is a 1 and  $B$  is a 1.

The six-variable map may contain 2-squares, 4-squares, 8-squares, 16-squares, 32-squares or a 64-square involving these four blocks. Squares are considered adjacent in two blocks, if upon superimposing one block on top of another block, that is, above or below or beside the first block, the squares coincide with one another. Diagonal elements like  $m_{10}, m_{58}; m_{15}, m_{63}; m_{18}, m_{34}; m_{29}, m_{45}$  are not adjacent to each other.

Some possible 2-squares are:  $m_0, m_{16}; m_{10}, m_{42}; m_{16}, m_{48}; m_7, m_{23}; m_7, m_{39}; m_{23}, m_{55}; m_{47}, m_{63}$ ; etc.

Some possible 4-squares are:  $m_0, m_{16}, m_{32}, m_{48}; m_0, m_1, m_{32}, m_{33}; m_{32}, m_{33}, m_{48}, m_{49}$ ; etc.

Some possible 8-squares are:  $m_1, m_3, m_{17}, m_{19}, m_{33}, m_{35}, m_{49}, m_{51}; m_0, m_2, m_{16}, m_{18}, m_{32}, m_{34}, m_{48}, m_{50}; m_{39}, m_{38}, m_{47}, m_{46}, m_{55}, m_{54}, m_{63}, m_{62}$ ; etc.

The squares are read by dropping out the variables which change. Some possible groupings shown in Figure 6.35 are as follows.

$$m_5, m_{21} = \overline{A}\overline{C}D\overline{E}F(A = C = E = 0, D = F = 1, B = 0 \text{ or } 1)$$

$$m_4, m_{12}, m_{36}, m_{44} = \overline{B}\overline{D}\overline{E}\overline{F}(B = E = F = 0, D = 1, A \text{ and } C \text{ are a } 0 \text{ or a } 1)$$

$$m_{45}, m_{47}, m_{41}, m_{43}, m_{61}, m_{63}, m_{57}, m_{59} = ACF(A = C = F = 1, B, D, \text{ and } E \text{ are a } 0 \text{ or a } 1)$$

$$m_0, m_1, m_2, m_3, m_{16}, m_{17}, m_{18}, m_{19}, m_{32}, m_{33}, m_{34}, m_{35}, m_{48}, m_{49}, m_{50}, m_{51} = \overline{C}\overline{D}(C = D = 0, A, B, E, \text{ and } F \text{ are a } 0 \text{ or a } 1).$$

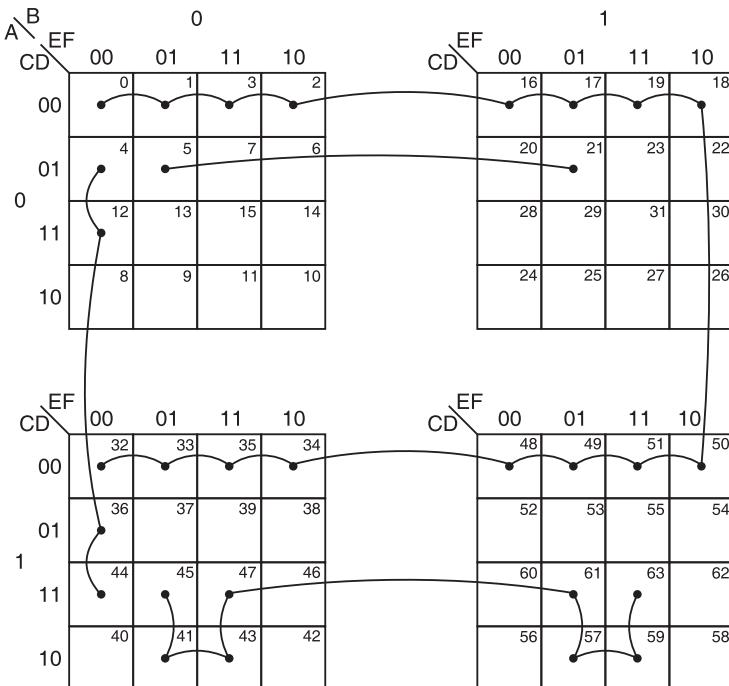


Figure 6.35 Some possible groupings in a six-variable K-map.

### EXAMPLE 6.19 Reduce the expression

$$f = \Sigma m(0, 2, 7, 8, 10, 13, 16, 18, 24, 26, 29, 31, 32, 34, 37, 39, 40, 42, 45, 47, 48, 50, 53, 55, 56, 58, 61, 63)$$

using mapping in SOP and POS forms.

#### Solution

The given expression in the POS form is

$$f = \Pi M(1, 3, 4, 5, 6, 9, 11, 12, 14, 15, 17, 19, 20, 21, 22, 23, 25, 27, 28, 30, 33, 35, 36, 38, 41, 43, 44, 46, 49, 51, 52, 54, 57, 59, 60, 62)$$

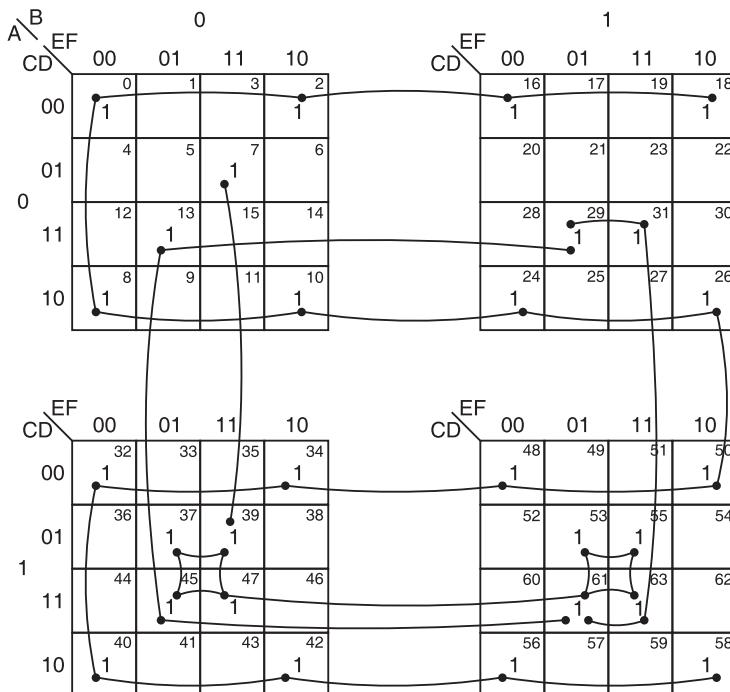
The real minimal expression is the minimal of the SOP and POS forms.

In the SOP K-map shown in Figure 6.36 the reduction is done as per the following steps:

1. There are no isolated 1s.
2.  $m_7$  has only one adjacency  $m_{39}$ . It can form a 2-square with  $m_{39}$ . Read it as  $\bar{B}\bar{C}DEF$ .
3.  $m_{13}$  can make a 4-square with  $m_{29}, m_{45}, m_{61}$ . Read it as  $CD\bar{E}F$ .
4.  $m_{31}$  can make a 4-square with  $m_{29}, m_{63}, m_{61}$ . Read it as  $BCDF$ .
5.  $m_{55}$  can make an 8-square with  $m_{53}, m_{61}, m_{63}, m_{37}, m_{39}, m_{45}, m_{47}$ . Read it as  $ADF$ .
6.  $m_0$  can make a 16-square with  $m_2, m_{16}, m_{18}, m_8, m_{10}, m_{24}, m_{26}, m_{32}, m_{34}, m_{40}, m_{42}, m_{48}, m_{50}, m_{56}, m_{58}$ . Read it as  $\bar{D}\bar{F}$ .
7. Write all the product terms in SOP form.

So the minimal SOP expression is

$$f_{\min} = \bar{B}\bar{C}DEF + CD\bar{E}F + BCDF + ADF + \bar{D}\bar{F} \text{ (23 inputs)}$$



**Figure 6.36** Example 6.19: K-map in SOP form.

In the POS K-map shown in Figure 6.37, the reduction is done as per the following steps:

1. There are no isolated 0s.
2.  $M_{15}$  has only two adjacencies  $M_{14}$  and  $M_{11}$ . It can make a 2-square with any one of them. Make a 2-square of  $M_{15}, M_{14}$ . Read it as  $(A + B + \bar{C} + \bar{D} + \bar{E})$ .
3.  $M_5$  can make a 4-square with  $M_4, M_{20}, M_{21}$  or with  $M_1, M_{17}, M_{21}$ . Do not take a decision yet.
4.  $M_4$  can be expanded into a 16-square with  $M_6, M_{12}, M_{14}, M_{20}, M_{22}, M_{28}, M_{30}, M_{36}, M_{38}, M_{44}, M_{46}, M_{52}, M_{54}, M_{60}$ , and  $M_{62}$ . Read it as  $(\bar{D} + F)$ .

5.  $M_1$  can be expanded into a 16-square with  $M_3, M_9, M_{11}, M_{17}, M_{19}, M_{25}, M_{27}, M_{33}, M_{35}, M_{41}, M_{43}, M_{49}, M_{51}, M_{57}$ , and  $M_{59}$ . Read it as  $(D + \bar{F})$ .
6. Only  $M_5, M_{21}$ , and  $M_{23}$  are left uncovered.  $M_{21}$ , and  $M_{23}$  can form a 4-square with  $M_{20}, M_{22}$  or with  $M_{17}, M_{19}$  which are already taken care of. Form a 4-square of  $M_{21}, M_{23}, M_{17}$  and  $M_{19}$ . Read it as  $(A + \bar{B} + C + \bar{F})$ .
7. Only  $M_5$  is left. Make a 4-square; say with  $M_4, M_{20}$ , and  $M_{21}$ . Read it as  $(A + C + \bar{D} + E)$ .
8. Write all the sum terms in POS form.

So the minimal POS expression is

$$f_{\min} = (A + B + \bar{C} + \bar{D} + \bar{E})(\bar{D} + F)(D + \bar{F})(A + \bar{B} + C + \bar{F})(A + C + \bar{D} + E) \quad (22 \text{ inputs})$$

The POS form is thus less expensive. So the real minimal expression is the POS form.

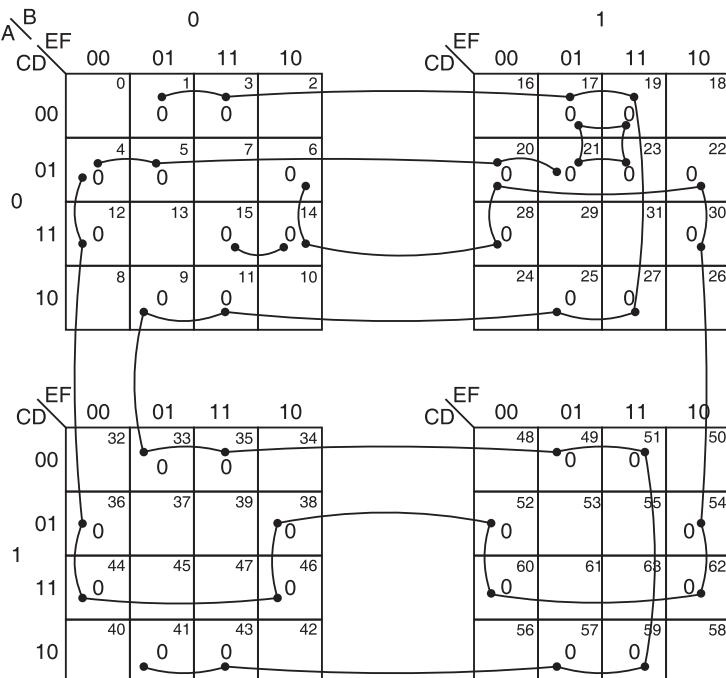


Figure 6.37 Example 6.19: K-map in POS form.

## 6.7 DON'T CARE COMBINATIONS

So far, the expressions considered have been completely specified for every combination of the input variables, that is, each minterm (maxterm) has been specified as a 1 or a 0. It often occurs that for certain input combinations, the value of the output is unspecified either because the input combinations are invalid or because the precise value of the output is of no consequence. The combinations for which the values of the expression are not specified are called *don't care* combinations or *optional* combinations and such expressions, therefore, stand incompletely specified. The output is a don't care for these invalid combinations. For example, in Excess-3 code

system, the binary states 0000, 0001, 0010, 1101, 1110, and 1111 are unspecified and never occur. These are called don't cares. Similarly in 8421 code, the binary states 1010, 1011, 1100, 1101, 1110, and 1111 are invalid and the corresponding outputs are don't cares. The don't care terms are denoted by d, X or  $\phi$ . During the process of design using an SOP map, each don't care is treated as a 1 if it is helpful in map reduction; otherwise it is treated as a 0 and left alone. During the process of design using a POS map, each don't care is treated as a 0 if it is useful in map reduction, otherwise it is treated as a 1 and left alone.

A standard SOP expression with don't cares can be converted into a standard POS form by keeping the don't cares as they are, and writing the missing minterms of the SOP form as the maxterms of the POS form. Similarly, to convert a POS expression with don't cares into an SOP expression, keep the don't cares of the POS expression as they are and write the missing maxterms of the POS expression as the minterms of the SOP expression.

**EXAMPLE 6.20** Reduce the expression  $f = \sum m(1, 5, 6, 12, 13, 14) + d(2, 4)$  and implement the real minimal expression in universal logic.

### Solution

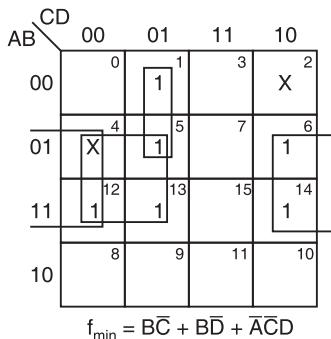
The given expression written in the POS form is  $f = \prod M(0, 3, 7, 8, 9, 10, 11, 15) \cdot \prod d(2, 4)$ . The K-maps in the SOP and POS forms, their reduction and the minimal expressions obtained from them are shown in Figures 6.38a and b respectively. The POS form is less expensive, because it requires less number of gate inputs (9 compared to 10 required for the SOP form). The implementation of the minimal expression using universal logic is shown in Figure 6.38c.

SOP minimal is

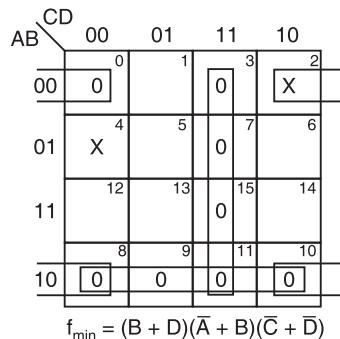
$$f_{\min} = B\bar{C} + B\bar{D} + \bar{A}\bar{C}\bar{D}$$

POS minimal is

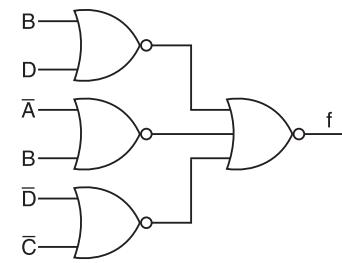
$$f_{\min} = (B + D)(\bar{A} + B)(\bar{C} + \bar{D}) = \overline{(B + D)} + \overline{(\bar{A} + B)} + \overline{(\bar{C} + \bar{D})}$$



(a) SOP K-map



(b) POS K-map



(c) NOR logic

Figure 6.38 Example 6.20.

**EXAMPLE 6.21** Minimize the following expressions using K-map:

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

(b)  $F(A, B, C, D) = \sum m(4, 5, 7, 12, 14, 15) + \sum d(3, 8, 10)$

(c)  $F(W, X, Y, Z) = \sum m(1, 3, 7, 11, 15) + \sum d(0, 2, 5)$

Also show the essential prime implicants and selective prime implicants on the K-map.

**Solution**

The K-maps in SOP form for the given three Boolean expressions (6.21a, b, and c), their minimization and the minimal expressions obtained from them are shown in Figures 6.39a, b, and c respectively. The EPIs and SPIs are indicated only for illustration. The minimal expressions are:

$$(a) F_{\min} = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{C}D + BD + ACD$$

$$(b) F_{\min} = A\bar{D} + BCD + \bar{A}\bar{B}\bar{C}$$

$$(c) F_{\min} = \bar{W}\bar{X} + YZ$$

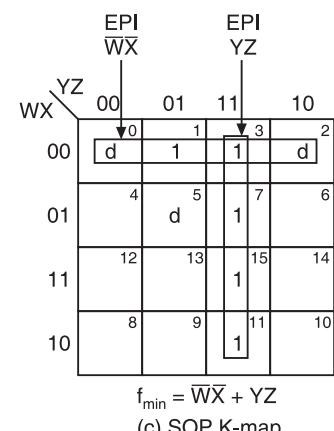
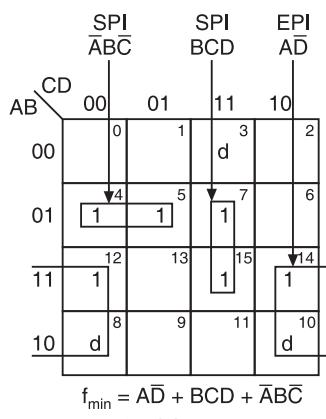
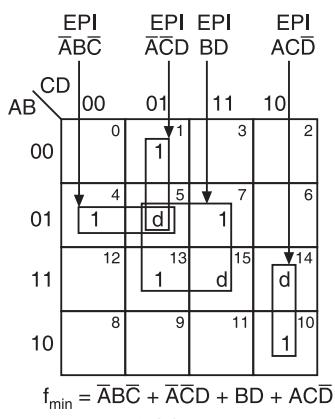


Figure 6.39 Example 6.21.

**EXAMPLE 6.22** Obtain the simplified expression in POS form

$$(a) F(A, B, C, D) = \Sigma m(5, 6, 7, 8, 9, 12, 13, 14)$$

$$(b) F(W, X, Y, Z) = \Sigma m(0, 1, 5, 7, 8, 10, 14, 15)$$

$$(c) F(A, B, C, D) = \Sigma m(0, 1, 2, 3, 4, 5) + d(10, 11, 12, 13, 14, 15)$$

Also indicate the essential false prime implicants and selective false prime implicants.

**Solution**

The given expressions are in SOP form. They can be written in POS form by treating the missing minterms of the SOP form as the maxterms of the POS form. Therefore, the given expressions in POS form are as follows:

$$(a) F(A, B, C, D) = \prod M(0, 1, 2, 3, 4, 10, 11, 15)$$

$$(b) F(W, X, Y, Z) = \prod M(2, 3, 4, 6, 9, 11, 12, 13)$$

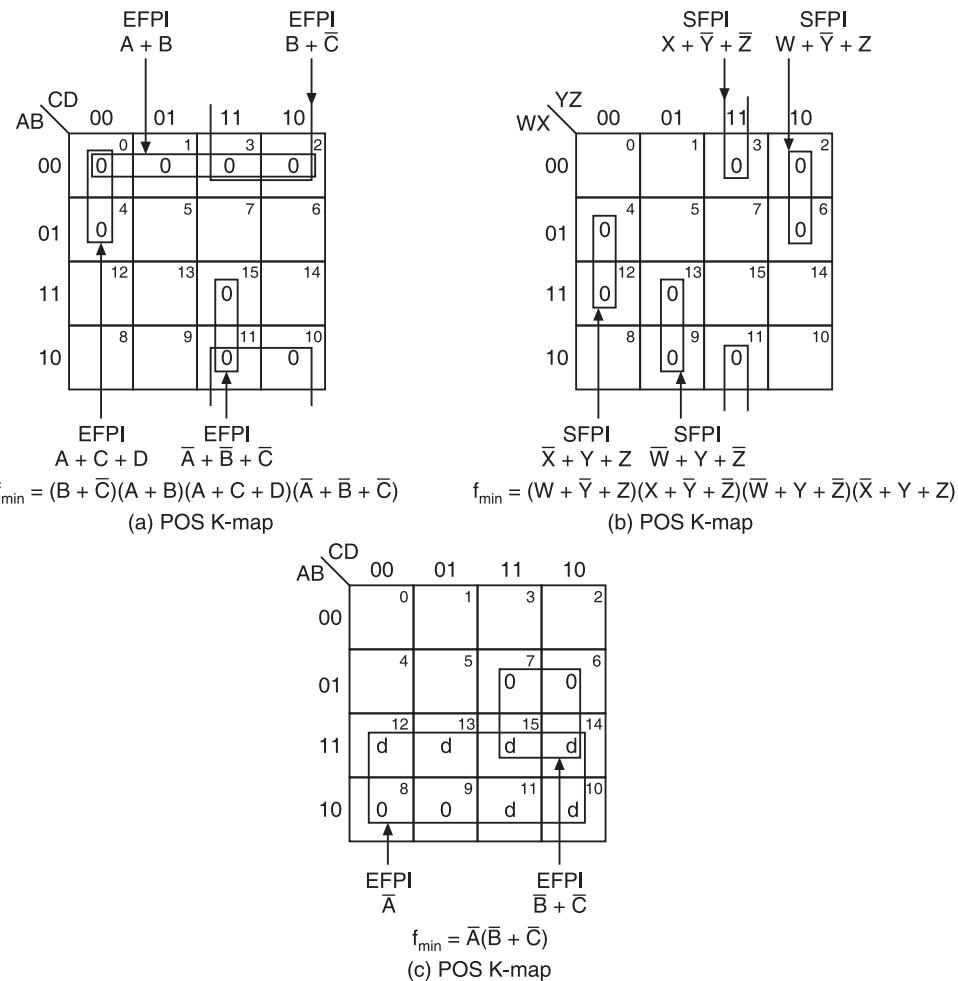
$$(c) F(A, B, C, D) = \prod M(6, 7, 8, 9). d(10, 11, 12, 13, 14, 15)$$

The K-maps in POS form, their minimization and the minimal expressions obtained from them are shown in Figure 6.40. The EFPIs, and SFPIs are shown for illustration. The minimal expressions are:

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

$$(b) F_{\min} = (W + \bar{Y} + Z)(X + \bar{Y} + \bar{Z})(\bar{W} + Y + \bar{Z})(\bar{X} + Y + Z)$$

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

**Figure 6.40** Example 6.22.

**EXAMPLE 6.23** (a) Design a logic circuit using minimum number of basic gates for the following Boolean expression  $f = (\bar{A}\bar{B}\bar{C}\bar{D}) + \bar{A}\bar{B}\bar{C}D + (\bar{A}\bar{B}C\bar{D}) + (\bar{A}\bar{B}CD) + (\bar{A}B\bar{C}\bar{D}) + (\bar{A}B\bar{C}D) + (\bar{A}BC\bar{D}) + (ABC\bar{D}) + (AB\bar{C}\bar{D}) + (AB\bar{C}D) + (AB\bar{C}D) + (ABC\bar{D})$ .

(b) Reduce the following expression using K-map:  $F = (\bar{B}\bar{A} + \bar{A}\bar{B} + A\bar{B})$ .

- (c) Find the output of a four variable K-map, when all the cells are filled with logic LOW.  
 (d) Indicate the essential false prime implicants in the K-map for part (a).

### Solution

(a) The given expression is in algebraic form and it is in SOP form. In terms of minterms, it is

$$f = \sum m(0, 1, 2, 3, 4, 5, 6, 9, 12, 13, 14)$$

The K-map for this SOP function, its minimization, the minimal expression in SOP form

obtained from it and the corresponding logic diagrams in AOI logic and universal logic are shown in Figures 6.41a, b, and c respectively. The EPIs are shown only for illustration. The minimal expression is

$$f_{\min} = \overline{AB} + \overline{CD} + B\overline{D} = \overline{\overline{AB} \cdot \overline{CD} \cdot B\overline{D}}$$

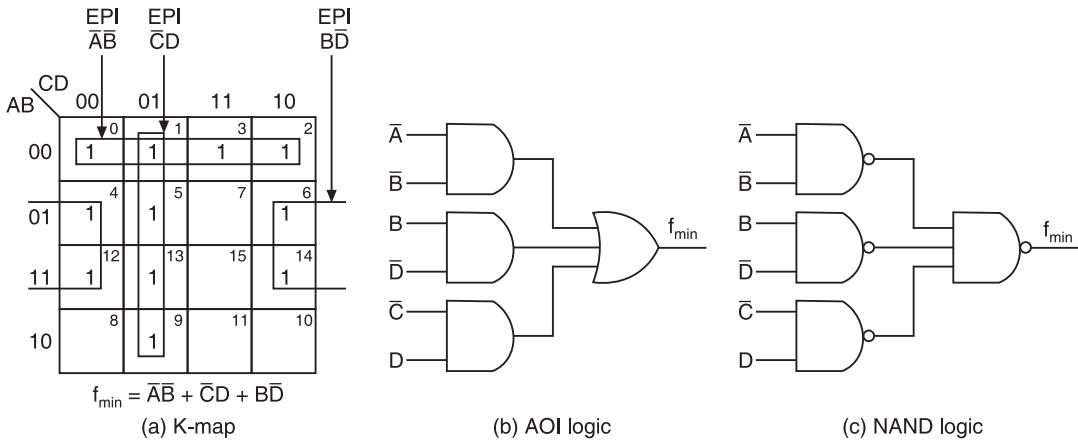


Figure 6.41 Example 6.23a.

(b) The given expression is in the SOP form. It is a two-variable function. In terms of minterms, it is

$$f = \Sigma m(0, 1, 2)$$

The K-map for this function, its minimization, the minimal expression in SOP form obtained from it, and the corresponding logic diagram are shown in Figure 6.42. The minimal expression is

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

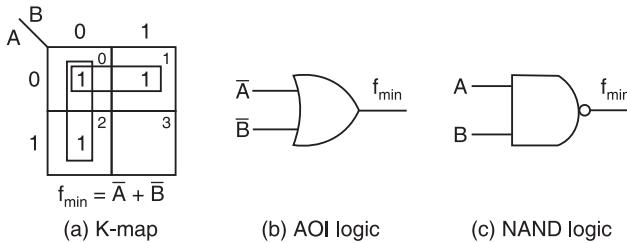


Figure 6.42 Example 6.23b.

(c) When all the cells of a four variable K-map are filled with logic LOW, the output is logic LOW.

**EXAMPLE 6.24** (a) Simplify the Boolean expression using K-map  $F = \overline{A} + AB + ABD + A\overline{B}\overline{D} + C$ .

(b) Obtain the simplified expression using K-map  $F = \overline{ABD} + \overline{ABC}\overline{D} + \overline{ABD} + AB\overline{CD}$ .

(c) Obtain the simplified expression using K-map  $F = ABD + \overline{AC}\overline{D} + \overline{AB} + \overline{AC}\overline{D} + A\overline{BD}$ .

**Solution**

The given Boolean expressions are in SOP form. Expand them into standard SOP form. Write the expressions in terms of minterms and simplify the expressions using K-maps. The given expressions are expressed in minterms as

(a) Given

$$F = \bar{A} + AB + A\bar{B}\bar{D} + C = 0XXX + 11XX + 11X0 + 10X0 + XX1X$$

So the minterms are: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1100, 1101, 1110, 1111, 1100, 1110, 1000, 1010, 0010, 0011, 0110, 0111, 1010, 1011, 1110, 1111.

So

$$\begin{aligned} F &= \sum m(0, 1, 2, 3, 4, 5, 6, 7, 12, 13, 14, 15, 12, 14, 8, 10, 2, 3, 6, 7, 10, 11, 14, 15) \\ &= \sum m(0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15) \end{aligned}$$

(b) Given

$$F = \bar{A}\bar{B}\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}D + A\bar{B}\bar{C}D = 00X0 + 0000 + 01X1 + 1101$$

So the minterms are: 0000, 0010, 0000, 0101, 0111, 1101.

So

$$F = \sum m(0, 2, 5, 7, 11)$$

(c) Given

$$F = ABD + \bar{A}\bar{C}\bar{D} + \bar{A}\bar{B} + \bar{A}\bar{C}\bar{D} + \bar{A}\bar{B}D = 11X1 + 0X00 + 01XX + 0X10 + 10X1$$

So the minterms are: 1101, 1111, 0000, 0100, 0100, 0101, 0110, 0111, 0010, 0110, 1001, 1011.

So

$$F = \sum m(13, 15, 0, 4, 4, 5, 6, 7, 2, 6, 9, 11) = \sum m(0, 2, 4, 5, 6, 7, 9, 11, 13, 15)$$

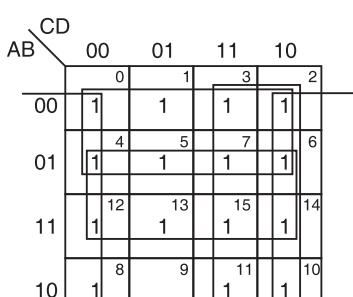
The K-maps for the above three Boolean expressions in SOP form, their minimization and the minimal expressions obtained from them are shown in Figures 6.43a, b, and c respectively.

The minimal expressions are:

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

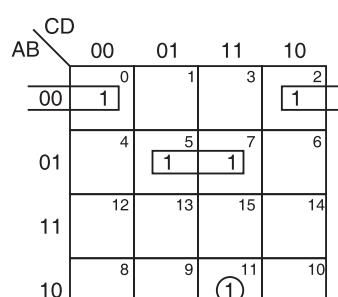
(b)  $F_{\min} = \bar{A}\bar{B}\bar{D} + \bar{A}\bar{B}D + A\bar{B}\bar{C}D$

(c)  $F_{\min} = \bar{A}\bar{D} + BD + AD$



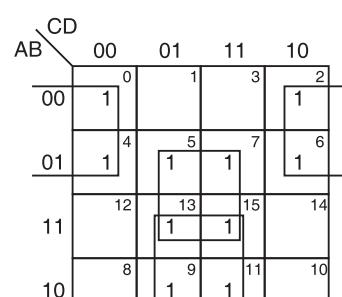
$$f_{\min} = \bar{A} + B + C + \bar{D}$$

(a) SOP K-map



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

(b) SOP K-map



$$f_{\min} = \bar{A}\bar{D} + BD + AD$$

(c) SOP K-map

Figure 6.43 Example 6.24.

**EXAMPLE 6.25** Reduce the following expression to the simplest possible POS and SOP forms.

$$f = \Sigma m(6, 9, 13, 18, 19, 25, 27, 29, 31) + d(2, 3, 11, 15, 17, 24, 28)$$

**Solution**

The given expression is in the SOP form. In the POS form it is

$$f = \prod M(0, 1, 4, 5, 7, 8, 10, 12, 14, 16, 20, 21, 22, 23, 26, 30) \cdot \prod d(2, 3, 11, 15, 17, 24, 28)$$

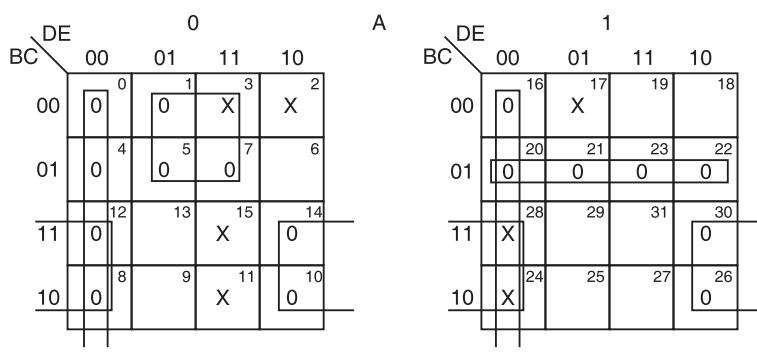
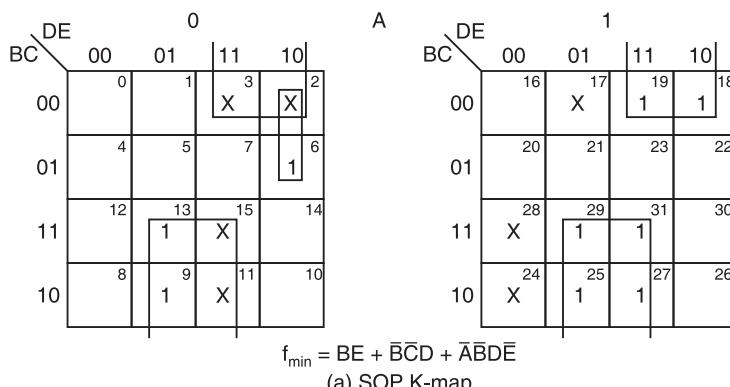
The K-maps in SOP and POS forms, their reduction and the minimal expressions obtained from them are shown in Figures 6.44 a and b respectively. The minimal expressions are:

SOP minimal is

$$f_{\min} = BE + \bar{B}\bar{C}D + \bar{A}\bar{B}D\bar{E}$$

POS minimal is

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



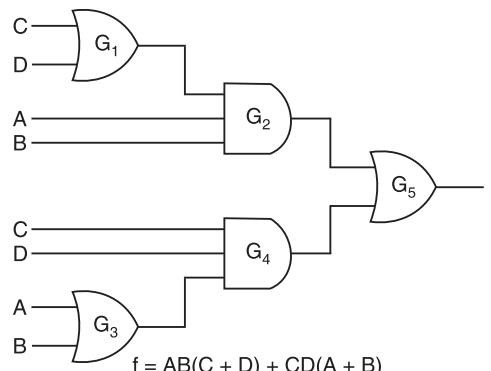
**Figure 6.44** Example 6.25.

## 6.8 HYBRID LOGIC

Both SOP and POS reductions result in a logic circuit in which each input signal has to pass through two gates to reach the output. It is, therefore, called *two-level logic* and has the advantage

of providing uniform time delay between input signals and the output. But the disadvantage is that the minimal expression obtained by either SOP reduction or POS reduction may not be the actual minimal. In fact, the actual minimal may be obtained by manipulating the minimals of SOP and POS forms into a hybrid form. For example, the minimal of the expression  $\Sigma m(0, 1, 3, 4, 5, 6, 7, 13, 15)$  in the SOP form is given by  $f = \bar{A}\bar{C} + \bar{A}\bar{D} + \bar{A}\bar{B} + BC$  (12 inputs). But this can be written as  $f = \bar{A}(\bar{C} + D + B) + BC$  (9 inputs).

Also, the expression  $ABC + ABD + ACD + BCD$  is in minimum SOP form and requires 16 inputs. It can, however, be reduced by factoring to:  $AB(C + D) + CD(A + B)$  and implemented as shown in Figure 6.45 with 12 inputs.



**Figure 6.45** Hybrid logic.

Figure 6.45 shows that we have reduced the number of inputs from 16 to 12. Note, however, that the C input to the OR gate must go through three levels of logic before reaching the output, whereas the C input to the AND gate must only go through two levels. This can result in a critical timing problem called *logic race*. Assume, for example, that each gate has a 10-ns delay and that  $A = 0, B = 0, C = 1$ , and  $D = 1$ . Gate  $G_2$  will not AND since A and B are zero; gate  $G_4$  will not AND since  $A + B = 0$ . Next assume that A and B go high at precisely the same instant when C and D go low. Gate  $G_1$  will continue to provide a 1 to  $G_2$  for 10 ns after C and D go low because of its propagation delay, and for that 10 ns all three inputs to  $G_2$  will be high causing a 10-ns pulse to be outputted by  $G_5$ . At the end of this narrow pulse,  $G_1$  output will go low, blocking  $G_2$ ; since C and D are already 0,  $G_5$  output will go low. Had two-level logic been used, this logic race and its resulting pulse would not have occurred.

In hybrid logic circuits, the input signals will pass through different numbers of gates to reach the output. Even though hybrid logic results in a minimal circuit, it may provide a critical timing problem called logic race which results in unwanted narrow pulses. The two-level logic is free from logic race.

## 6.9 MAPPING WHEN THE FUNCTION IS NOT EXPRESSED IN MINTERMS (MAXTERMS)

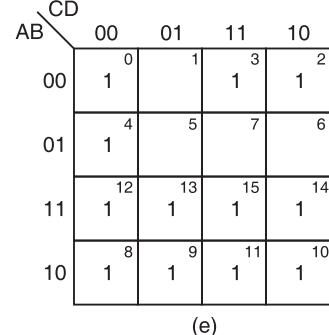
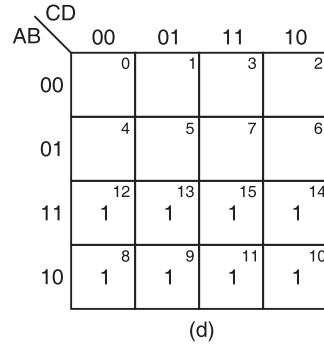
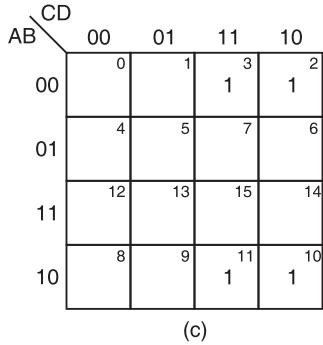
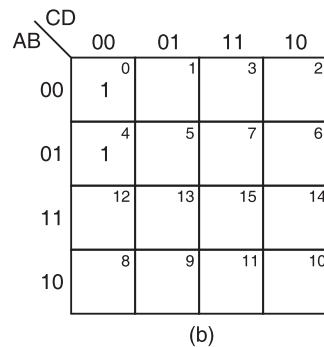
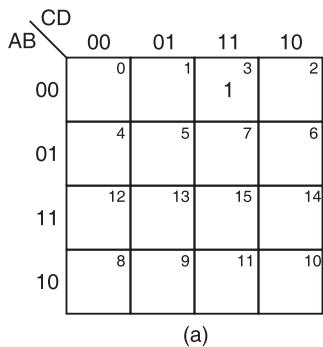
Our discussion of mapping suggests that if an expression is to be entered on a K-map, it must be available as a sum (product) of minterms (maxterms). However, if not so expressed, it is not

necessary to expand the expression algebraically into its minterms (maxterms). Instead, the expansion into minterms (maxterms) can be accomplished in the process of entering the terms of the expression on the K-map.

For example, let us enter on K-map the expression  $\bar{A}\bar{B}CD + \bar{A}\bar{C}\bar{D} + \bar{B}C + A$ .

- $\bar{A}\bar{B}CD$  is minterm  $m_3$ . Enter it as it is.
- $\bar{A}\bar{C}\bar{D}$  corresponds on the K-map to locations where  $A = C = D = 0$  and are independent of  $B$ . That is, intersections of rows 1 and 2 with column 1 ( $m_0$  and  $m_4$ ).
- $\bar{B}C$  corresponds on the K-map to locations where  $B = 0, C = 1$  and are independent of  $A$  and  $D$ . That is, intersections of rows 1 and 4 with columns 3 and 4 ( $m_2, m_3, m_{10}$ , and  $m_{11}$ ).
- $A$  corresponds on the K-map to locations where  $A = 1$  and are independent of  $B, C$ , and  $D$ . That is, complete rows 3 and 4 ( $m_8, m_9, m_{10}, m_{11}, m_{12}, m_{13}, m_{14}$ , and  $m_{15}$ ).

The entries on the K-map are shown in Figures 6.46a, b, c, and d. The complete mapping is shown in Figure 6.46e.



**Figure 6.46** Mapping of (a)  $\bar{A}\bar{B}CD$ , (b)  $\bar{A}\bar{C}\bar{D}$ , (c)  $\bar{B}C$ , (d)  $A$ , and (e)  $\bar{A}\bar{B}CD + \bar{A}\bar{C}\bar{D} + \bar{B}C + A$ .

As another example, consider mapping the expression  $(A + B)(A + \bar{B} + C)(A + \bar{C})$ .

The given expression is in the POS form, where

- $(A + B)$  corresponds on the K-map to locations where  $A = 0$  and  $B = 0$ , and  $C$  and  $D$  can be a 0 or a 1. That is, the entire row 1 ( $M_0, M_1, M_2$ , and  $M_3$ ).
- $(A + \bar{B} + C)$  corresponds on the K-map to locations where  $A = 0, B = 1$ , and  $C = 0$ , and are independent of  $D$ . That is, the intersection of row 2 with columns, 1 and 2 ( $M_4$  and  $M_5$ ).

- (c)  $(A + \bar{C})$  corresponds on the K-map to locations where  $A = 0$  and  $C = 1$ , and are independent of  $B$  and  $D$ . That is, the intersection of rows 1 and 2 with columns 3 and 4 ( $M_2, M_3, M_6$ , and  $M_7$ ).

Hence the given expression can be mapped as  $\Pi M(0, 1, 2, 3, 4, 5, 2, 3, 6, 7)$ , i.e.  $\Pi M(0, 1, 2, 3, 4, 5, 6, 7)$ . The mapping is shown in Figure 6.47.

		CD		AB			
		00	01	11	10		
AB	00	0	1	3	2		
	01	4	5	7	6		
		0	0	0	0		
		12	13	15	14		
		8	9	11	10		

**Figure 6.47** Mapping of  $(A + B)(A + \bar{B} + C)(A + \bar{C})$ .

**EXAMPLE 6.26** Make a K-map of the following expression and obtain the minimal SOP and POS forms.

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

#### Solution

Obtain the expression in the standard SOP form by expansion. The expression can also be expanded directly on the K-map. Each of the terms can be directly mapped as shown in Figure 6.48.

$AB = ABXX(1 1 X X)$ , i.e.  $A = 1, B = 1$ ;  $C$  and  $D$  can be a 0 or a 1, i.e. the entire third row ( $m_{12}, m_{13}, m_{14}$ , and  $m_{15}$ ).

$A\bar{C} = AX\bar{C}X(1 X 0 X)$ , i.e.  $A = 1, C = 0$ ;  $B$  and  $D$  can be a 0 or a 1, i.e. the intersections of the third and fourth rows with the first and second columns ( $m_8, m_9$  and  $m_{12}, m_{13}$ ).

$C = XXCX(X X 1 X)$ , i.e.  $C = 1, A, B$ , and  $D$  can be a 0 or a 1, i.e. the entire third and fourth columns ( $m_2, m_3, m_6, m_7$  and  $m_{10}, m_{11}, m_{14}, m_{15}$ ).

$AD = AXXD(1 X X 1)$ , i.e.  $A = 1, D = 1$ ;  $B$  and  $C$  can be a 0 or a 1, i.e. the intersection of the third and fourth rows with the second and third columns ( $m_9, m_{11}$  and  $m_{13}, m_{15}$ ).

$A\bar{B}C = A\bar{B}CX(1 0 1 X)$ , i.e.  $A = 1, B = 0, C = 1$ ;  $D$  can be a 0 or a 1, i.e. the intersection of the fourth row with the third and fourth columns ( $m_{10}$  and  $m_{11}$ ).

$ABC = ABCX(1 1 1 X)$ , i.e.  $A = 1, B = 1, C = 1$ ;  $D$  can be a 0 or a 1, i.e. the intersection of the third row with the third and fourth columns ( $m_{14}$  and  $m_{15}$ ).

By expansion on the map we see that  $f = (2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)$ . So the POS expression is  $f = (0, 1, 4, 5)$ .

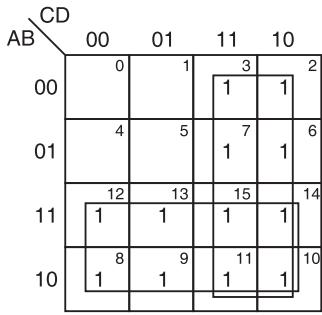
The K-maps in SOP and POS forms, the minimal expressions obtained from them, and the logic diagram corresponding to the minimal of those minimals are shown in Figures 6.48a, b, and c respectively. Both the SOP and POS forms give the same minimal expression.

SOP minimal is

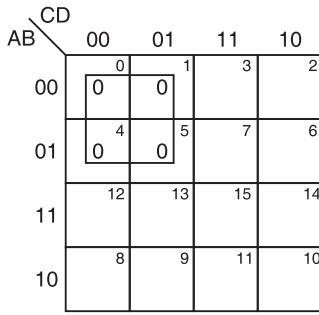
$$f_{\min} = A + C$$

POS minimal is

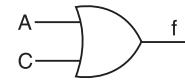
$$f_{\min} = A + C$$



(a) SOP K-map



(b) POS K-map



(c) Logic diagram

Figure 6.48 Example 6.26.

## 6.10 MINIMIZATION OF MULTIPLE OUTPUT CIRCUITS

So far, we have discussed the minimization of single expressions by the K-map method. In practice, many logic design problems involve designing of circuits with more than one output. We discuss here the design of such problems using K-maps. The multiple output minimization criterion is as follows:

1. Each minimized expression should have as many terms in common as possible with those in the other minimized expressions. For this, in addition to separate K-maps for each output expression, draw an additional K-map called the *shared minterm K-map* for the minterms which are common to all the output expressions and then obtain the common terms from it.
2. Each minimized expression should have a minimum number of product (sum) terms and no product (sum) term of which can be replaced by another product (sum) term with a fewer variables. For this, out of the common terms obtained from the shared minterm K-map, select only those terms whose inclusion will result in the reduction of the overall cost.

For example, consider the multiple output circuit shown in Figure 6.49 with three inputs A, B, and C, and two outputs  $f_1$  and  $f_2$  given by

$$f_1(A, B, C) = \Sigma m(0, 1, 2, 5, 6, 7)$$

$$f_2(A, B, C) = \Sigma m(2, 4, 5, 6)$$

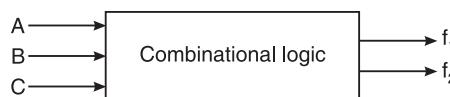
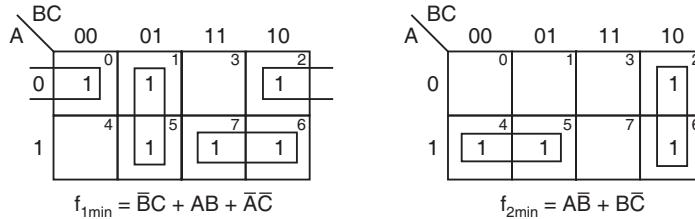


Figure 6.49 Block diagram of a multiple output circuit.

We want to design a minimal circuit to get the above outputs. If we design the circuit by obtaining the minimal expressions for  $f_1$  and  $f_2$  separately, we may not get the overall minimal circuit. Suppose the groupings on the individual K-maps are as shown in Figure 6.50; the minimal expressions for the outputs would require 15 gate inputs for realization. Here,

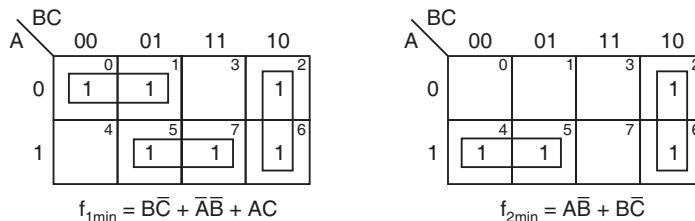
$$f_{1\min} = \overline{A}\overline{C} + \overline{B}C + AB \quad (9 \text{ inputs}); \quad f_{2\min} = A\overline{B} + B\overline{C} \quad (6 \text{ inputs})$$

(Total 15 inputs)



**Figure 6.50** One way of grouping  $f_1$  and  $f_2$ .

An alternative way of grouping the outputs  $f_1$  and  $f_2$  is shown in Figure 6.51.

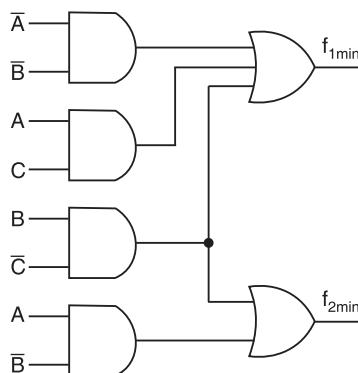


**Figure 6.51** Alternative way of grouping  $f_1$  and  $f_2$ .

Now,  $f_{1\min} = B\overline{C} + \overline{A}\overline{B} + AC \quad (9 \text{ inputs}); \quad f_{2\min} = A\overline{B} + B\overline{C} \quad (6 \text{ inputs})$

(Total 13 inputs because the term  $B\overline{C}$  is common to  $f_{1\min}$  and  $f_{2\min}$ )

As the term  $B\overline{C}$  is present in both the expressions, it can be generated only once and utilized for generating both the expressions, thus reducing the cost and complexity. The realization is shown in Figure 6.52. So the first step in multiple output minimization is to find the common terms and try to utilize them so that the overall cost is reduced.



**Figure 6.52** Minimal circuit.

**EXAMPLE 6.27** Minimize and implement the following multiple output functions.

$$f_1 = \Sigma m(1, 2, 3, 6, 8, 12, 14, 15)$$

$$f_2 = \Pi M(0, 4, 9, 10, 11, 14, 15)$$

**Solution**

Here  $f_2$  is in the POS form and  $f_1$  is in the SOP form. Express both  $f_1$  and  $f_2$  either in the SOP form or in the POS form and obtain the minimal expressions. Therefore, in the SOP form, we have

$$f_1 = \Sigma m(1, 2, 3, 6, 8, 12, 14, 15); \quad f_2 = \Sigma m(1, 2, 3, 5, 6, 7, 8, 12, 13)$$

First form a function  $f$  with the minterms common to both the functions, i.e.  $f = f_1 \cdot f_2$ . Therefore,

$$f = f_1 \cdot f_2 = \Sigma m(1, 2, 3, 6, 8, 12)$$

Draw the K-maps for  $f_1$ ,  $f_2$  and  $f$  and form the minimal expressions for  $f_1$ ,  $f_2$ , and  $f$ . The K-maps for  $f_1$ ,  $f_2$ , and  $f$  and their reductions are shown in Figure 6.53.

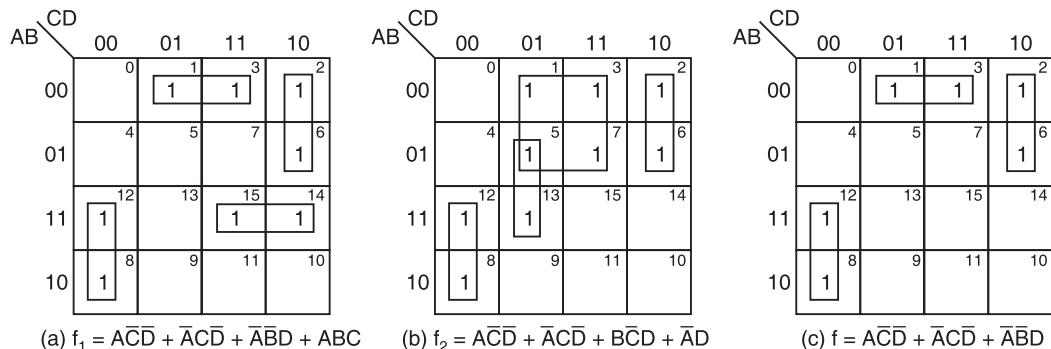


Figure 6.53 Example 6.27: K-maps for  $f_1$ ,  $f_2$ , and  $f$ .

In  $f_1$ , all the terms of  $f$  are present. We cannot make any larger square using any of these terms. In  $f_2$ , out of the three terms of  $f$ ,  $\overline{A}BD$  becomes part of a 4-square, so,  $\overline{AD}$  is read.  $\overline{ACD}$  can also be made part of a 4-square, but it does not reduce the hardware. So it is not considered. The circuit with the minimum gate inputs is shown in Figure 6.54.

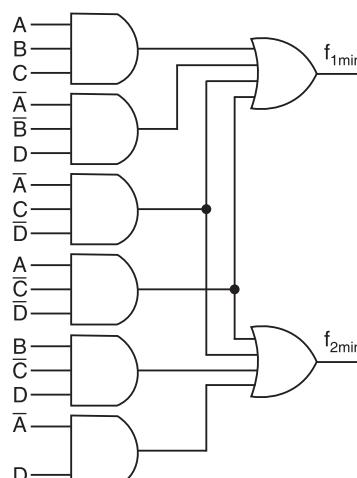


Figure 6.54 Example 6.27: Logic diagram.

### 6.10.1 Don't Care Conditions

When there are incompletely specified functions, the minterms for the shared map are generated by the rules listed in Table 6.2.

**Table 6.2** Generating minterms for a shared map

$f_1$	$f_2$	$f = f_1 \cdot f_2$
0	0	0
0	1	0
1	0	0
0	X	0
X	0	0
1	X	1
X	1	1
1	1	1
X	X	X

**EXAMPLE 6.28** Minimize the following multiple output functions.

$$f_1 = \Sigma m(0, 2, 6, 10, 11, 12, 13) + d(3, 4, 5, 14, 15)$$

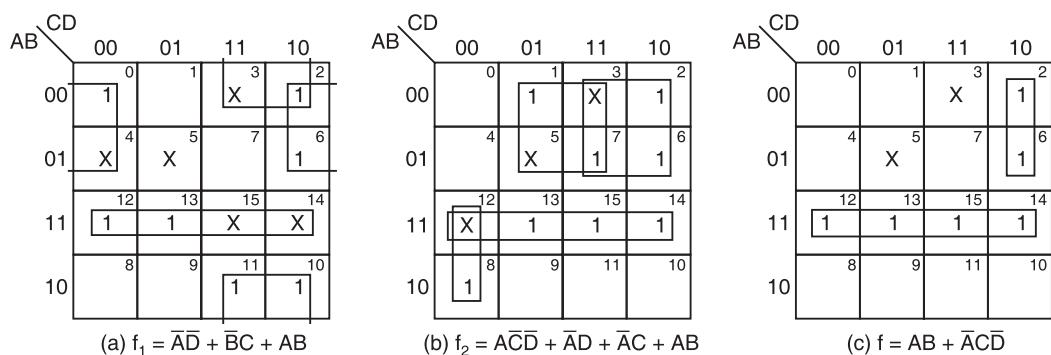
$$f_2 = \Sigma m(1, 2, 6, 7, 8, 13, 14, 15) + d(3, 5, 12)$$

**Solution**

The shared minterm function generated according to the rules listed in Table 6.2 is

$$f = \Sigma m(2, 6, 12, 13, 14, 15) + d(3, 5)$$

The K-maps for  $f_1$ ,  $f_2$ , and  $f$  and their minimization are shown in Figure 6.55.



**Figure 6.55** Example 6.28: K-maps for  $f_1$ ,  $f_2$ , and  $f$ .

The shared minterm function  $f$  has two terms  $AB$  and  $\bar{A}\bar{C}\bar{D}$ . Out of these two, only  $AB$  is utilized, because  $\bar{A}\bar{C}\bar{D}$  can be merged into a bigger square.

**EXAMPLE 6.29** Find the minimal expressions for the multiple output functions

$$f_1(X_1, X_2, X_3, X_4) = \prod M(3, 4, 5, 7, 11, 13, 15) \cdot d(6, 8, 10, 12)$$

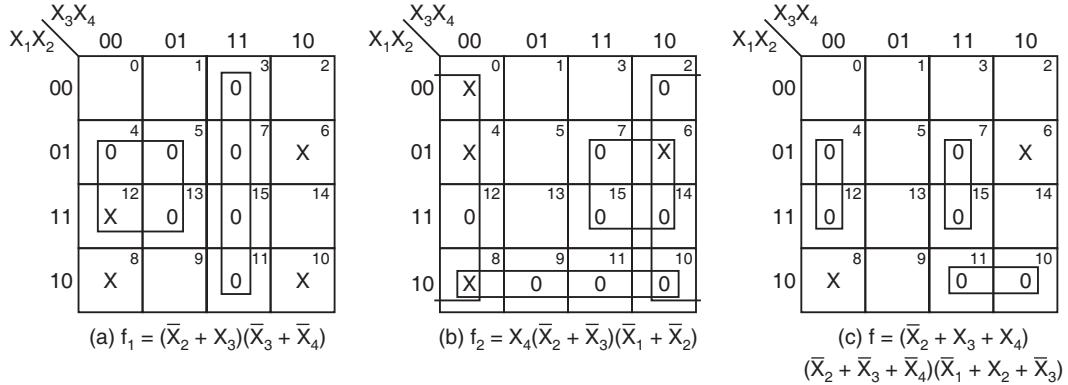
$$f_2(X_1, X_2, X_3, X_4) = \prod M(2, 7, 9, 10, 11, 12, 14, 15) \cdot d(0, 4, 6, 8)$$

**Solution**

The given expressions are in the POS form. Generate the shared maxterm function using the same rules as for the minterms. Therefore,

$$F = f_1 \cdot f_2 = \prod M(4, 7, 10, 11, 12, 15) \cdot d(6, 8)$$

The K-maps for  $f_1$ ,  $f_2$ , and  $f$ , their reduction, and the minimal expressions obtained from them are shown in Figure 6.56.



**Figure 6.56** Example 6.29: K-maps for  $f_1$ ,  $f_2$ , and  $f$ .

None of the terms of  $f$  can be used in the minimal expressions for  $f_1$  and  $f_2$  because these terms are combined into bigger ones in  $f_1$  and  $f_2$ .

The problem may be converted to and solved in SOP form too.

**EXAMPLE 6.30** Minimize the following multiple output functions using K-map.

$$f_1(X_1, X_2, X_3, X_4) = \Sigma m(1, 2, 3, 5, 7, 8, 9) + d(12, 14)$$

$$f_2(X_1, X_2, X_3, X_4) = \Sigma m(0, 1, 2, 3, 4, 6, 8, 9) + d(10, 11)$$

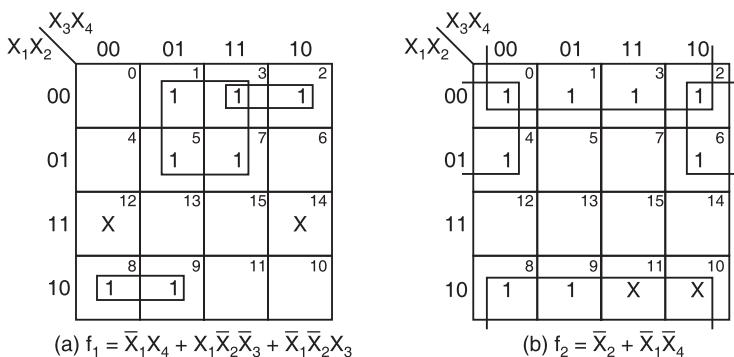
$$f_3(X_1, X_2, X_3, X_4) = \Sigma m(1, 3, 5, 7, 8, 9, 12, 13) + d(14, 15)$$

**Solution**

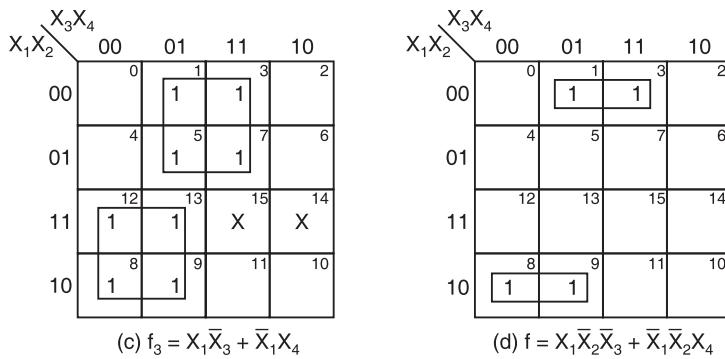
The shared minterm function generated according to the rules listed in Table 6.2 is

$$f(X_1, X_2, X_3, X_4) = f_1 \cdot f_2 \cdot f_3 = \Sigma m(1, 3, 8, 9)$$

The K-maps for  $f_1$ ,  $f_2$ ,  $f_3$ , and  $f$ , their minimization, and the minimal expressions obtained from them are shown in Figure 6.57.



**Figure 6.57** Example 6.30: K-maps for  $f_1$ ,  $f_2$ ,  $f_3$ , and  $f$  (Contd.)...



**Figure 6.57** Example 6.30: K-maps for  $f_1$ ,  $f_2$ ,  $f_3$ , and  $f$ .

None of the terms in the minimal expression of the shared map is useful for overall reduction. However, we can see that  $\bar{X}_1X_4$  is common to  $f_1$  and  $f_3$ .

## 6.11 VARIABLE MAPPING

Variable mapping is a powerful and useful tool that can be used in a wide variety of problems. The design of sequential circuits is greatly simplified by this technique. Variable mapping can also be used to minimize Boolean expressions, which involve infrequently used variables. If properly used, it can reduce the work required for plotting and reading maps. It allows us to reduce a large mapping problem to one that uses just a small map. This technique can reduce the map size for 3, 4, 5, 6, 7, and 8 variable maps. It is especially useful in those problems which have a few isolated variables among more frequently used variables. Consider the equation

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

Normally, this would be a four-variable problem. Notice that the variable D is used only once whereas the variables A, B, and C are used many times. Using variable mapping, we can make it a three-variable problem. Assuming this to be a three-variable problem in A, B, and C, we have

$$f = m_0 + m_1 + m_2 + m_3 + m_6D$$

For mapping such a function, put a 1 on the map where a minterm appears and a 0 (or no entry) where it does not. Since each 1 represents a minterm, the entry  $1 \cdot D$  represents the minterm multiplied by D. In fact, each 1 entered onto the map represents  $D + \bar{D}$ , because if it were to be treated as a four-variable problem instead of a three-variable one, each term in the function is to be multiplied by  $(D + \bar{D}) = 1$  to convert it to the standard SOP form. For minimization, we must cover each of the individual variables. We can make a 4-square of minterms  $m_0$ ,  $m_1$ ,  $m_2$ , and  $m_3$  [covering  $(D + \bar{D})$  of all the terms] and can read it as usual as  $\bar{A}(D + \bar{D}) = \bar{A}$ . We can also make a 2-square of the D in  $m_6$  and D in  $(D + \bar{D})$  of  $m_2$  and read it as  $(B\bar{C})D = B\bar{C}D$ . This yields  $f_{\min} = \bar{A} + B\bar{C}D$  as shown in Figure 6.58.

Note that the D in  $m_2$  is covered twice. However, our objective is to cover all the variables at least once.

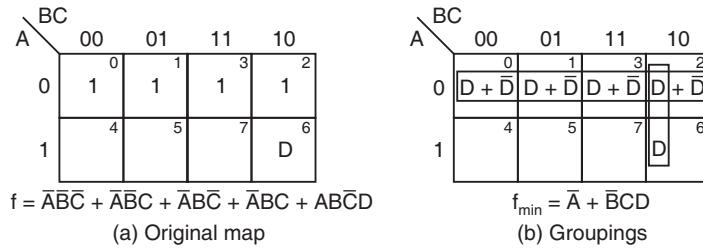


Figure 6.58 Variable mapping.

**EXAMPLE 6.31** Reduce by mapping:

$$\begin{aligned} f = & \overline{A}\overline{B}CD + \overline{A}\overline{B}C\overline{D} + A\overline{B}\overline{C}D + A\overline{B}\overline{C}\overline{D} + ABC\overline{D} + ABCD + A\overline{B}CD \\ & + \overline{A}BCD + A\overline{B}\overline{C}D + A\overline{B}\overline{C}\overline{D} \end{aligned}$$

**Solution**

Although this is a four-variable problem, it can be treated as a three-variable one and plotted on a three-variable K-map and reduced as shown in Figure 6.59. Thus, the three-variable problem in A, B, C would be:

$$F = m_1(D + \overline{D}) + m_6(D + \overline{D}) + m_7(D + \overline{D}) + m_5\overline{D} + m_3D + m_2\overline{D} + m_4D$$

Note that all parts of  $1(D + \overline{D})$  must be covered.

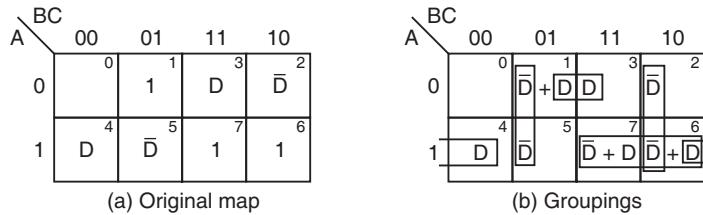


Figure 6.59 Example 6.31: K-maps.

As seen from the K-maps,

The  $\overline{D}$ 's of  $m_1$  and  $m_5$  form a 2-square which is read as  $\overline{BCD}$ .

The  $D$ 's of  $m_1$  and  $m_3$  form a 2-square which is read as  $\overline{ACD}$ .

The  $D$ 's of  $m_4$  and  $m_6$  form a 2-square which is read as  $\overline{ACD}$ .

The  $\overline{D}$ 's of  $m_2$  and  $m_6$  form a 2-square which is read as  $B\overline{CD}$ .

$m_1$  and  $m_6$  are fully covered.

Since  $m_7$  is not fully covered it can be combined with  $m_6$  to form a 2-square which is read as  $AB$ . So, the reduced expression is

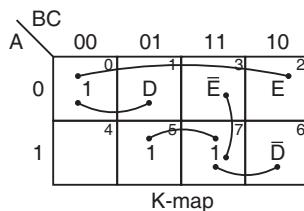
$$f = A\overline{C}D + \overline{B}CD + \overline{ACD} + B\overline{CD} + AB$$

**EXAMPLE 6.32** Reduce by mapping:

$$f = \overline{ABC} + \overline{ABC}D + \overline{ABC}\overline{E} + \overline{ABC}E + A\overline{B}CE + ABC + A\overline{B}CD$$

**Solution**

Although this is a five-variable problem, it can be treated as a three-variable one and plotted on a three-variable K-map and then reduced as shown in Figure 6.60.

**Figure 6.60** Example 6.32: K-maps.

The given function in three-variable form may be expressed as

$$f = m_0 + m_1 D + m_3 \bar{E} + m_2 E + m_5 + m_7 + m_6 \bar{D}$$

As seen from the K-map,

D of m<sub>1</sub> forms a 2-square with D present in m<sub>0</sub> which is read as  $\bar{A}\bar{B}D$ .

E of m<sub>2</sub> forms a 2-square with E present in m<sub>0</sub> which is read as  $\bar{A}\bar{C}E$ .

$\bar{E}$  of m<sub>3</sub> forms a 2-square with  $\bar{E}$  present in m<sub>7</sub> which is read as  $B\bar{C}\bar{E}$ .

$\bar{D}$  of m<sub>6</sub> forms a 2-square with  $\bar{D}$  present in m<sub>7</sub> which is read as  $A\bar{B}\bar{D}$ .

A minterm is said to be fully covered, only when atleast any one of the variables as well as its complement are covered. Only  $\bar{D}$  and  $\bar{E}$  of m<sub>7</sub> are covered. So, m<sub>7</sub> is not fully covered. Also, m<sub>5</sub> is not covered at all. So, make a 2-square with m<sub>5</sub> and m<sub>7</sub> which is read as AC.

All parts of m<sub>0</sub> are not covered. Only D and E of m<sub>0</sub> are covered and m<sub>0</sub> cannot also be combined with any other minterms. So, it appears in the reduced expression.

From the K-map, the reduced expression is

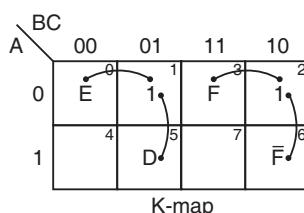
$$f = AC + ABD + BCE + \bar{A}\bar{B}D + \bar{A}\bar{C}E + \bar{A}\bar{B}\bar{C}$$

**EXAMPLE 6.33** Reduce by mapping:

$$f = \bar{A}\bar{B}\bar{C}E + \bar{A}\bar{B}C + \bar{A}BCF + \bar{A}\bar{B}\bar{C} + A\bar{B}CD + AB\bar{C}\bar{F}$$

### Solution

Although this is a six-variable problem, it can be treated as a three-variable one and plotted on a three-variable K-map and then reduced as shown in Figure 6.61.

**Figure 6.61** Example 6.33: K-map.

The given function may be expressed in three variable form as

$$f = m_0 E + m_1 + m_2 + m_3 F + m_5 D + m_6 \bar{F}$$

Here m<sub>1</sub> is not fully covered, because only D and E of m<sub>1</sub> are covered. So it will appear in the final expression. m<sub>2</sub> is fully covered because both F and  $\bar{F}$  of it are covered. So, it will not appear in the final expression.

From the K-map the reduced expression is, therefore, given by

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

**EXAMPLE 6.34** Reduce by mapping:

$$\begin{aligned} f = & \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}DE + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D}\bar{E} + \bar{A}\bar{B}\bar{C}DF + \bar{A}\bar{B}C\bar{D}E + \bar{A}\bar{B}CD \\ & + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}DE + A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D}\bar{E} + AB\bar{C}\bar{D} + ABC\bar{D} + ABCD\bar{F} \end{aligned}$$

### Solution

Although this is a six-variable problem, it can be treated as a four-variable one and plotted on a four-variable K-map and then reduced as shown in Figure 6.62.

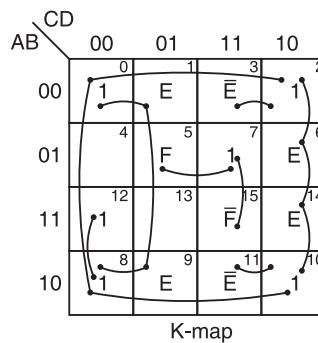


Figure 6.62 Example 6.34: K-map.

Expressed as a four-variable problem, we have

$$\begin{aligned} f = & m_0 + m_1 E + m_2 + m_3 \bar{E} + m_5 F + m_6 E + m_7 + m_8 + m_9 E + m_{10} + m_{11} \bar{E} + m_{12} \\ & + m_{14} E + m_{15} \bar{F} \end{aligned}$$

$m_7$  is fully covered because it combines with  $F$  and  $\bar{F}$ ; likewise,  $m_2$  and  $m_{10}$  are also fully covered because each one of them combines with  $E$  and  $\bar{E}$ . The  $E$ 's of  $m_1$  and  $m_9$  form a 4-square with the  $E$ 's present in  $m_0$  and  $m_8$ . The  $\bar{E}$ 's of  $m_3$  and  $m_{11}$  form a 4-square with the  $\bar{E}$ 's present in  $m_2$  and  $m_{10}$ . The  $E$ 's of  $m_6$  and  $m_{14}$  form a 4-square with the  $E$ 's present in  $m_2$  and  $m_{10}$ . Now  $m_0$  is not fully covered, because  $\bar{E}$  of this is not used. So, form a 4-square with  $m_0$ ,  $m_2$ ,  $m_8$ , and  $m_{10}$ .

From the K-map, the reduced expression is, therefore, obtained as

$$f = \bar{A}BDF + \bar{B}CE + \bar{B}C\bar{E} + CDE + BCD\bar{F} + A\bar{C}\bar{D} + \bar{B}\bar{D}$$

### 6.11.1 Incompletely Specified Functions

The procedure used for previous don't care problems has to be modified only slightly to accommodate variable mapping. Suppose a four-variable problem is treated as a three-variable ( $A, B, C$ ) one and entered on a three-variable K-map. Each 1 in the three-variable map represents that the corresponding minterm is multiplied by  $(D + \bar{D})$ . In fact, it may be entered as a 1 or  $(D + \bar{D})$ . If this minterm is a don't care, it is entered as  $XD + X\bar{D}$  or simply as  $X$ . Suppose the term is  $ABC\bar{D}$ . It is entered as a  $\bar{D}$  in the cell for  $m_7$  ( $ABC\bar{D} = m_7\bar{D}$ ). If  $ABC\bar{D}$  is a don't care, it is entered as an  $X\bar{D}$  in the cell for  $m_7$ . Similarly, when a five-variable problem is treated as a

three-variable one, each 1 on the map represents the corresponding minterm multiplied by  $(D + \bar{D})(E + \bar{E})$ . If it is a don't care, it contains the terms  $X\bar{D}$ ,  $X\bar{D}$ ,  $XE$ , and  $X\bar{E}$ . These don't cares may or may not be covered. They should be used to make  $2^n$  squares when covering other *do care* terms. Any minterm is considered to be completely covered only if any of its variable plus its complement present in that minterm are both covered. Other variables can be covered partially or fully, if required or not covered at all.

**EXAMPLE 6.35** Reduce by mapping:

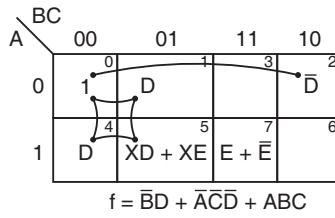
$$f = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}CD + \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + ABCE + ABC\bar{E} + d(A\bar{B}CD + A\bar{B}CE)$$

**Solution**

The given problem is actually a five-variable one. Treating it as a three-variable one, we get

$$f = m_0 + m_1 D + m_2 \bar{D} + m_4 D + m_7 E + m_7 \bar{E} + d(m_5 D + m_5 \bar{E})$$

It is mapped onto a K-map and reduced as shown in Figure 6.63.



**Figure 6.63** Example 6.35: K-map.

The D's of  $m_1$  and  $m_4$  form a 4-square with the D present in  $m_0$  and the  $XD$  of  $m_5$ . The  $\bar{D}$  of  $m_2$  combines with the  $\bar{D}$  present in  $m_0$  to form a 2-square. So,  $m_0$  is fully covered. The E of  $m_7$  can combine with the  $XE$  of  $m_5$ . This is not done, because there is no way to combine the  $\bar{E}$  of  $m_7$  with anything else, and so,  $m_7$  has to appear in the final expression. From the K-map, the reduced expression is, therefore, obtained as

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

**EXAMPLE 6.36** Reduce by mapping:

$$\begin{aligned} f = m_0 + m_1 F + m_2 + m_4 E + m_6(E + \bar{E}) + m_7 F + m_{10} E + m_{12} + m_{15} F \\ + d(m_5 F + m_9 + m_{11} \bar{E} + m_8 E) \end{aligned}$$

**Solution**

The given problem is actually a six-variable one. The four-variable K-map and its minimization are shown in Figure 6.64.

The minterm  $m_6$  contains  $E + \bar{E}$ . Its E is used to make a 4-square with the E's of  $m_0$ ,  $m_2$ , and  $m_4$ . Since  $m_6$  is not fully covered, it is combined with  $m_2$  to make a 2-square. The E of  $m_{12}$  can be combined with the E's of  $m_0$ ,  $m_4$ , and  $m_8$  but this is not done because  $m_{12}$  cannot be fully covered by this operation. So,  $m_{12}$  has to be read anyway. The minterm  $m_0$  is not fully covered. Only E of it is used. So, combine it with  $m_2$  to form a 2-square.

From the K-map, the reduced expression is, therefore, obtained as

$$f = \bar{B}DE + \bar{A}\bar{D}E + \bar{A}\bar{B}\bar{D} + \bar{A}\bar{C}DF + BCDF + AB\bar{C}\bar{D} + \bar{A}\bar{C}\bar{D}$$

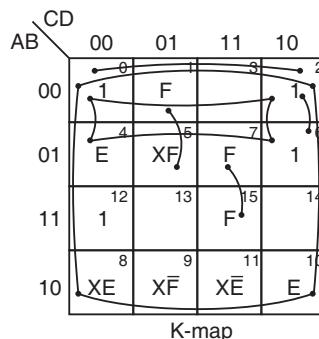


Figure 6.64 Example 6.36: K-map.

## 6.12 LIMITATIONS OF KARNAUGH MAPS

The K-map method of simplification is convenient as long as the number of variables does not exceed six. As the number of variables increases it becomes difficult to make judgments about which combinations form the minimal expression. It is almost impossible to work with problems of 7 or more variables using K-maps. Another important point is that the K-map simplification is a manual technique and simplification process is heavily dependent on the human abilities. It cannot be programmed. To meet this need, W.V. Quine and E.J. McCluskey developed an exact tabular method to simplify the Boolean expressions. This method is called the Quine–McCluskey method or tabular method.

## 6.13 IMPLEMENTATION OF LOGIC FUNCTIONS

### 6.13.1 Two-level Implementation

The implementation of a logic expression such that each one of the inputs has to pass through only two gates to reach the output is called *two-level implementation*. Both SOP and POS forms result in two-level logic. Two-level implementation can be with AND and OR gates or with only NAND gates or with only NOR gates.

The implementation of Boolean expressions with only NAND gates requires that the function be in SOP form. The implementation of the function

$$F = AB + CD$$

with (a) AND-OR logic and (b) with NAND-NAND logic is shown in Figure 6.65.

$$F = AB + CD = \overline{\overline{AB} + \overline{CD}} = \overline{\overline{AB}} \cdot \overline{\overline{CD}}$$

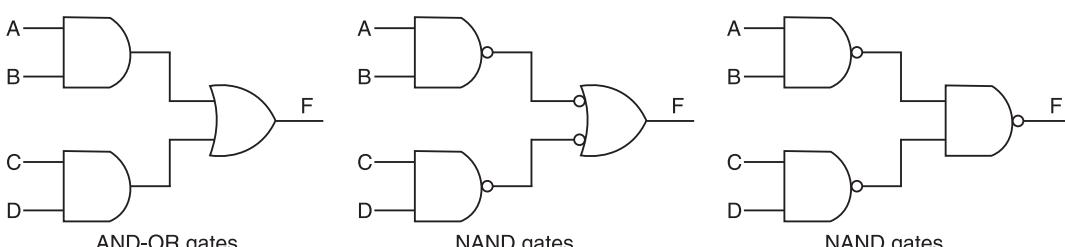
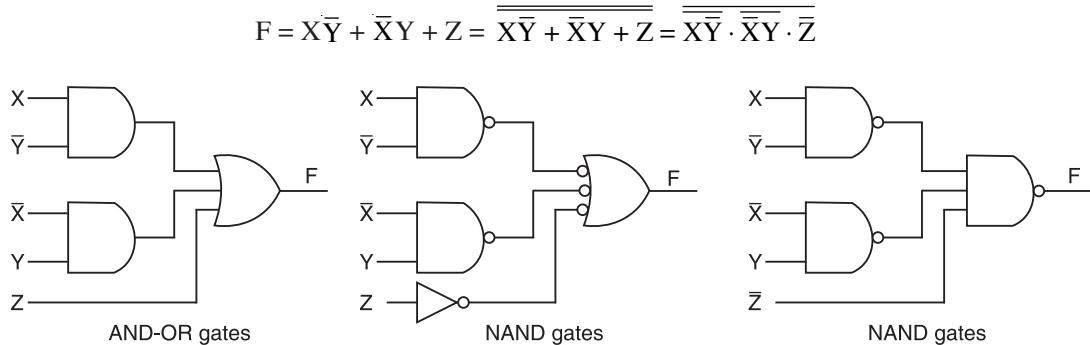


Figure 6.65 Two-level implementation using AND-OR logic and NAND logic.

The implementation of the function

$$F = X\bar{Y} + \bar{X}Y + Z$$

with (a) AND-OR logic and (b) NAND-NAND logic is shown in Figure 6.66.

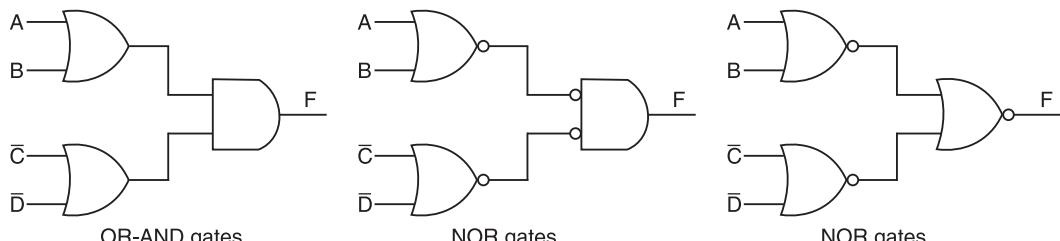


**Figure 6.66** Two-level implementation using AND-OR logic and NAND logic.

The implementation of Boolean expressions with only NOR gates requires that the function be in POS form. Implementation of the function

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

with (a) OR-AND logic and (b) NOR-NOR logic is shown in Figure 6.67.

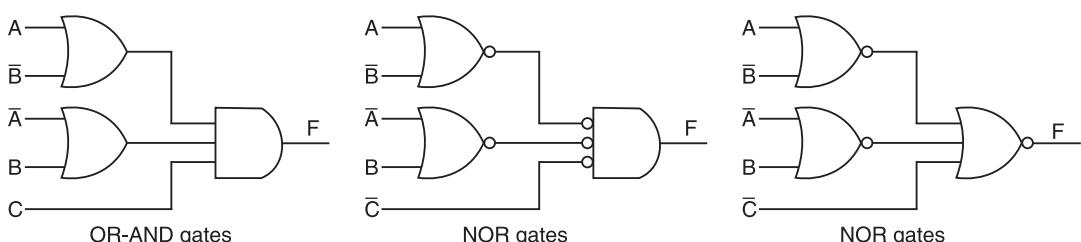


**Figure 6.67** Two-level implementation using OR-AND logic and NOR logic.

The implementation of the function

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

with (a) OR-AND logic and (b) NOR logic is shown in Figure 6.68.



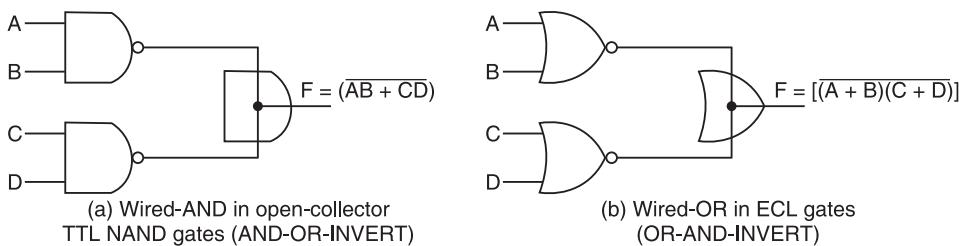
**Figure 6.68** Two-level implementation using OR-AND logic and NOR logic.

### 6.13.2 Other Two-level Implementations

The types of gates most often found in ICs are NAND and NOR. For this reason, NAND and NOR logic implementations are the most important from a practical point of view. Some NAND or NOR gates (but not all) allow the possibility of wire connection between the outputs of two gates to provide a specific logic function. This type of logic is called wired logic. For example, open collector TTL NAND gates, when tied together, perform the wired-AND logic. The wired-AND logic performed with two NAND gates is shown in Figure 6.69a. The AND gate is drawn with lines going through the centre of the gate to distinguish it from a conventional gate. The wired-AND gate is not a physical gate, but only a symbol to designate the function obtained from the indicated wired connection. The logic function implemented by the circuit of Figure 6.69a is

$$F = (\overline{AB}) \cdot (\overline{CD}) = (\overline{AB + CD})$$

and is called an AND-OR-Invert function.



**Figure 6.69** Wired AND and wired OR gates.

Similarly, the NOR outputs of ECL gates can be tied together to perform a wired NOR function. The logic function implemented by the circuit of Figure 6.69b is

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

and is called an OR-AND-INVERT function.

A wired logic gate does not produce a second level gate since it is just a wire connection, but these circuits are equivalent to two-level implementations. The first level consists of NAND (or NOR) gates and the second level has a single AND (or OR) gate.

## 6.14 NONDEGENERATE FORMS

If we consider four types of gates, i.e. AND, OR, NAND, and NOR and assign one type of gate for the first level and one type of gate for the second level, we find that there are 16 possible combinations of two-level forms (The same type of gate can be in the first and second levels, as in NAND-NAND implementation).

Eight of these combinations are said to be degenerate forms because they degenerate to a single operation. This can be seen from the circuit with AND gates in the first level and an AND gate in the second level. The output of the circuit is nearly the AND function of all input variables. The other eight nondegenerate forms produce an implementation in SOP or POS. The eight nondegenerate forms are as follows:

AND-OR  
NAND-NAND

OR-AND  
NOR-NOR

NOR-OR  
OR-NAND

NAND-AND  
AND-NOR

The first gate listed in each of the forms constitutes a first level in the implementation. The second gate listed is a single gate placed in the second level. Note that any two forms listed in the same line are duals of each other. The AND-OR and OR-AND forms are the basic two-level forms discussed earlier. The NAND-NAND and NOR-NOR forms were also discussed earlier. The remaining four forms are investigated below.

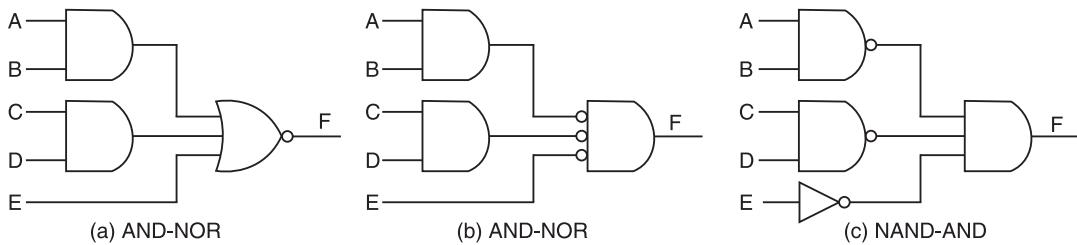
#### 6.14.1 AND-OR-INVERT Implementation

The two forms NAND-AND and AND-NOR are equivalent forms and can be treated together. Both perform the AND-OR-INVERT function as shown in Figure 6.70a. The AND-NOR form resembles the AND-OR form with an inversion done by the bubble in the output of the NOR gate. It implements the function

$$F = \overline{(AB + CD + E)}$$

By using the alternate graphic symbol for the NOR gate, we obtain the diagram of Figure 6.70b. Note that the single variable E is not complemented because the only change made is in the graphic symbol of the NOR gate. Now we move the bubble from the input terminal of the second-level gate to the output terminal of the first-level gate. An inverter is needed for the single variable to compensate for the bubble. Alternatively, the inverter can be removed provided input E is complemented. The circuit of Figure 6.70c is a NAND-AND form and implements the AND-OR-INVERT function.

An AND-OR-implementation requires an expression in sum of products. The AND-OR-INVERT implementation is similar except inversion. Therefore, if the complement of the function is simplified in SOP (by combining the 0s of the map), it will be possible to implement  $\bar{F}$  with the AND-OR part of the function. When  $\bar{F}$  passes through the always present output inversion (the INVERT part), it will generate the output F of the function.



**Figure 6.70** Two-level implementation in AND-NOR form and NAND-AND form.

#### 6.14.2 OR-AND-INVERT Implementation

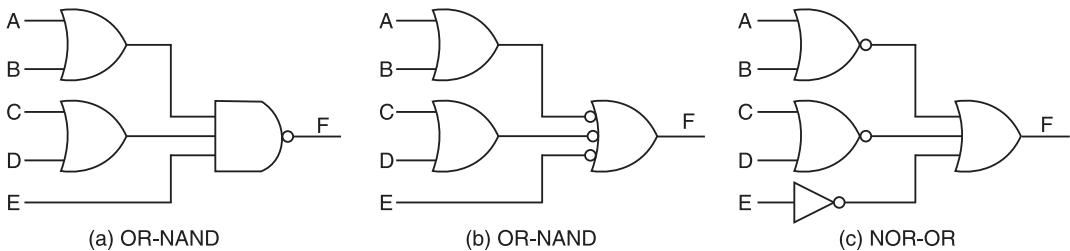
The OR-NAND and NOR-OR forms perform the OR-AND-INVERT functions. This is shown in Figure 6.71. The OR-NAND form resembles the OR-AND form, except for the inversion done by the bubble in the NAND gate. It implements the function

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

By using the alternate graphic symbol for the NAND gate, we obtain the diagram of Figure 6.71b. The circuit in Figure 6.71c is obtained by moving the small circles from the inputs

of the second-level gate to the outputs of the first-level gates. The circuit of Figure 6.71c is a NOR-OR form and was shown to implement the AND-OR-INVERT function.

The OR-AND-INVERT implementation requires an expression in POS. If the complement of the function is simplified in POS, we can implement  $\bar{F}$  with the OR-AND part of the function. When  $\bar{F}$  passes through the INVERT part we obtain the complement of  $\bar{F}$ , or  $F$ , in the output.



**Figure 6.71** Two-level implementation in OR-NAND form and NOR-OR form.

#### Tabular summary

Table 6.3 summarizes the procedures for implementing a Boolean function in any one of the four two-level forms. Because of the INVERT part in each case, it is convenient to use the simplification of  $\bar{F}$  (the complement) of the function. When  $\bar{F}$  is implemented in one of these forms, we obtain the complement of the function in the AND-OR or OR-AND form. The four two-level forms invert this function, giving an output that is the complement of  $\bar{F}$ . This is the normal output  $F$ .

**Table 6.3**

Equivalent nondegenerate form		Implements the function	Simplify $\bar{F}$ in	To get an output of
(a)	(b)*			
AND-NOR	NAND-AND	AND-OR-INVERT	Sum of products by combining 0s in the map	$F$
OR-NAND	NOR-OR	OR-AND-INVERT	Product of sums by combining 1s in the map and then complementing	$F$

\*Form (b) requires an inverter for a single literal term.

**EXAMPLE 6.37** Implement the function  $F = \sum m(0, 6)$  with the four two-level forms listed below.

AND-NOR OR-NAND	NAND-AND NOR-OR
--------------------	--------------------

**Solution**

The K-map, its minimization and the minimal expressions for  $\bar{F}$  in SOP and POS forms obtained from it are shown in Figure 6.72a. For AND-NOR and NAND-AND realizations,

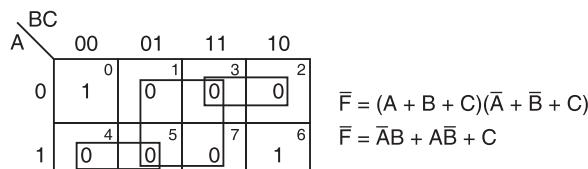
the function should be in AND-OR-INVERT form. The complement of the function is simplified in sum of products by combining the 0s in the map. So

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

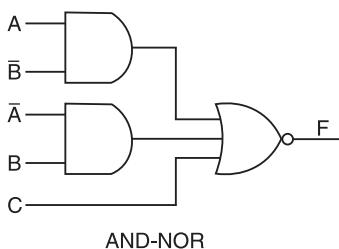
The normal output for this function can be expressed as

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

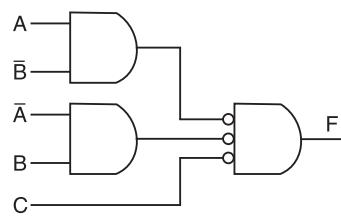
which is in the AND-OR-INVERT form. The AND-NOR and NAND-AND implementations are shown in Figure 6.72b. Observe that a one-input NAND or inverter gate is needed in the NAND-AND implementation, but not in the AND-NOR case. The inverter can be removed if we apply the input variable  $\bar{C}$  instead of C.



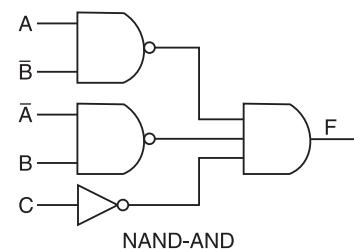
(a) Map simplification in sum of products



AND-NOR

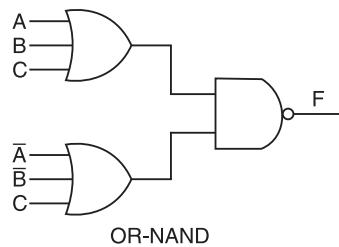


AND-NOR

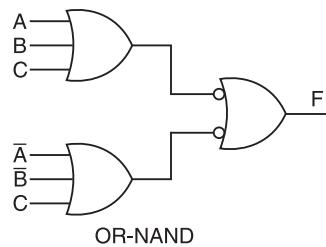


NAND-AND

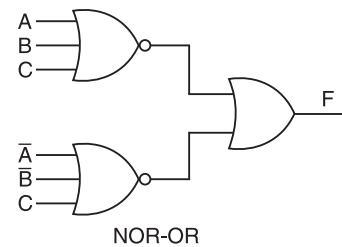
$$(b) F = (\bar{A}\bar{B} + A\bar{B} + C)$$



OR-NAND



OR-NAND



NOR-OR

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

**Figure 6.72** K-map, AND-NOR and NAND-AND form, and OR-NAND and NOR-OR form.

For OR-NAND and NOR-OR realization, the function should be in OR-AND-INVERT form. The OR-AND-INVERT forms require a simplified expression of the function in product of sums. The complement of the function is simplified in product of sums by combining 1s on the map. To obtain this expression first combine the 1s in the map. So

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

The normal output  $F$  can now be expressed as

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

which is in the OR-AND-INVERT form. From this expression, we can implement the function in OR-NAND and NOR-OR forms as shown in Figure 6.72c.

**EXAMPLE 6.38** Implement the function  $F$  with the following four two-level forms:

- |              |             |
|--------------|-------------|
| (a) NAND-AND | (b) AND-NOR |
| (c) OR-AND   | (d) NOR-OR  |

$$F(A, B, C, D) = \Sigma m(0, 1, 2, 3, 4, 8, 9, 12)$$

**Solution**

The K-map, its minimization and the minimal expressions for  $\bar{F}$  in SOP and POS forms obtained from it are shown in Figure 6.73.

For AND-NOR and NAND-AND form implementation the function should be in AND-OR-INVERT form. So simplify the complement of the function in sum of products by combining the 0s in the map. So

$$\bar{F} = AC + BD + BC$$

The normal output for this function can be expressed as

$$F = \overline{(AC + BD + BC)}$$

which is in the AND-OR-INVERT form. The AND-NOR and NAND-AND implementations are shown in Figure 6.74a. A NOR gate is nothing but a bubbled AND gate.

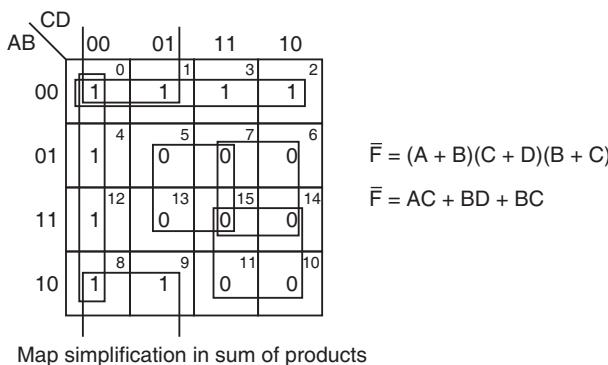
For OR-NAND and NOR-OR form implementation, the function should be in OR-AND-INVERT form. So simplify the function  $\bar{F}$  in product of sums by combining 1s on the K-map. So

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

The normal output  $F$  can now be expressed as

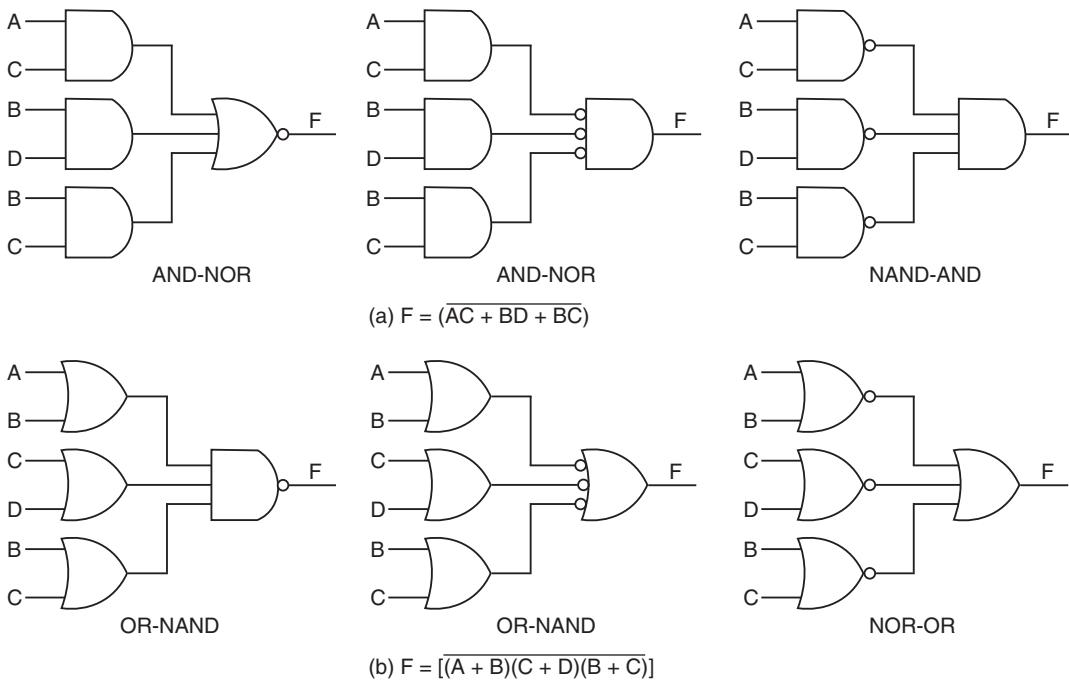
$$F = \overline{[(A + B)(C + D)(B + C)]}$$

which is in the OR-AND-INVERT form. The implementation of this function in the OR-NAND and NOR-OR forms is as shown in Figure 6.74b. A NAND gate is nothing but a bubbled OR gate.



Map simplification in sum of products

**Figure 6.73** K-map, Example 6.38.



**Figure 6.74** AND-NOR and NAND-AND form, and OR-NAND and NOR-OR forms.

## 6.15 QUINE-McCLUSKEY METHOD

The minimization of Boolean expressions using K-maps is usually limited to a maximum of six variables. The Quine–McClusky method, also known as the *tabular method*, is a more systematic method of minimizing expressions of even larger number of variables. This is suitable for hand computation as well as computation by machines, i.e. it is programmable.

The fundamental idea on which this tabulation procedure is based is that, repeated application of the combining theorem  $PA + P\bar{A} = P$  (where  $P$  is a set of literals) on all adjacent pairs of terms, yields the set of all prime implicants, from which a minimal sum may be selected.

Consider the minimization of the expression

$$\Sigma m(0, 1, 4, 5) = \overline{ABC} + \overline{ABC} + \overline{ABC} + \overline{ABC}$$

The first two terms, and the third and fourth terms can be combined to yield

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

This expression can further be reduced to

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

In the first step, we combined two pairs of adjacent terms, each of three literals per term, into two terms each of two literals. In the second step, these two terms are combined again and reduced to one term of a single variable.

The same result can be obtained by combining  $m_0$  and  $m_4$ , and  $m_1$  and  $m_5$  in the first step and the resulting terms in the second step. Minterms  $m_0(\bar{A}\bar{B}\bar{C})$  and  $m_1(\bar{A}\bar{B}C)$  are adjacent to each other because they differ in only literal C. Similarly, minterms  $m_4(A\bar{B}\bar{C})$  and  $m_5(A\bar{B}C)$  are adjacent to each other because they differ in only one literal C. Minterms  $m_0(\bar{A}\bar{B}\bar{C})$  and  $m_5(A\bar{B}C)$  or  $m_1(\bar{A}BC)$  and  $m_4(A\bar{B}\bar{C})$  cannot be combined, being not adjacent to each other since they differ in more than one variable. If we consider the binary representation of minterms,  $m_0(0\ 0\ 0)$  and  $m_1(0\ 0\ 1)$ , i.e. 0 0 0 and 0 0 1, they differ in only one position. When combined, they result in 0 0 –, i.e. variable C is absorbed. Similarly,  $m_4(1\ 0\ 0)$  and  $m_5(1\ 0\ 1)$ , i.e. 1 0 0 and 1 0 1 differ in only one position. So, when combined, they result in 1 0 –. Now 0 0 – and 1 0 – can be combined because they differ in only one position. The result is a – 0 –.

For the binary representation of two minterms to be different in just one position, it is necessary (but not sufficient) that the number of 1s in those two minterms differs exactly by one. Consequently, to facilitate the combination process, the minterms are arranged in groups according to the number of 1s in their binary representation.

The procedure for the minimization of a Boolean expression by the tabular method may, therefore, be described as follows.

*Step 1.* List all the minterms.

*Step 2.* Arrange all minterms in groups of the same number of 1s in their binary representation in column 1. Start with the least number of 1s group and continue with groups of increasing number of 1s. The number of 1s in a term is called the *index* of the term. The number of 1s in the binary form of a minterm is also called its *weight*.

*Step 3.* Compare each term of the lowest index group with every term in the succeeding group. Whenever possible, combine the two terms being compared by means of the combining theorem. Two terms from adjacent groups are combinable, if their binary representations differ by just a single digit in the same position; the combined terms consist of the original fixed representation with the differing one replaced by a dash (–). Place a check mark (✓) next to every term, which has been combined with at least one term (each term may be combined with several terms, but only a single check is required) and write the combined terms in column 2. Repeat this by comparing each term in a group of index  $i$  with every term in the group of index  $i + 1$ , until all possible applications of the combining theorem have been exhausted.

*Step 4.* Compare the terms generated in step 2 in the same fashion; combine two terms which differ by only a single 1 and whose dashes are in the same position to generate a new term. Two terms with dashes in different positions cannot be combined. Write the new terms in column 3 and put a check mark next to each term which has been combined in column 2. Continue the process with terms in columns 3, 4 etc. until no further combinations are possible. The remaining unchecked terms constitute the set of prime implicants of the expression. They are called *prime implicants* because they are not covered by any other term with fewer literals.

*Step 5.* List all the prime implicants and draw the *prime implicant chart*. (The don't cares if any should not appear in the prime implicant chart).

*Step 6.* Obtain the *essential prime implicants* and write the minimal expression.

### 6.15.1 The Decimal Representation

The tabulation procedure can be further simplified by adopting the decimal code for the minterms rather than the binary representation. Two minterms can be combined, only if they differ by a power of 2; that is, the difference between their decimal codes is  $2^i$ , namely 1, 2, 4, 8, and so on. The combined term consists of the same literals as the minterms with the exception of the variable whose weight is  $2^i$ , being deleted. For example, minterms  $m_0$  (0000) and  $m_8$  (1000) differ by  $8 = 2^3$ . So, they can be combined and the combined term can be written as 0, 8 (8), instead of – 0 0 0. The 8 in the parentheses indicates that the variable A whose weight is 8 can be deleted.

The condition that the decimal codes of two combinable terms must differ by a power of 2 is necessary but not sufficient. The terms whose codes differ by a power of 2, but which have the same index cannot be combined, since they differ by more than one variable. For example minterms 1 and 2, 2 and 4, 4 and 8, 10 and 12, etc. cannot be combined. Similarly, if a term with a smaller index has a higher decimal value than another term whose index is higher, then the two terms cannot be combined although they may differ by a power of 2; for example, minterms 9 and 7, 17 and 13, 20 and 19, 25 and 23, etc. cannot be combined although they differ by a power of 2. Except for the above phenomenon, the tabulation procedure using decimal representation is completely analogous to that using binary representation. In practice, only the decimal representations are used.

### 6.15.2 Don't Cares

When don't care terms are present in an expression, during the process of generating the set of prime implicants, don't care combinations are regarded as true combinations, i.e. combinations for which the expression assumes a 1. This, in effect, increases to the maximum, the number of possible prime implicants. The don't care terms are, however, not considered in the next step of selecting a minimum set of prime implicants, i.e. don't care minterms are not listed as column headings in the prime implicant chart since they do not have to be covered by the minimal expression. By not listing them, we actually leave the specification of don't care terms open. The prime implicant chart, thus, yields a minimal of an expression, which covers all the specified minterms.

### 6.15.3 The Prime Implicant Chart

The prime implicant chart is a pictorial representation of the relationships between the prime implicants and the minterms of the expression. It consists of an array of  $u$  rows and  $v$  columns where  $u$  and  $v$  designate the number of prime implicants and the number of minterms for which the expression takes on the value of 1 respectively. The entries of the  $i$ th row consist of  $\times$ s placed at the intersections with the columns corresponding to minterms covered by the  $i$ th prime implicant. The don't cares are not entered in the prime implicant chart.

### 6.15.4 Essential Prime Implicants

Essential prime implicants are the implicants which will definitely occur in the final expression. Any row in a prime implicant chart which has at least one minterm that is not present in any other row is called the *essential row* and the corresponding prime implicant is called the *essential prime implicant*. In other words, if any column contains a single  $\times$ , the prime implicant corresponding to the row in which this  $\times$  appears is the essential prime implicant.

Once an essential prime implicant has been selected, all the minterms it covers are checked off. After all the essential prime implicants and their corresponding columns have been checked off, if all the minterms are covered, the union of all the essential prime implicants yields the minimal expression. If this is not the case, additional prime implicants are necessary. We have to draw a reduced PI chart and find the minimal set of PIs from that.

#### 6.15.5 Dominating Rows and Columns

Two rows (or columns) I and J of a prime implicant chart, which have  $\times$ s in exactly the same columns (or rows) are said to be equal (written  $I = J$ ).

A column I in a prime implicant chart is said to dominate another column J of that chart, if column I has a  $\times$  in every row in which column J has a  $\times$ . Any minimal expression derived from a chart which contains both columns I and J can be obtained from a chart containing the dominated column. Hence, if column I dominates column J, then column I can be deleted from the chart without affecting the search for a minimal expression.

A row I in a prime implicant chart is said to dominate another row J, if row I has a  $\times$  in every column in which row J has a  $\times$ . Any minimal expression derived from a chart which contains both rows I and J can be derived from a chart which contains only the dominating row. Hence, if row I dominates row J, then row J can be deleted from the chart without affecting the search for a minimal expression.

#### 6.15.6 Determination of Minimal Expressions in Complex Cases

In simple cases, we can determine the minimal expression by simply inspecting the prime implicant chart. In more complex cases, however, the inspection method becomes prohibitive. Here, the procedure is:

*Step 1.* Determine the essential prime implicants from the prime implicant chart.

*Step 2.* Form a reduced prime implicant chart by removing all essential prime implicants and the columns covered by them. Although, none of the rows in the reduced chart is essential, only some of them may be removed.

*Step 3.* Remove all the dominating columns and the dominated rows of this reduced chart and form a new reduced chart.

*Step 4.* Look for the secondary essential prime implicants in the new reduced chart, and form another chart by removing the secondary essential prime implicants and the columns covered by them and write the minimal expression in SOP form. Continue the process, if required.

#### 6.15.7 The Branching Method

If the prime implicant chart has no essential prime implicants, dominated rows and dominated columns, the minimal expression can be obtained by a different approach called the *branching method*. Here, we consider any column and note the rows which cover that column. Make an arbitrary selection of one of those rows and apply the normal reduction procedure for the prime implicant chart without this row and the selected column and the columns covered by this row. The entire procedure is repeated for each row. Take the minimal of all such expressions obtained.

**EXAMPLE 6.39** Obtain the set of prime implicants for the Boolean expression  $f = \sum m(0, 1, 6, 7, 8, 9, 13, 14, 15)$  using the tabular method.

**Solution**

Group the minterms in terms of the number of 1s present in them and write their binary designations. The procedure to obtain the prime implicants is shown in Table 6.4.

**Table 6.4** Example 6.39

Column 1			Column 2				Column 3					
Index	Minterm	Binary designation	Pairs	A	B	C	D	Quads	A	B	C	D
Index 0	0	0 0 0 0 ✓	0, 1 (1)	0	0	0	–	✓	0, 1, 8, 9 (1, 8) – 0 0 – Q			
Index 1	1	0 0 0 1 ✓	0, 8 (8)	–	0	0	0	✓	...	...	...	...
	8	1 0 0 0 ✓	1, 9 (8)	–	0	0	1	✓	6, 7, 14, 15 (1, 8) – 1 1 – P			
Index 2	6	0 1 1 0 ✓	8, 9 (1)	1	0	0	–	✓				
	9	1 0 0 1 ✓	6, 7 (1)	0	1	1	–	✓				
Index 3	7	0 1 1 1 ✓	6, 14 (8)	–	1	1	0	✓				
	13	1 1 0 1 ✓	9, 13 (4)	1	–	0	1	S				
	14	1 1 1 0 ✓	7, 15 (8)	–	1	1	1	✓				
Index 4	15	1 1 1 1 ✓	13, 15 (2)	1	1	–	1	R				
			14, 15 (1)	1	1	1	–	✓				

Comparing the terms of index 0 with the terms of index 1 of column 1,  $m_0(0000)$  is combined with  $m_1(0001)$  to yield 0, 1 (1), i.e. 000 –. This is recorded in column 2 and 0000 and 0001 are checked off in column 1.  $m_0(0000)$  is combined with  $m_8(1000)$  to yield 0, 8 (8), i.e. – 000. This is recorded in column 2 and 1000 is checked off in column 1. Note that 0000 of column 1 has already been checked off. No more combinations of terms of index 0 and index 1 are possible. So, draw a line below the last combination of these groups, i.e. below 0, 8 (8), – 000 in column 2. Now 0, 1 (1), i.e. 000 – and 0, 8 (8), i.e. – 000 are the terms in the first group of column 2.

Comparing the terms of index 1 with the terms of index 2 in column 1,  $m_1(0001)$  is combined with  $m_9(1001)$  to yield 1, 9 (8), i.e. – 001. This is recorded in column 2 and 1001 is checked off in column 1 because 0001 has already been checked off.  $m_8(1000)$  is combined with  $m_9(1001)$  to yield 8, 9 (1), i.e. 100 –. This is recorded in column 2. 1000 and 1001 of column 1 have already been checked off. So, no need to check them off again. No more combinations of terms of index 1 and index 2 are possible. So, draw a line below the last combination of these groups, i.e. 8, 9 (1), – 001 in column 2. Now 1, 9 (8), i.e. – 001 and 8, 9 (1), i.e. 100 – are the terms in the second group of column 2.

Similarly, comparing the terms of index 2 with the terms of index 3 in column 1,  $m_6(0110)$  and  $m_7(0111)$  yield 6, 7 (1), i.e. 011-. Record it in column 2 and check off 6(0110) and 7(0111).

$m_6(0110)$  and  $m_{14}(1110)$  yield 6, 14 (8), i.e. -110. Record it in column 2 and check off 6(0110) and 14(1110).

$m_9(1001)$  and  $m_{13}(1101)$  yield 9, 13 (4), i.e. 1-01. Record it in column 2 and check off 9(1001) and 13(1101).

So, 6, 7 (1), i.e. 011-, and 6, 14 (8), i.e. -110 and 9, 13 (4), i.e. 1-01 are the terms in group 3 of column 2. Draw a line at the end of 9, 13 (4), i.e. 1-01.

Also, comparing the terms of index 3 with the terms of index 4 in column 1,

$m_7(0111)$  and  $m_{15}(1111)$  yield 7, 15 (8), i.e. -111. Record it in column 2 and check off 7(0111) and 15(1111).

$m_{13}(1101)$  and  $m_{15}(1111)$  yield 13, 15 (2), i.e. 11-1. Record it in column 2 and check off 13 and 15.

$m_{14}(1110)$  and  $m_{15}(1111)$  yield 14, 15 (1), i.e. 111-. Record it in column 2 and check off 14 and 15.

So, 7, 15 (8), i.e. -111, and 13, 15 (2), i.e. 11-1 and 14, 15 (1), i.e. 111- are the terms in group 4 of column 2. Column 2 is completed now.

Comparing the terms of group 1 with the terms of group 2 in column 2, the terms 0, 1 (1), i.e. 000- and 8, 9 (1), i.e. 100- are combined to form 0, 1, 8, 9 (1, 8), i.e. -00-. Record it in group 1 of column 3 and check off 0, 1 (1), i.e. 000-, and 8, 9 (1), i.e. 100- of column 2. The terms 0, 8 (8), i.e. -000 and 1, 9 (8), i.e. -001 are combined to form 0, 1, 8, 9 (1, 8), i.e. -00-. This has already been recorded in column 3. So, no need to record again. Check off 0, 8 (8), i.e. -000 and 1, 9 (8), i.e. -001 of column 2. Draw a line below 0, 1, 8, 9 (1, 8), i.e. -00-. This is the only term in group 1 of column 3. No term of group 2 of column 2 can be combined with any term of group 3 of column 2. So, no entries are made in group 2 of column 2.

Comparing the terms of group 3 of column 2 with the terms of group 4 of column 2, the terms 6, 7 (1), i.e. 011-, and 14, 15 (1), i.e. 111- are combined to form 6, 7, 14, 15 (1, 8), i.e. -11-. Record it in group 3 of column 3 and check off 6, 7 (1), i.e. 011- and 14, 15 (1), i.e. 111- of column 2. The terms 6, 14 (8), i.e. -110 and 7, 15 (8), i.e. -111 are combined to form 6, 7, 14, 15 (1, 8), i.e. -11-. This has already been recorded in column 3; so, check off 6, 14 (8), i.e. -110 and 7, 15 (8), i.e. -111 of column 2.

Observe that the terms 9, 13 (4), i.e. 1-01 and 13, 15 (2), i.e. 11-1 cannot be combined with any other terms. Similarly in column 3, the terms 0, 1, 8, 9 (1, 8), i.e. -00- and 6, 7, 14, 15 (1, 8), i.e. -11- cannot also be combined with any other terms. So, these 4 terms are the prime implicants.

The terms, which cannot be combined further, are labelled as P, Q, R, and S. These form the set of prime implicants.

**EXAMPLE 6.40** Obtain the minimal expression for  $f = \Sigma m(1, 2, 3, 5, 6, 7, 8, 9, 12, 13, 15)$  using the tabular method.

### Solution

The procedure to obtain the set of prime implicants is illustrated in Table 6.5.

**Table 6.5** Example 6.40

Column 1		Column 2	Column 3	
Index	Minterm	Pairs	Quads	
Index 1	1 ✓	1, 3 (2) ✓	1, 3, 5, 7 (2, 4)	T
	2 ✓	1, 5 (4) ✓	1, 5, 9, 13 (4, 8)	S
	8 ✓	1, 9 (8) ✓	2, 3, 6, 7 (1, 4)	R
Index 2	3 ✓	2, 3 (1) ✓	8, 9, 12, 13 (1, 4)	Q
	5 ✓	2, 6 (4) ✓	5, 7, 13, 15 (2, 8)	P
	6 ✓	8, 9 (1) ✓		
	9 ✓	8, 12 (4) ✓		
	12 ✓	3, 7 (4) ✓		
Index 3	7 ✓	5, 7 (2) ✓		
	13 ✓	5, 13 (8) ✓		
Index 4	15 ✓	6, 7 (1) ✓		
		9, 13 (4) ✓		
		12, 13 (1) ✓		
		7, 15 (8) ✓		
		13, 15 (2) ✓		

The non-combinable terms P, Q, R, S and T are recorded as prime implicants.

$$P \rightarrow 5, 7, 13, 15 (2, 8) = X 1 X 1 = BD$$

(Literals with weights 2 and 8, i.e. C and A are deleted. The lowest minterm is  $m_5$  ( $5 = 4 + 1$ ). So, literals with weights 4 and 1, i.e. B and D are present in non-complemented form. So, read it as BD.)

$$Q \rightarrow 8, 9, 12, 13 (1, 4) = 1 X 0 X = AC$$

(Literals with weights 1 and 4, i.e. D and B are deleted. The lowest minterm is  $m_8$ . So, literal with weight 8 is present in non-complemented form and literal with weight 2 is present in complemented form. So, read it as  $AC$ .)

$$R \rightarrow 2, 3, 6, 7 (1, 4) = 0 X 1 X = \bar{AC}$$

(Literals with weights 1 and 4, i.e. D and B are deleted. The lowest minterm is  $m_2$ . So, literal with weight 2 is present in non-complemented form and literal with weight 8 is present in complemented form. So, read it as  $\bar{AC}$ .)

$$S \rightarrow 1, 5, 9, 13 (4, 8) = X X 0 1 = \bar{CD}$$

(Literals with weights 4 and 8, i.e. B and A are deleted. The lowest minterm is  $m_1$ . So, literal with weight 1 is present in non-complemented form and literal with weight 2 is present in complemented form. So, read it as  $\bar{CD}$ .)

$$T \rightarrow 1, 3, 5, 7 (2, 4) = 0 X X 1 = \bar{AD}$$

(Literals with weights 2 and 4, i.e. C and B are deleted. The lowest minterm is 1. So, literal with weight 1 is present in non-complemented form and literal with weight 8 is present in complemented form. So, read it as  $\bar{A}D$ .)

The prime implicant chart of the expression

$$f = \Sigma m(1, 2, 3, 5, 6, 7, 8, 9, 12, 13, 15)$$

is as shown in Table 6.6. It consists of 11 columns corresponding to the number of minterms and 5 rows corresponding to the prime implicants P, Q, R, S, and T generated. Row R contains four  $\times$ s at the intersections with columns 2, 3, 6, and 7, because these minterms are covered by the prime implicant R. A row is said to cover the columns in which it has  $\times$ s. The problem now is to select a minimal subset of prime implicants, such that each column contains at least one  $\times$  in the rows corresponding to the selected subset and the total number of literals in the prime implicants selected is as small as possible. These requirements guarantee that the number of unions of the selected prime implicants is equal to the original number of minterms and that, no other expression containing fewer literals can be found.

**Table 6.6** Example 6.40: Prime implicant chart

	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PIs/Minterms	1	2	3	5	6	7	8	9	12	13	15
*P → 5, 7, 13, 15 (2, 8)					×		×			×	×
*Q → 8, 9, 12, 13 (1, 4)								×	×	×	×
*R → 2, 3, 6, 7 (1, 4)		×	×			×	×				
S → 1, 5, 9, 13 (4, 8)	×				×				×		×
T → 1, 3, 5, 7 (2, 4)	×		×	×		×					

In the prime implicant chart of Table 6.6,  $m_2$  and  $m_6$  are covered by R only. So, R is an essential prime implicant. So, check off all the minterms covered by it, i.e.  $m_2, m_3, m_6$ , and  $m_7$ . Q is also an essential prime implicant because only Q covers  $m_8$  and  $m_{12}$ . Check off all the minterms covered by it, i.e.  $m_8, m_9, m_{12}$ , and  $m_{13}$ . P is also an essential prime implicant, because  $m_{15}$  is covered only by P. So check off  $m_{15}, m_5, m_7$ , and  $m_{13}$  covered by it. Thus, only minterm 1 is not covered. Either row S or row T can cover it and both have the same number of literals. Thus, two minimal expressions are possible.

$$P + Q + R + S = BD + \bar{A}\bar{C} + \bar{A}C + \bar{C}D$$

or

$$P + Q + R + T = BD + \bar{A}\bar{C} + \bar{A}C + \bar{A}D$$

**EXAMPLE 6.41** Using the tabular method, obtain the minimal expression for

$$f = \Sigma m(6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15)$$

### **Solution**

In the given Boolean expression there are don't cares. Treat the don't cares as minterms and apply the usual procedure to obtain the set of prime implicants as shown in Table 6.7.

**Table 6.7** Example 6.41

Column 1		Column 2	Column 3	Column 4
Index	Minterm	Pairs	Quads	Octets
Index 1	8 ✓	8, 9 (1) ✓	8, 9, 10, 11 (1, 2) ✓	8, 9, 10, 11, 12, 13, 14, 15 (1, 2, 4) P
	6 ✓	8, 10 (2) ✓	8, 9, 12, 13 (1, 4) ✓	
	9 ✓	8, 12 (4) ✓	8, 10, 12, 14 (2, 4) ✓	
Index 2	10 ✓	6, 7 (1) ✓	6, 7, 14, 15 (1, 8) Q	
	12 ✓	6, 14 (8) ✓	9, 11, 13, 15, (2, 4) ✓	
	7 ✓	9, 11 (2) ✓	10, 11, 14, 15 (1, 4) ✓	
	11 ✓	9, 13 (4) ✓	12, 13, 14, 15 (1, 2) ✓	
Index 3	13 ✓	10, 11 (1) ✓		
	14 ✓	10, 14 (4) ✓		
Index 4	15 ✓	12, 13 (1) ✓		
		12, 14 (2) ✓		
		7, 15 (8) ✓		
		11, 15 (4) ✓		
		13, 15 (2) ✓		
		14, 15 (1) ✓		

From this table, we see that the prime implicants are  $P \rightarrow 8, 9, 10, 11, 12, 13, 14, 15 (1, 2, 4)$  and  $Q \rightarrow 6, 7, 14, 15 (1, 8)$ . The term  $6, 7, 14, 15 (1, 8)$  means that literals with weights 1 and 8, i.e. D and A are deleted and the lowest designated minterm is  $m_6(4 + 2)$ , i.e. literals with weights 4 and 2 are present in non-complemented form. So it is read as BC. The term  $8, 9, 10, 11, 12, 13, 14, 15 (1, 2, 4)$  means that literals with weights 1, 2, and 4, i.e. D, C, and B are deleted. The lowest designated minterm is, therefore,  $m_8$ . So, literal with weight 8 is present in non-complemented form. So, it is read as A.

In the prime implicant chart of  $\Sigma (6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15)$  shown in Table 6.8, all the don't care minterms are omitted.

**Table 6.8** Example 6.41: Prime implicant chart

	✓	✓	✓	✓
PIs/Minterms	6	7	8	9
* $P \rightarrow 8, 9, 10, 11, 12, 13, 14, 15 (1, 2, 4)$			✗	✗
* $Q \rightarrow 6, 7, 14, 15 (1, 8)$	✗	✗		

As seen from the table, P and Q are both essential prime implicants. So, the minimal expression is  $A + BC$ .

**EXAMPLE 6.42** Using the Quine–McCluskey method of tabular reduction minimize the given combinational single output function  $f(W, X, Y, Z) = \Sigma m(0, 1, 5, 7, 8, 10, 14, 15)$ .

**Solution**

Minimization using the tabular method is as shown in Table 6.9.

**Table 6.9** Example 6.42

Index	Minterm	Column 2	
		Pairs	
Index 0	0 ✓	0, 1 (1)	A
Index 1	1 ✓	0, 8 (8)	B
	8 ✓	1, 5 (4)	C
Index 2	5 ✓	8, 10 (2)	D
	10 ✓	5, 7 (2)	E
Index 3	7 ✓	10, 14 (4)	F
	14 ✓	7, 15 (8)	G
Index 4	15 ✓	14, 15 (1)	H

None of the terms in any group of step 2 can be combined with any other term in the next group. So all of them are prime implicants. The PI chart is shown in Table 6.10.

**Table 6.10** Example 6.42: Prime implicant chart

PIs/Minterms	0	1	5	7	8	10	14	15
A → 0, 1 (1)	×	×						
B → 0, 8 (8)	×				×			
C → 1, 5 (4)		×	×					
D → 8, 10 (2)					×	×		
E → 5, 7 (2)			×	×				
F → 10, 14 (4)						×	×	
G → 7, 15 (8)				×				×
H → 14, 15 (1)						×	×	

We can see from the PI chart that there are no essential prime implicants and one possible minimal combination of PIs that can cover all minterms is A, E, D, H. Therefore,

$$\begin{aligned}
 f_{\min} &= A + E + D + H = 000- + 01 - 1 + 10 - 0 + 111- = \bar{W}\bar{X}\bar{Y} + \bar{W}XZ + W\bar{X}\bar{Z} + WXY \\
 &= \overline{(\bar{W}\bar{X}\bar{Y})} \overline{(\bar{W}XZ)} \overline{(W\bar{X}\bar{Z})} \overline{(WXY)}
 \end{aligned}$$

**EXAMPLE 6.43** Minimize the following expression:

$$f = \Sigma m(0, 1, 2, 8, 9, 15, 17, 21, 24, 25, 27, 31)$$

**Solution**

Table 6.11 shows the procedure for obtaining all the prime implicants.

**Table 6.11** Example 6.43

Column 1		Column 2	Column 3
Index	Minterm	Pairs	Quads
Index 0	0 ✓	0, 1 (1) ✓	0, 1, 8, 9 (1, 8) R
	1 ✓	0, 2 (2) W	1, 9, 17, 25 (8, 16) Q
Index 1	2 ✓	0, 8 (8) ✓	8, 9, 24, 25 (1, 16) P
	8 ✓	1, 9 (8) ✓	
	9 ✓	1, 17 (16) ✓	
Index 2	17 ✓	8, 9 (1) ✓	
	24 ✓	8, 24 (16) ✓	
Index 3	21 ✓	9, 25 (16) ✓	
	25 ✓	17, 21 (4) V	
Index 4	15 ✓	17, 25 (8) ✓	
	27 ✓	24, 25 (1) ✓	
Index 5	31 ✓	25, 27 (2) U	
		15, 31 (16) T	
		27, 31 (4) S	

From Table 6.11 we see that the prime implicants are P, Q, R, S, T, U, V, and W. The prime implicant chart is shown in Table 6.12.

**Table 6.12** Example 6.43: Prime implicant chart

PIs/Minterms	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
*P → 8, 9, 24, 25				×	×			×	×		
Q → 1, 9, 17, 25		×			×		×			×	
R → 0, 1, 8, 9	×	×		×	×						
S → 27, 31										×	×
*T → 15, 31						×					×
U → 25, 27									×	×	
*V → 17, 21							×	×			
*W → 0, 2		×		×							

From the PI chart we observe that P, T, V, and W are the essential prime implicants because  $m_{24}$  is covered by P only,  $m_{15}$  is covered by T only,  $m_{21}$  is covered by V only and  $m_2$  is covered

by W only. Delete the essential prime implicants and the columns covered by them and form the reduced prime implicant chart as shown in Table 6.13.

**Table 6.13** Example 6.43: Reduced prime implicant chart

PIs/Minterms	1	27
Q	×	
R	×	
S		×
U		×

$m_1$  and  $m_{27}$  can be covered by  $Q + S$  or  $Q + U$  or  $R + S$  or  $R + U$ . So there are 4 minimal SOP expressions as shown below.

$$P + T + V + W + Q + S = \bar{B}\bar{C}\bar{D} + BCDE + A\bar{B}\bar{D}\bar{E} + \bar{A}\bar{B}\bar{C}\bar{E} + \bar{C}\bar{D}\bar{E} + ABDE$$

$$P + T + V + W + Q + U = \bar{B}\bar{C}\bar{D} + BCDE + A\bar{B}\bar{D}\bar{E} + \bar{A}\bar{B}\bar{C}\bar{E} + \bar{C}\bar{D}\bar{E} + AB\bar{C}\bar{E}$$

$$P + T + V + W + R + S = \bar{B}\bar{C}\bar{D} + BCDE + A\bar{B}\bar{D}\bar{E} + \bar{A}\bar{B}\bar{C}\bar{E} + \bar{A}\bar{C}\bar{D} + ABDE$$

$$P + T + V + W + R + U = \bar{B}\bar{C}\bar{D} + BCDE + A\bar{B}\bar{D}\bar{E} + \bar{A}\bar{B}\bar{C}\bar{E} + \bar{A}\bar{C}\bar{D} + AB\bar{C}\bar{E}$$

**EXAMPLE 6.44** Find the minimal expression for  $f = \prod M(2, 3, 8, 12, 13) \cdot d(10, 14)$ .

#### *Solution*

The tabulation method for the POS form is exactly the same as that for the SOP form. Treat the maxterms as if they are the minterms and complete the process. While writing the minimal expression in the POS form, treat the non-complemented variable as a 0 and the complemented variable as a 1 and write the terms in sum form. The prime implicants are obtained as shown in Table 6.14.

**Table 6.14** Example 6.44

Column 1		Column 2	Column 3
Index	Minterm	Pairs	Quads
Index 1	2 ✓	2, 3 (1) S	8, 10, 12, 14 (2, 4) P
	8 ✓	2, 10 (8) R	
	3 ✓	8, 10 (2) ✓	
Index 2	10 ✓	8, 12 (4) ✓	
	12 ✓	10, 14 (4) ✓	
Index 3	13 ✓	12, 13 (1) Q	
	14 ✓	12, 14 (2) ✓	

There are four prime implicants P, Q, R, and S. Draw the prime implicant chart as shown in Table 6.15.

**Table 6.15** Example 6.44: Prime implicant chart

	✓	✓	✓	✓	✓
PIs/Minterms	2	3	8	12	13
*P → 8, 10, 12, 14 (2, 4)			×	×	
*Q → 12, 13 (1)				×	×
R → 2, 10 (8)	×				
*S → 2, 3 (1)	×	×	×		

In Table 6.15  $M_8$  is covered by P only. Further,  $M_{13}$  is covered by Q only and  $M_3$  is covered by S only. Therefore, P, Q, and S are the essential prime implicants. Check them out and also check the Maxterms covered by them. They cover all the Maxterms. So, the final expression is given by

$$f_{\min} = PQS = (\bar{A} + D)(\bar{A} + \bar{B} + C)(A + B + \bar{C}) \quad (11 \text{ gate inputs})$$

**EXAMPLE 6.45** Obtain the minimal POS expression for the following:

$$f = \prod M(0, 1, 4, 5, 9, 11, 13, 15, 16, 17, 25, 27, 28, 29, 31) \cdot d(20, 21, 22, 30)$$

**Solution**

The prime implicants are obtained as shown in Table 6.16.

**Table 6.16** Example 6.45

Column 1 Minterms	Column 2 Pairs	Column 3		Column 4 Octets
		Quads	Octets	
0 ✓	0, 1 (1) ✓	0, 1, 4, 5 (1, 4) ✓	0, 1, 4, 5, 16, 17, 20, 21 (1, 4, 16) R	
1 ✓	0, 4 (4) ✓	0, 1, 16, 17 (1, 16) ✓	1, 5, 9, 13, 17, 21, 25, 29 (4, 8, 16) Q	
4 ✓	0, 16 (16) ✓	0, 4, 16, 20 (4, 16) ✓	9, 11, 13, 15, 25, 27, 29, 31 (2, 4, 16) P	
16 ✓	1, 5 (4) ✓	1, 5, 9, 13 (4, 8) ✓		
5 ✓	1, 9 (8) ✓	1, 5, 17, 21 (4, 16) ✓		
9 ✓	1, 17 (16) ✓	1, 9, 17, 25 (8, 16) ✓		
17 ✓	4, 5 (1) ✓	4, 5, 20, 21 (1, 16) ✓		
20 ✓	4, 20 (16) ✓	16, 17, 20, 21 (1, 4) ✓		
11 ✓	16, 17 (1) ✓	5, 13, 21, 29 (8, 16) ✓		
13 ✓	16, 20 (4) ✓	9, 11, 13, 15 (2, 4) ✓		
21 ✓	5, 13 (8) ✓	9, 11, 25, 27 (2, 16) ✓		
22 ✓	5, 21 (16) ✓	9, 13, 25, 29 (4, 16) ✓		
25 ✓	9, 11 (2) ✓	17, 21, 25, 29 (4, 8) ✓		
28 ✓	9, 13 (4) ✓	20, 21, 28, 29 (1, 8) U		
15 ✓	9, 25 (16) ✓	20, 22, 28, 30 (2, 8) T		
27 ✓	17, 21 (4) ✓	11, 15, 27, 31 (4, 16) ✓		
29 ✓	17, 25 (8) ✓	13, 15, 29, 31 (2, 16) ✓		

(Contd.)

**Table 6.16** Example 6.45 (*Contd.*)

Column 1	Column 2	Column 3	Column 4
Minterms	Pairs	Quads	Octets
<u>30 ✓</u>	20, 21 (1) ✓	25, 27, 29, 31 (2, 4) ✓	
<u>31 ✓</u>	20, 22 (2) ✓	28, 29, 30, 31 (1, 2) S	
	<u>20, 28 (8) ✓</u>		
	<u>11, 15 (4) ✓</u>		
	11, 27 (16) ✓		
	13, 15 (2) ✓		
	13, 29 (16) ✓		
	21, 29 (8) ✓		
	22, 30 (8) ✓		
	25, 27 (2) ✓		
	25, 29 (4) ✓		
	28, 29 (1) ✓		
	<u>28, 30 (2) ✓</u>		
	<u>15, 31 (16) ✓</u>		
	27, 31 (4) ✓		
	29, 31 (2) ✓		
	30, 31 (1) ✓		

From Table 6.16 we see that there are six prime implicants P, Q, R, S, T, and U. The prime implicant chart is shown in Table 6.17.

**Table 6.17** Example 6.45: Prime implicant chart

✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
0	1	4	5	9	11	13	15	16	17	25	27	28	29	31
*P					✗	✗	✗	✗		✗	✗		✗	✗
Q		✗		✗	✗		✗			✗	✗		✗	
*R	✗	✗	✗	✗					✗	✗				
S												✗	✗	✗
T												✗		
U												✗	✗	

Here  $M_{11}$ ,  $M_{15}$ , and  $M_{27}$  are covered by P only. So P is an essential prime implicant. Also  $M_0$ ,  $M_4$ , and  $M_{16}$  are covered by R only. So, R is an essential prime implicant. The only Maxterm left uncovered is  $M_{28}$ . It can be covered by S or T or U. So the minimum POS expression is given by PRS or PRT or PRU and each one requires ten gate inputs.

$$f_{\min} = PRS = (\bar{B} + \bar{E})(B + D)(\bar{A} + \bar{B} + \bar{C})$$

$$f_{\min} = PRT = (\bar{B} + \bar{E})(B + D)(\bar{A} + \bar{C} + E)$$

$$f_{\min} = PRU = (\bar{B} + \bar{E})(B + D)(\bar{A} + \bar{C} + D)$$

**EXAMPLE 6.46** Use the tabular procedure to simplify the given expression

$$F(V, W, X, Y, Z) = \Sigma m(0, 4, 12, 16, 19, 24, 27, 28, 29, 31)$$

in SOP form and draw the circuit using only NAND gates.

**Solution**

The tabular procedure is shown in Table 6.18.

**Table 6.18** Example 6.46

Column 1		Column 2	
Index	Minterms	Pairs	
Index 0	0 ✓	0, 4 (4)	A
Index 1	4 ✓	0, 16 (16)	B
	16 ✓	4, 12 (8)	C
Index 2	12 ✓	16, 24 (8)	D
	24 ✓	12, 28 (16)	E
Index 3	19 ✓	24, 28 (4)	F
	28 ✓	19, 27 (8)	G
Index 4	27 ✓	28, 29 (1)	H
	29 ✓	27, 31 (4)	I
Index 5	31 ✓	29, 31 (2)	J

No term in any group of step 2 can be combined with any other term in the next group. So all the terms in step 2 are prime implicants. The prime implicant chart is shown in Table 6.19.

**Table 6.19** Example 6.46: Prime implicant chart

PIs/Minterms	✓									
	0	4	12	16	19	24	27	28	29	31
A → 0, 4 (4)	✗	✗								
B → 0, 16 (16)	✗				✗					
C → 4, 12 (8)		✗	✗							
D → 16, 24 (8)				✗			✗			
E → 12, 28 (16)			✗						✗	
F → 24, 28 (4)						✗		✗		
G* → 19, 27 (8)					✗		✗			
H → 28, 29 (1)								✗	✗	
I → 27, 31 (4)						✗				✗
J → 29, 31 (2)								✗	✗	

In the PI chart  $m_{19}$  can be covered only by the prime implicant G. So G is an essential PI. It also covers  $m_{27}$ . From the PI chart we can observe that the remaining minterms are covered by the minimal set of prime implicants A, D, E, J. Therefore,

$$\begin{aligned} f_{\min} &= G + A + D + E + J = 1-011 + 00-00 + 1-000 + -1100 + 111-1 \\ &= \bar{V}\bar{X}YZ + \bar{V}\bar{W}\bar{Y}\bar{Z} + \bar{V}\bar{X}\bar{Y}\bar{Z} + W\bar{X}\bar{Y}\bar{Z} + VWXZ \\ &= \overline{(V\bar{X}YZ)} \overline{(V\bar{W}\bar{Y}\bar{Z})} \overline{(V\bar{X}\bar{Y}\bar{Z})} \overline{(W\bar{X}\bar{Y}\bar{Z})} \overline{(VWXZ)} \end{aligned}$$

The above functions can be realized using AOI logic and NAND logic.

**EXAMPLE 6.47** Apply branching method to simplify the following function:

$$f(A, B, C, D) = \Sigma m(2, 3, 4, 6, 9, 11, 12, 13)$$

**Solution**

The branching method is applied if the prime implicant chart contains no essential prime implicants, dominated rows, and dominating columns. The PI chart is obtained using the tabular method.

**Table 6.20** Example 6.47

Column 1		Column 2
Index	Minterm	Pairs
Index 1	2	2, 3 (1) W
	4	2, 6 (4) V
Index 2	3	4, 6 (2) U
	6	4, 12 (8) T
	9	3, 11 (8) S
	12	9, 11 (2) R
Index 3	11	9, 13 (4) Q
	13	12, 13 (1) P

There are eight prime implicants of equal size. The prime implicant chart is shown in Table 6.21.

**Table 6.21** Example 6.47: Prime implicant chart

PIs/Minterms	2	3	4	6	9	11	12	13
W → 2,3 (1)	x	x						
V → 2,6 (4)	x			x				
U → 4,6 (2)			x	x				
T → 4,12 (8)			x				x	
S → 3,11(8)		x			x			
R → 9,11 (2)					x	x		
Q → 9,13 (4)					x			x
P → 12,13( 1)						x	x	

From the prime implicant chart, we observe that all the minterms are covered by the prime implicants W, U, R and P. So the minimal expression is

$$f_{\min} = W + U + R + P = \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{D} + A\bar{B}\bar{D} + AB\bar{C}$$

*Branching method:* Since the PI chart (Table 6.21) contains no essential prime implicants, dominated rows, or dominating columns, the branching method can be used. For this we can select any column. Let us select the first column with minterm 2. This minterm can be covered by either row W or row V. Let us select row W first and remove that row and the columns covered by it, i.e. columns corresponding to minterms 2 and 3 and write the reduced PI chart shown in Table 6.22.

**Table 6.22** Example 6.47: Reduced chart after selection of row W

PIs/Minterms	4	6	9	11	12	13
V → 2,6 (4)		×				
U → 4,6 (2)	×	×				
T → 4,12 (8)	×				×	
S → 3,11(8)				×		
R → 9,11 (2)			×	×		
Q → 9,13 (4)			×			×
P → 12,13( 1)					×	×

In Table 6.22, row V is dominated by row U and row S is dominated by row R. So remove rows V and S. Once rows V and S are removed, column 4 dominates column 6 and column 9 dominates column 11. So remove columns 4 and 9 and draw the second reduced PI chart shown in Table 6.23. Now U and R are the essential secondary prime implicants and together with P they cover all the minterms. So the minimal expression is

$$f_{\min} = W + U + R + P = \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{D} + A\bar{B}\bar{D} + AB\bar{C}$$

This is one minimal expression.

**Table 6.23** Example 6.47: Reduced chart

PIs/Minterms	6	11	12	13
*U → 4,6 (2)	×			
T → 4,12 (8)			×	
*R → 9,11 (2)		×		
Q → 9,13 (4)				×
*P → 12,13( 1)			×	×

Now select row V and remove that row and the columns corresponding to minterms 2 and 6 covered by it and draw the reduced PI chart shown in Table 6.24.

**Table 6.24** Example 6.47: Reduced prime implicant chart

PIs/Minterms	3	4	9	11	12	13
W → 2,3 (1)	x					
U → 4,6 (2)		x				
T → 4,12 (8)		x			x	
S → 3,11(8)	x			x		
R → 9,11 (2)			x	x		
Q → 9,13 (4)			x			x
P → 12,13( 1)				x	x	

In Table 6.24, row W is dominated by row S and row U is dominated by row T. So remove rows W and U. After removing rows W and U, we observe that column 11 is dominating column 3 and column 12 is dominating column 4. So remove columns 11 and 12 and draw the reduced PI chart shown Table 6.25. From that table we observe that T and S are essential prime implicants and together with Q they cover all the minterms. So the minimal expression is

$$F_{\min} = V + T + S + Q = \bar{A}\bar{C}\bar{D} + B\bar{C}\bar{D} + \bar{B}CD + ACD$$

This is another minimal expression. The minimal of these two is to be chosen. Since both of them require 16 gate inputs for realization, any one of them can be chosen.

**Table 6.25** Example 6.47: Reduced chart

PIs/Minterms	3	4	9	11	12	13
T → 4,12 (8)		x			x	
S → 3,11(8)	x			x		
R → 9,11 (2)			x	x		
Q → 9,13 (4)			x			x
P → 12,13( 1)				x	x	

**EXAMPLE 6.48** Simplify the following function using the branching method:

$$f(A, B, C, D, E) = \sum m(0, 4, 12, 16, 19, 24, 28, 29, 31)$$

**Solution**

Table 6.26 shows the procedure to obtain all the prime implicants. The PI chart is shown in Table 6.27.

**Table 6.26** Example 6.48

Column 1		Column 2
Index	Minterm	Pairs
Index 0	0 ✓	0, 4 (4) W
Index 1	4 ✓	0, 16 (16) V
	16 ✓	4, 12 (8) U
Index 2	12 ✓	16, 24 (8) T
	24 ✓	12, 28 (16) S
Index 3	19 X	24, 28 (4) R
	28 ✓	28, 29 (1) Q
Index 4	29 ✓	29, 31 (2) P
Index 5	31 ✓	

**Table 6.27** Example 6.48: PI chart

PIs/Minterms	0	4	12	16	19	24	28	29	31
*X → 19						×			
W → 0, 4 (4)	×	×							
V → 0, 16 (16)	×				×				
U → 4, 12 (8)		×	×						
T → 16, 24 (8)				×			×		
S → 12, 28 (16)			×					×	
R → 24, 28(4)						×	×		
Q → 28, 29 (1)							×	×	
*P → 29, 31 (2)								×	×

From the PI chart we observe that all the minterms are covered by PIs W, S, T, X, and P.

$$\therefore f_{\min} = W + S + T + X + P = \overline{AB}\overline{D}\overline{E} + BC\overline{D}\overline{E} + A\overline{C}\overline{D}\overline{E} + A\overline{B}\overline{C}DE + ABCE$$

From the PI chart observe that PIs X and P are essential prime implicants because  $m_{19}$  is covered only by X and  $m_{31}$  is covered only by P. So delete rows X and P and columns for  $m_{19}$ ,  $m_{29}$ , and  $m_{31}$  and write the reduced PI chart shown in Table 6.28 to apply the branching method.

**Table 6.28** Example 6.48: Reduced PI chart after removing row X

PIs/Minterms	0	4	12	16	24	28
W → 0, 4 (4)	×	×				
V → 0, 16 (16)	×			×		
U → 4, 12 (8)		×	×			
T → 16, 24 (8)				×	×	
S → 12, 28 (16)			×			×
R → 24, 28 (4)					×	×
Q → 28, 29 (1)						×

In the reduced PI chart of Table 6.28, there are no essential PIs, dominated rows or dominating columns. So the branching method can be applied. For that let us select column 0. It is covered by rows W and V. Let us first select row W and delete this row and the columns 0 and 4 covered by it and form Table 6.29.

**Table 6.29** Example 6.48: Reduced chart

PIs/Minterms	12	16	24	28
V → 0, 16 (16)		×		
U → 4, 12 (8)	×			
T → 16, 24 (8)		×	×	
S → 12, 28 (16)	×			×
R → 24, 28 (4)			×	×
Q → 28, 29 (1)				×

In Table 6.29 row V is dominated by row T, row U and row Q are dominated by row S. So rows V, U and Q can be deleted. The remaining table is shown in Table 6.30.

**Table 6.30**

PIs/Minterms	12	16	24	28
T → 16, 24 (8)		×	×	
S → 12, 28 (16)	×			×
R → 24, 28 (4)			×	×

Looking at Table 6.29 we can see that T and S are secondary essential prime implicants and together they cover all the minterms. So R is redundant. So the minimal expression is

$$f_{\min} = W + S + T + X + P = \bar{A}\bar{B}\bar{D}\bar{E} + BC\bar{D}\bar{E} + A\bar{C}\bar{D}\bar{E} + A\bar{B}\bar{C}DE + ABCE$$

The above procedure can be repeated by selecting row V and deleting this row and the columns covered by it, i.e. 0 and 16 and the second minimal expression can be obtained. The minimal of these two gives the real minimal expression.

## SHORT QUESTIONS AND ANSWERS

- 1.** What is a K-map? What are its advantages and disadvantages?  
 A. The K-map is a chart or a graph composed of an arrangement of adjacent cells, each representing a particular combination of variables in a sum or product term. Or simply we can say a K-map is a mapping technique used to minimize logical expression. The advantages of K-maps are: It is a systematic method of simplifying the Boolean expressions and it is very simple. The disadvantage is that it becomes tedious for problems involving five or more variables and also this method cannot be programmed.
- 2.** Why is minimization of switching functions required?  
 A. Minimization of switching functions is required in order to reduce the cost of the circuitry (using contact or gate networks) according to some prescribed cost criterion. Realization with minimal expressions also reduces complexity and improves reliability.
- 3.** What is a literal?  
 A. A literal is any variable appearing in the expression in complemented or un-complemented form.
- 4.** What is meant by SOP form?  
 A. SOP form is one in which each term of the logical expression is a logical product (AND) of some literals and all the terms are logically added (OR).
- 5.** What is meant by POS form?  
 A. POS form is one in which each term of the logical expression is a logical sum (OR) of some literals and all the terms are logically multiplied (AND).
- 6.** How many cells an  $n$  variable K-map can have?  
 A. An  $n$  variable K-map can have  $2^n$  cells or squares.
- 7.** What is a standard SOP form?  
 A. A standard or canonical or expanded SOP form is one in which each one of the product terms contains all the variables of the expression either in complemented or non-complemented form. Or a standard SOP form is one in which the logic expression consists of only minterms.
- 8.** What is a standard POS form?  
 A. A standard or canonical or expanded POS form is one in which each one of the sum terms contains all the variables of the expression either in complemented or non-complemented form. Or a standard POS form is one in which the logic expression consists of only maxterms.
- 9.** What is a minterm?  
 A. Each one of the product terms in the canonical SOP form is called a minterm, i.e. a minterm is a product term which contains all the variables of the function either in complemented or non-complemented form. Or a minterm is a term containing literals corresponding to all the variables in the ANDed form.
- 10.** What is a true minterm?  
 A. A true minterm is a minterm which is present in the canonical SOP expression.
- 11.** What is a false minterm?  
 A. A false minterm is a minterm which is not present in the canonical SOP expression.
- 12.** What is a maxterm?  
 A. Each one of the sum terms in the canonical POS form is called a maxterm, i.e. a maxterm is a sum term which contains all variables of the expression either in complemented or non-complemented form. Or a maxterm is a term containing literals corresponding to all the variables in ORed form.

- 13.** What is a true maxterm?
  - A.** A true maxterm is a maxterm which is present in canonical POS expression.
- 14.** What is a false maxterm?
  - A.** A false maxterm is a maxterm which is not present in the canonical POS expression.
- 15.** How do you convert a standard SOP form into a standard POS form?
  - A.** A standard SOP form can always be converted into a standard POS form by treating the missing minterms of the SOP form as the maxterms of the POS form.
- 16.** How do you convert a standard POS form into a standard SOP form?
  - A.** A standard POS form can always be converted into a standard SOP form by treating the missing maxterms of the POS form as the minterms of the SOP form.
- 17.** What do 1s and 0s on the SOP K-map represent?
  - A.** On the SOP K-map, the 1s represent the true minterms and the 0s represent the false minterms or the maxterms.
- 18.** What do 1s and 0s on the POS K-map represent?
  - A.** On the POS K-map, the 1s represent the true maxterms and the 0s represent the false maxterms or the minterms.
- 19.** When can two minterms or maxterms be combined?
  - A.** Two minterms or maxterms can be combined only when they are adjacent to each other.
- 20.** What is a subcube?
  - A.** Each square or rectangle made up of the bunch of adjacent minterms or maxterms is called a subcube.
- 21.** When are two squares on a K-map adjacent to each other?
  - A.** Two squares on a K-map are said to be adjacent to each other if they are physically adjacent to each other or can be made adjacent to each other by wrapping around.
- 22.** The binary number designations of the rows and columns of the K-map are in which code? Why?
  - A.** The binary number designations of the rows and columns of the K-map are in the Gray code. This is to ensure that two physically adjacent squares are really adjacent, i.e. their minterms or maxterms differ by only one variable.
- 23.** Write the codes of binary designations of the rows and columns of a four-variable K-map.
  - A.** The codes of binary designations of rows and columns of a four-variable K-map are 00,01,11,10 in order.
- 24.** What is adjacency ordering?
  - A.** Keeping the binary number designations of the rows and columns in Gray code is called adjacency ordering.
- 25.** What is the main criterion in the design of digital circuits?
  - A.** The main criterion in the design of digital circuits is the cost (indicated by the number of gate inputs) which must be kept to a minimum.
- 26.** What do you mean by real minimal expression?
  - A.** A real minimal expression is the minimal of the SOP and POS expressions.
- 27.** What is the shape of 2-squares, 4-squares, or 8-squares, etc.?
  - A.** The shape of 2-squares, 4-squares or 8-squares, etc. must be either geometric square or rectangle.
- 28.** How are a 2-square, 4-square, 8-square called?
  - A.** A 2-square is called a pair, a 4-square is called a quad and an 8-square is called an octet.

- 29.** How many variables do a 2-square, 4-square, 8-square, 16-square etc. eliminate?  
**A.** A 2-square eliminates one variable, a 4-square eliminates 2-variables, an 8-square eliminates 3 variables, a 16-square eliminates 4 variables and so on.
- 30.** What is a prime implicant in K-map?  
**A.** The bunch of 1s on the K-map which form a 2-square, 4-square, etc. is called a prime implicant or subcube. It is called true prime implicant.
- 31.** What is an essential prime implicant?  
**A.** The prime implicant which contains at least one 1 which cannot be covered by any other prime implicant is called an essential prime implicant.
- 32.** What is a redundant prime implicant?  
**A.** The prime implicant whose each 1 is covered by at least one EPI is called a redundant prime implicant (RPI).
- 33.** What is a selective prime implicant?  
**A.** A prime implicant which is neither an essential prime implicant nor a redundant prime implicant is called a selective prime implicant (SPI).
- 34.** What is a false prime implicant?  
**A.** The prime implicant made up of a bunch of 0s is called a false prime implicant.
- 35.** PIs of a function  $f(W, X, Y, Z) = \Sigma m(0, 1, 3, 7, 8, 9, 11, 15)$  are given by the following code groups. Which of these are EPIS?
- $$A = (0, 1, 8, 9) \quad B = (1, 3, 9, 11) \quad C = (3, 7, 11, 15)$$
- A.** A and C are essential prime implicants.
- 36.** In 5- and 6-variable K-maps when are squares in two blocks considered adjacent?  
**A.** In 5- and 6-variable K-maps, squares in two blocks are considered adjacent if when superimposing one block above or beside the other block, the squares coincide with one another.
- 37.** What are “don’t cares”?  
**A.** Combinations for which the value of the expression is not specified are called “don’t care” combinations.
- 38.** What are incompletely specified expressions?  
**A.** Incompletely specified expressions are those which are not specified for certain combinations.
- 39.** How is an SOP expression with “don’t cares” converted into a POS expression and vice versa?  
**A.** An SOP expression with “don’t cares” can be converted into POS form by keeping the “don’t cares” as they are and writing the missing minterms of the SOP form as the maxterms of the POS form and vice versa.
- 40.** What is two-level logic? What is its advantage?  
**A.** Two-level logic is one in which each input signal passes through two gates to reach the output. The SOP and POS forms of realization give two-level logic. It provides uniform propagation delay between the input and the output but may not yield the real minimal.
- 41.** What is hybrid logic? What is its advantage, disadvantage?  
**A.** Hybrid logic is one in which different input signals pass through different numbers of gates to reach the outputs. The advantage is it results in a circuit with the least number of gate inputs, so cost will be less. The disadvantage is that it does not produce uniform time delay and may suffer from the problem of logic race.
- 42.** Why is the name minterm?  
**A.** A minterm fills with 1s the minimum possible area of the K-map, short of filling no area at all. Hence the name minterm.

## **310 FUNDAMENTALS OF DIGITAL CIRCUITS**

**43.** Why is the name maxterm?

- A. A maxterm fills with 1s the maximum possible area of the K-map, short of filling the entire area. Hence the name maxterm.

**44.** What do you mean by looping?

- A. Combining of adjacent squares in a K-map containing 1s (or 0s) for the purpose of simplification of a SOP (or POS) expression is called looping.

**45.** What is variable mapping technique? What is its advantage?

- A. Variable mapping (variable entry mapping) technique is a technique used to minimize the given Boolean expressions which involve infrequently used variables. The advantage is it allows us to reduce a large mapping problem to one that uses just a small map.

**46.** What is the criterion for the minimization of multiple output switching functions?

- A. The criterion for the minimization of multiple output switching functions is:

- (i) Each minimized expression should have as many terms in common as possible with those in the other minimized expressions.
- (ii) Each minimized expression should have a minimum number of product (sum) terms and no product (sum) terms of its should be replaceable by a product (sum) term with fewer variables.

**47.** What is Quine–McClusky method?

- A. The Quine–McClusky method also known as the tabular method is a systematic method for minimizing functions of a large number of variables.

**48.** What is the advantage of the tabular method?

- A. The advantage of the tabular method is that it is fully algorithmic and hence programmable.

**49.** What do you mean by the index of a term?

- A. The index of a term indicates the total number of 1s present in that term. It is also called the weight of the term.

**50.** In the tabular method, why cannot two terms whose codes differ by a power of 2 but have the same index be combined?

- A. Two terms whose codes differ by a power of 2 but have the same index cannot be combined since they differ by more than one variable.

**51.** Why cannot a term with a smaller index but having a higher decimal value be combined with a term whose index is higher even though they differ by a power of 2?

- A. A term with a smaller index but having a higher decimal value cannot be combined with a term whose index is higher even though they may differ by a power 2, because they would differ in more than one variable.

**52.** What are prime implicants?

- A. The terms which cannot be combined further in the tabular method are called prime implicants. These terms may occur in the final expression.

**53.** What are essential prime implicants?

- A. Essential prime implicants are the prime implicants which will definitely occur in the final expression. Each essential prime implicant will cover at least one minterm which is not covered by any other prime implicant.

**54.** What is a prime implicant chart?

- A. The prime implicant chart is a pictorial representation of the relationships between the prime implicants and the minterms of the expression.

- 55.** How are “don’t care” minterms and maxterms used in the tabular method?
- A. “Don’t care” minterms and maxterms are used in the table only to obtain the set of prime implicants. They are not used in the prime implicant chart to obtain the essential prime implicants.
- 56.** When do you say that one row is dominating any other row?
- A. Any row in a prime implicant chart is said to dominate any other row, if the first row has a  $\times$  in every column in which the second row has a  $\times$ .
- 57.** When do you say that one column is dominating any other column?
- A. Any column in a prime implicant chart is said to dominate any other column if the first column has a  $\times$  in every row in which the second column has a  $\times$ .
- 58.** Which rows and columns can be removed while drawing the reduced prime implicant chart?
- A. All dominating columns and dominated rows can be removed while drawing the reduced prime implicant chart.
- 59.** When is the minimal expression obtained by the branching method?
- A. If the prime implicant chart has no essential prime implicants and dominated rows and dominating columns, the minimal expressions can be obtained by the branching method.
- 60.** How do you get all the possible minimal expressions by the tabular method?
- A. In the tabular method, to get all the possible minimal expressions, multiply the sums of the rows in the reduced prime implicant chart that take care of each minterm.
- 61.** Does elimination of dominating columns and dominated rows end the search for a minimal expression? If not, why not?
- A. Elimination of dominating columns and dominated rows does not end the search for a minimal expression. It is because the minimal expression can be obtained only from the final reduced prime implicant chart.

### REVIEW QUESTIONS

1. Write the procedure to expand an SOP expression into standard SOP form.
2. Write the procedure to expand a POS expression into standard POS form.
3. Write the procedure to simplify the Boolean expressions using K-maps.
4. Compare K-map and tabular methods of minimization.
5. Write the steps in the minimization using the tabular method.
6. Explain the branching method.
7. What are the limitations of K-maps.

### FILL IN THE BLANKS

1. The advantages of K-map are: It is a \_\_\_\_\_ method of simplifying the Boolean expressions and it is \_\_\_\_\_.
2. The disadvantages of K-map are: It becomes tedious for problems involving \_\_\_\_\_ variables and also this method cannot be \_\_\_\_\_.

### 312 FUNDAMENTALS OF DIGITAL CIRCUITS

3. An  $n$  variable K-map can have \_\_\_\_\_ cells or squares.
4. Each of the product terms in the standard SOP form is called a \_\_\_\_\_.
5. Each of the sum terms in the standard POS form is called a \_\_\_\_\_.
6. A product term which contains all the variables of the function either in complemented form or in uncomplemented form is called a \_\_\_\_\_.
7. A sum term which contains all the variables of the function either in complemented form or in uncomplemented form is called a \_\_\_\_\_.
8. The cost of a circuit is measured in terms of \_\_\_\_\_.
9. A \_\_\_\_\_ minterm is a minterm which is present in the canonical SOP expression.
10. A \_\_\_\_\_ minterm is a minterm which is not present in the canonical SOP expression.
11. A standard SOP form can always be converted into a standard POS form by treating the \_\_\_\_\_.
12. A standard POS form can always be converted into a standard SOP form by treating the \_\_\_\_\_.
13. Two squares on a K-map are said to be adjacent to each other if they are \_\_\_\_\_.
14. The binary number designations of the rows and columns of the K-map are in \_\_\_\_\_.
15. The main criterion in the design of digital circuits is \_\_\_\_\_.
16. Subcubes in the K-map form either \_\_\_\_\_ or \_\_\_\_\_.
17. A real minimal expression is the minimal of the \_\_\_\_\_ minimals.
18. A 2-square is called \_\_\_\_\_.
19. A 4-square is called \_\_\_\_\_.
20. An 8-square is called \_\_\_\_\_.
21. Two minterms or maxterms can be combined only when they are \_\_\_\_\_.
22. A 16-square eliminates \_\_\_\_\_ variables.
23. Combinations for which the value of an expression is not specified are called \_\_\_\_\_ combinations.
24. \_\_\_\_\_ logic is one in which each input signal passes through only two gates to reach the output.
25. \_\_\_\_\_ logic provides uniform propagation delay between input and output.
26. \_\_\_\_\_ logic is one in which different input signals pass through different numbers of gates to reach the output.
27. The \_\_\_\_\_ technique is used to reduce a large mapping problem to one that uses just a small map.
28. Combining of adjacent squares in a K-map containing 1s (or 0s) for the purpose of simplification of a SOP (or POS) expression is called \_\_\_\_\_.
29. The total number of 1s present in a term is called its \_\_\_\_\_ or \_\_\_\_\_.
30. Any variable appearing in the expression in complemented or uncomplemented form is called a \_\_\_\_\_.
31. The advantage of the tabular method is it is \_\_\_\_\_ and hence \_\_\_\_\_.
32. The terms which cannot be combined further in the tabular method are called \_\_\_\_\_.
33. The implicants which will definitely occur in the final expression are called \_\_\_\_\_.
34. The \_\_\_\_\_ is a pictorial representation of the relationship between the prime implicants and the minterms of the expression.
35. All dominating \_\_\_\_\_ and dominated \_\_\_\_\_ can be removed while drawing the reduced prime implicant chart.
36. The bunch of 1s on the K-map which form a 2-square, 4-square, etc. is called a \_\_\_\_\_.

37. The prime implicant which contains at least one 1 which cannot be covered by any other prime implicant is called an \_\_\_\_\_.
38. The PI whose each 1 is covered by at least one EPI is called a \_\_\_\_\_.
39. A PI which is neither an EPI nor a RPI is called a \_\_\_\_\_.
40. The PI made of a bunch of 0s is called a \_\_\_\_\_.

### OBJECTIVE TYPE QUESTIONS

1. An  $n$  variable K-map can have  
 (a)  $n^2$  cells      (b)  $2^n$  cells      (c)  $n^n$  cells      (d)  $n^{2n}$  cells
2. Each term in the standard SOP form is called a  
 (a) minterm      (b) maxterm      (c) don't care      (d) literal
3. Each term in the standard POS form is called a  
 (a) minterm      (b) maxterm      (c) don't care      (d) literal
4. The main criterion in the design of a digital circuit is reduction of  
 (a) cost      (b) size      (c) weight      (d) volume
5. The binary number designations of the rows and columns of the K-map are in  
 (a) binary code      (b) BCD code      (c) Gray code      (d) XS-3 code
6. An 8-square eliminates  
 (a) 2 variables      (b) 3 variables      (c) 4 variables      (d) 8 variables
7. An 8-square is called  
 (a) a pair      (b) a quad      (c) an octet      (d) a cube
8. Uniform propagation delay is provided using  
 (a) two level logic      (b) multilevel logic      (c) hybrid logic      (d) high level logic
9. Any variable appearing in the final expression is called a  
 (a) literal      (b) a real variable      (c) final variable      (d) variable
10. The total number of 1s present in a term is called the  
 (a) index      (b) weight      (c) logic level      (d) term number
11. Combining of adjacent squares on a K-map containing 1s (or 0s) for the purpose of simplification of a SOP (or POS) expression is called  
 (a) looping      (b) squaring      (c) charting      (d) forming
12. The terms which cannot be combined further in the tabular method are called  
 (a) implicants      (b) prime implicants  
 (c) essential prime implicants      (d) selective prime implicants
13. The implicants which will definitely occur in the final expression are called  
 (a) prime implicants      (b) essential prime implicants  
 (c) selective prime implicants      (d) redundant prime implicants
14. The code used for labeling the cells of a K-map is  
 (a) 8-4-2-1 binary      (b) hexadecimal      (c) gray      (d) octal



## PROBLEMS

- 6.1** Convert the following to minterms:

  - (a)  $A + \bar{B}\bar{C}$
  - (b)  $\bar{A} + B + CA$
  - (c)  $ABC + AB + DC + \bar{D}$
  - (d)  $ABCDE + AB\bar{E} + ACD$

**6.2** Convert the following to maxterms:

  - (a)  $A(B + \bar{C})$
  - (b)  $(A + \bar{B})(\bar{A} + D)$
  - (c)  $(A + B + \bar{D})(\bar{A} + C + D)(\bar{A} + \bar{D})$
  - (d)  $A(\bar{A} + B)(\bar{C})$

**6.3** How many gate inputs are required to realize the following expressions:

  - (a)  $AB\bar{C} + \bar{A}BC + ABCD + ABD$
  - (b)  $WX\bar{Y} + WXZ + VUX + XY\bar{Z}W$
  - (c)  $A + BC + \bar{D}EF$
  - (d)  $(A + C)(A + \bar{B} + C)(A + C + \bar{D})(A + B + C + \bar{D})$
  - (e)  $A(B + \bar{D})(A + C + E)(B + \bar{C} + \bar{D} + E)$
  - (f)  $(A + B)(\bar{C} + D)(E + F + \bar{G})\bar{D}$

**6.4** Reduce the following expressions using K-map.

  - (a)  $AB + A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{B}\bar{C}$
  - (b)  $AB\bar{C} + AB + C + B\bar{C} + D\bar{B}$
  - (c)  $AB + A\bar{C} + C + AD + A\bar{B}\bar{C} + ABC$
  - (d)  $A\bar{B}C + B + B\bar{D} + ABD + \bar{A}C$

**6.5** Reduce the following expressions using K-map.

  - (a)  $(A + B)(A + \bar{B} + C)(A + \bar{C})$
  - (b)  $A(B + \bar{C})(A + \bar{B})(B + C + \bar{D})$
  - (c)  $(\bar{A} + B)(A + B + \bar{D})(B + \bar{C})(B + C + D)$

**6.6** Obtain the minimal SOP expression for  $\Sigma m(2, 3, 5, 7, 9, 11, 12, 13, 14, 15)$  and implement it in NAND logic.

**6.7** Obtain the minimal POS expression for  $\Pi M(0, 1, 2, 4, 5, 6, 9, 11, 12, 13, 14, 15)$  and implement it in NOR logic.

**6.8** Reduce  $\Pi M(1, 2, 3, 5, 6, 7, 8, 9, 12, 13)$  and implement it in universal logic.

## 316 FUNDAMENTALS OF DIGITAL CIRCUITS

**6.9** Reduce the following expressions using K-map and implement them in universal logic.

- (a)  $\Sigma m(5, 6, 7, 9, 10, 11, 13, 14, 15)$
- (b)  $\Sigma m(0, 1, 2, 3, 4, 6, 8, 9, 10, 11)$
- (c)  $\Pi M(1, 4, 5, 11, 12, 14) \cdot d(6, 7, 15)$
- (d)  $\Pi M(3, 6, 8, 11, 13, 14) \cdot d(1, 5, 7, 10)$
- (e)  $\Sigma m(0, 1, 4, 5, 6, 7, 9, 11, 15) + d(10, 14)$
- (f)  $\Sigma m(9, 10, 12) + d(3, 5, 6, 7, 11, 13, 14, 15)$

**6.10** Simplify the following logic expressions and realize them using universal gates.

- (a)  $\Sigma m(6, 9, 13, 18, 19, 25, 27, 29, 31)$
- (b)  $\Sigma m(0, 2, 3, 10, 12, 16, 17, 18, 21, 26, 27) + d(11, 13, 19, 20)$
- (c)  $\Sigma m(0, 1, 2, 4, 5, 7, 8, 9, 10, 14, 15, 17, 19, 20, 28, 29, 34, 36, 40, 41, 42, 43)$
- (d)  $\Sigma m(4, 6, 8, 9, 10, 12, 13, 18, 19, 25, 26, 29, 33, 35, 36, 41, 42, 48, 49, 50, 56, 57) + d(0, 1, 11, 15, 30, 38, 40)$

**6.11** Obtain the minimal expression using the tabular method and implement it in universal logic.

- |  |   |
|--|---|
| (a) $\Sigma m(0, 1, 3, 4, 5, 7, 10, 13, 14, 15)$ | (b) $\Sigma m(0, 2, 3, 6, 7, 8, 10, 11, 12, 15)$        |
| (c) $\Sigma m(0, 1, 3, 4, 5, 6, 7, 13, 15)$      | (d) $\Pi M(6, 7, 8, 9) \cdot d(10, 11, 12, 13, 14, 15)$ |
| (e) $\Pi M(1, 5, 6, 7, 11, 12, 13, 15)$          | (f) $\Sigma m(1, 5, 6, 12, 13, 14) + d(2, 4)$           |

**6.12** Implement the function F with the following four two-level forms:

- |              |             |
|--------------|-------------|
| (a) NAND-AND | (b) AND-NOR |
| (c) OR-AND   | (d) NOR-OR  |

$$F(A, B, C, D) = \Sigma m(0, 1, 4, 6, 8, 9, 10, 12)$$

**6.13** Implement the function F with the following four two-level forms:

- |              |             |
|--------------|-------------|
| (a) NAND-AND | (b) AND-NOR |
| (c) OR-AND   | d) NOR-OR   |

$$F(A, B, C, D) = \Sigma m(0, 2, 3, 4, 7, 9, 15) + d(6, 8, 11)$$

**6.14** Minimize and implement the following multiple output functions in SOP form using K-maps.

- (a)  $f_1 = \Sigma m(1, 2, 5, 6, 8, 9, 10)$   
 $f_2 = \Sigma m(2, 4, 6, 8, 10, 12, 15)$
- (b)  $f_1 = \Sigma m(0, 1, 4, 6, 8, 9, 11) + d(2, 7, 13)$   
 $f_2 = \Sigma m(2, 4, 5, 7, 9, 12) + d(0, 1, 6)$
- (c)  $f_1 = \Sigma m(0, 1, 2, 4, 6, 7, 10, 14, 15)$   
 $f_2 = \Sigma m(3, 4, 5, 9, 10, 11, 14)$
- (d)  $f_1 = \Sigma m(2, 3, 7, 10, 11, 14) + d(1, 5, 15)$   
 $f_2 = \Sigma m(0, 1, 4, 7, 13, 14) + d(5, 8, 15)$

**6.15** Reduce the following expressions using a three-variable map.

- (a)  $A \bar{B}C + \bar{A}BC\bar{D} + ABC\bar{D} + ABC$
- (b)  $AB\bar{E}\bar{C} + ABC\bar{D} + ABC\bar{E} + A\bar{B}C\bar{D} + \bar{A}BC$
- (c)  $\bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}CD + A\bar{B}C\bar{D} + A\bar{B}CD + \bar{A}BC\bar{D} + AB\bar{C}\bar{D} + AB\bar{C}D$
- (d)  $m_1 + Dm_2 + m_3 + \bar{D}m_5 + m_7 + d(m_0 + Dm_6)$
- (e)  $Em_2 + m_3 + m_4 + Dm_5 + \bar{D}m_7 + d(m_0 + Em_1 + \bar{E}m_6)$
- (f)  $Dm_2 + Dm_5 + m_6 + d(m_1 + \bar{D}m_7)$

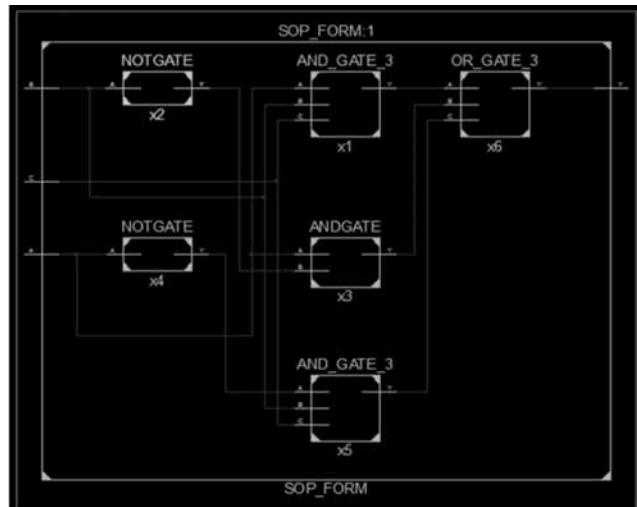
**6.16** Reduce the following expressions using a four-variable map.

- (a)  $Q = m_0 + m_2 + Fm_6 + m_7 + Em_8 + Em_{10} + m_{12} + m_{14} + Fm_{15}$
- (b)  $R = m_0 + m_2 + Fm_4 + Gm_7 + \bar{F}m_9 + m_{10} + \bar{E}m_{11} + m_{12} + Em_{14} + m_{15}$
- (c)  $S = m_1 + Fm_2 + Gm_4 + m_6 + \bar{F}m_7 + Hm_9 + (F + G)m_{10} + d[(F + \bar{G})m_0 + Gm_5 + m_7 + m_{14} + \bar{F}m_{15}]$
- (d)  $T = Em_0 + Hm_1 + Fm_2 + (\bar{E} + G)m_6 + m_8 + d(m_{10} + \bar{E}m_{14} + Gm_7)$
- (e)  $U = Em_3 + m_5 + m_7 + Fm_{10} + m_{12} + m_{14}$
- (f)  $V = m_0 + (E + G)m_2 + \bar{E}m_5 + Gm_{10} + Fm_{13} + m_{14} + m_{15} + d(m_1 + Em_4 + \bar{F}m_8 + \bar{F}m_9).$

## VHDL PROGRAMS

### 1. VHDL PROGRAM IN STRUCTURAL MODELING FOR A GIVEN SOP FORM

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



```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity SOP_FORM is
Port (A, B,C: in STD_LOGIC;
Y: out STD_LOGIC);
end SOP_FORM;
architecture Behavioral of SOP_FORM is
component AND_GATE_3 is
Port ( A,B,C : in STD_LOGIC;
Y : out STD_LOGIC);
end component;
component ANDGATE is
Port ( A,B : in STD_LOGIC;
Y : out STD_LOGIC);
end component;
component NOTGATE is
Port ( A : in STD_LOGIC;
Y : out STD_LOGIC);
end component;
component OR_GATE_3 is
Port ( A,B,C : in STD_LOGIC;
Y : out STD_LOGIC);
end component;
signal n1,n2,n3,n4,n5:STD_LOGIC;

```

```

begin
x1:AND_GATE_3 port map(A,B,C,n1);
x2:NOTGATE port map(B,n2);
x3:ANDGATE port map(A,n2,n3);
x4:NOTGATE port map(A,n4);
x5:AND_GATE_3 port map(n4,B,C,n5);
x6:OR_GATE_3 port map(n1,n3,n5,Y);
end Behavioral;

```

### **COMPONENT INSTANTIATION VHDL PROGRAM FOR 3-INPUT AND GATE**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity AND_GATE_3 is
    Port ( A,B,C : in STD_LOGIC; Y : out STD_LOGIC) ;
end AND_GATE_3;
architecture Behavioral of AND_GATE_3 is
signal n1:STD_LOGIC;
begin
n1 <= A AND B;
Y <= n1 AND C;
end Behavioral;

```

### **VHDL PROGRAM FOR NOT GATE**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity NOTGATE is
Port ( A : in STD_LOGIC; Y : out STD_LOGIC) ;
end NOTGATE;
architecture Behavioral of NOTGATE is
begin
Y <= (NOT (A));
end Behavioral;

```

### **VHDL PROGRAM FOR 2-INPUT AND GATE**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ANDGATE is
Port (A, B: in STD_LOGIC;
Y: out STD_LOGIC);
end ANDGATE;
architecture Behavioral of ANDGATE is
begin
Y <= A AND B;
end Behavioral;

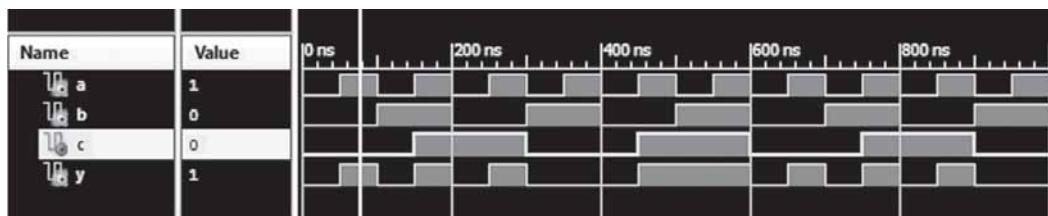
```

**VHDL PROGRAM FOR 3-INPUT OR GATE**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity OR_GATE_3 is
    Port ( A,B,C : in STD_LOGIC; Y : out STD_LOGIC);
end OR_GATE_3;
architecture Behavioral of OR_GATE_3 is
signal n1:STD_LOGIC;
begin
n1 <= A OR B;
Y <= n1 OR C;
end Behavioral;

```

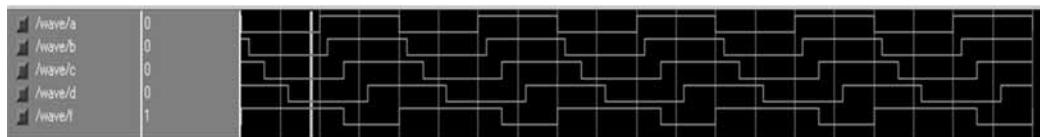
**SIMULATION OUTPUT:****2. VHDL PROGRAM IN DATA FLOW MODELING FOR A GIVEN POS FORM**

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

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity POS is
    Port ( A,B,C,D : in STD_LOGIC;
            F : out STD_LOGIC);
end POS;
architecture Behavioral of POS is
signal n1,n2,n3,n4: STD_LOGIC;
begin
n1 <= (not A);
n2 <= (not B);
n3 <= (not C);
n4 <= (not D);
F <= ((A or n2 or D)and(n1 or n3 or n4) and (n1 or n2 or n3));
end Behavioral;

```

**SIMULATION OUTPUT:****VERILOG PROGRAMS****1. VERILOG PROGRAM IN STRUCTURAL MODELING FOR A GIVEN SOP FORM**

$$Y = ABC + AB + AC$$

```

module and_3_gate(
    input a,b,c,
    output y
);
assign y = a & b & c;
endmodule

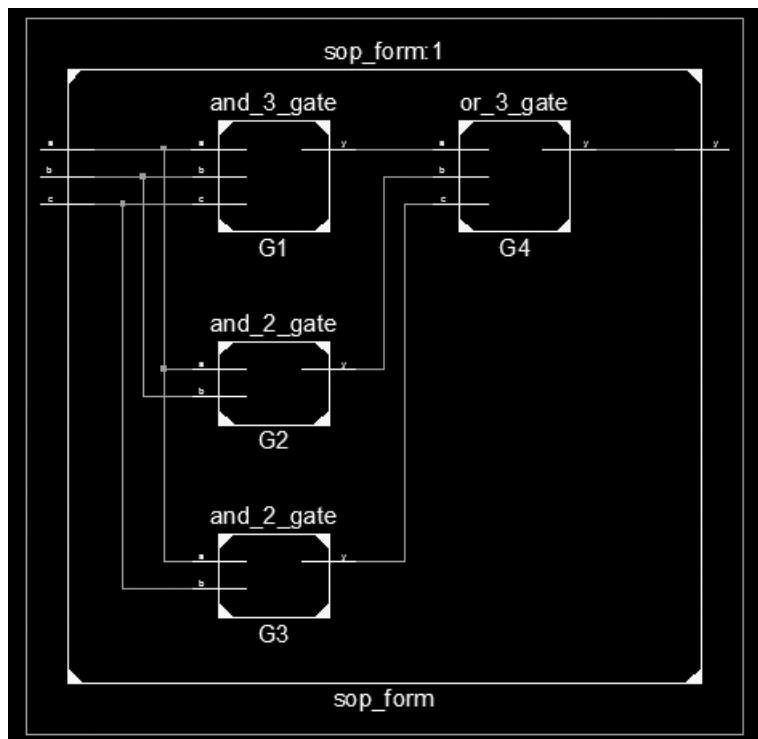
module and_2_gate(
    input a,b,
    output y
);
assign y = a & b;
endmodule

module or_3_gate(
    input a,b,c,
    output y
);
assign y = a | b | c;
endmodule

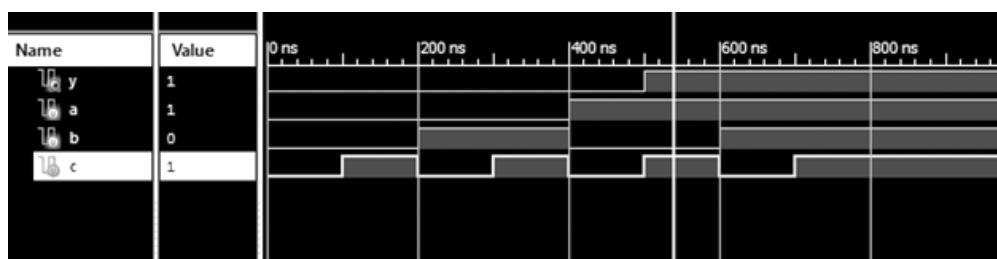
module sop_form (a,b,c,y);
input a,b,c;
output y;
wire N1,N2,N3;
and_3_gate G1(a,b,c,N1);
and_2_gate G2(a,b,N2);
and_2_gate G3(a,c,N3);
or_3_gate G4(N1,N2,N3,y);
endmodule

```

## RTL SCHEMATIC



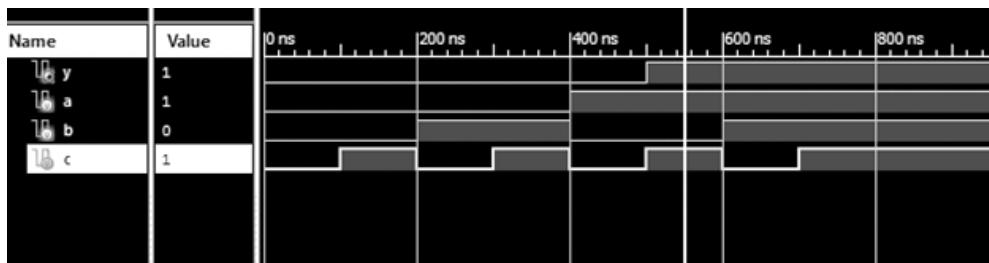
## SIMULATION OUTPUT:



## 2. VERILOG PROGRAM IN DATA FLOW MODELING FOR A GIVEN SOP FORM

$$Y = ABC + AB + AC$$

```
module sop_form (a,b,c,y);
input a,b,c;
output y;
assign y = ((a&b&c) | (a&b) | (a&c));
endmodule
```

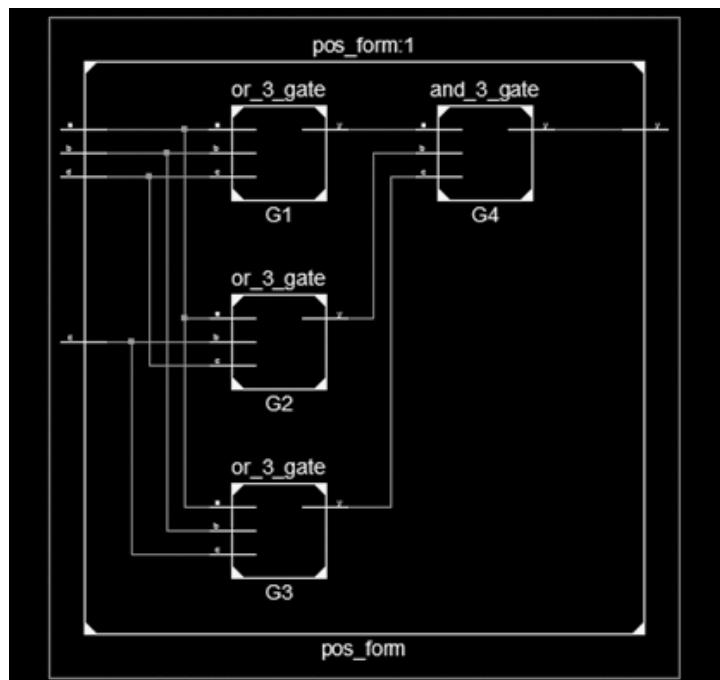
**SIMULATION OUTPUT:****3. VERILOG PROGRAM IN STRUCTURAL MODELING FOR A GIVEN POS FORM**

```

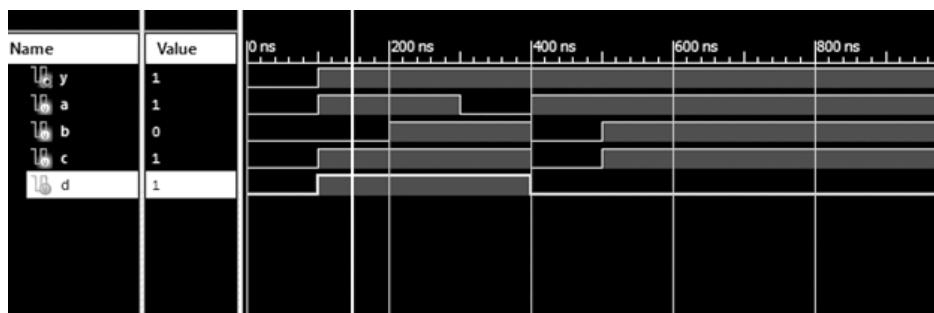
Y = (A + B + D) (A + C + D) (A + B + C)
module and_3_gate(
    input a,b,c,
    output y
);
assign y = a & b & c;
endmodule
module or_3_gate(
    input a,b,c,
    output y
);
assign y = a | b | c;
endmodule
module pos_form (a,b,c,d,y);
    input a,b,c,d;
    output y;
    wire N1,N2,N3;
    or_3_gate G1(a,b,d,N1);
    or_3_gate G2(a,c,d,N2);
    or_3_gate G3(a,b,c,N3);
    and_3_gate G4(N1,N2,N3,y);
endmodule

```

## RTL SCHEMATIC



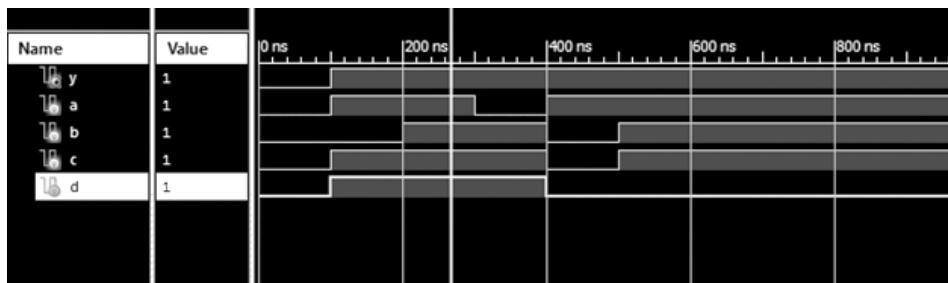
## SIMULATION OUTPUT:



## 4. VERILOG PROGRAM IN DATA FLOW MODELING FOR A GIVEN POS FORM

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

```
module pos_form (a,b,c,d,y);
input a,b,c,d;
output y;
assign y = ((a|b|d)&(a|c|d)&(a|b|c));
endmodule
```

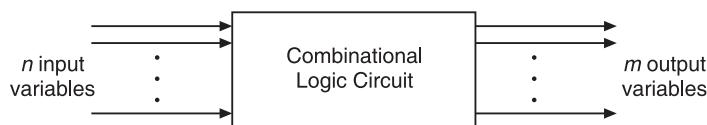
**SIMULATION OUTPUT:**

# 7

## COMBINATIONAL LOGIC DESIGN

### 7.1 INTRODUCTION

Logic circuits for digital systems may be combinational or sequential. The output of a combinational circuit depends on its present inputs only. Combinational circuits perform a specific information processing operation fully specified logically by a set of Boolean functions. A combinational circuit consists of input variables, logic gates, and output variables. The logic gates accept signals from the inputs and generate signals to the outputs. This process transforms binary information from the given input data to the required output data. Obviously both input and output data are represented by signals, i.e. they exist in two possible values, one representing logic-1 and the other logic-0. The block diagram of a combinational circuit is shown in Figure 7.1. The  $n$  input binary variables come from an external source and the  $m$  output variables go to an external destination. For  $n$  input variables, there are  $2^n$  possible combinations of binary input values. For each possible input combination, there is one and only one possible output combination. A combinational circuit can be described by  $m$  Boolean functions one for each output variable. Each output function is expressed in terms of the  $n$  input variables. Usually the inputs come from flip-flops and outputs go to flip-flops. So both the variable and its complement are assumed to be available.



**Figure 7.1** The block diagram of a combinational circuit.

## 7.2 DESIGN PROCEDURE

The design of combinational circuits starts from the verbal description of the problem and ends in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be easily obtained. The procedure involves the following steps:

1. The problem is stated.
2. The number of available input variables and required output variables is determined.
3. The input and output variables are assigned letter symbols.
4. The truth table that defines the required relationship between inputs and outputs is derived.
5. The simplified Boolean function for each output is obtained.
6. The logic diagram is drawn.

## 7.3 ADDERS

Digital computers perform a variety of information processing tasks. Among the basic tasks encountered are the various arithmetic operations. The most basic arithmetic operation is the addition of two binary digits. This simple addition consists of 4 possible operations, namely,

$$0 + 0 = 0, \quad 0 + 1 = 1, \quad 1 + 0 = 1, \quad \text{and} \quad 1 + 1 = 10.$$

The first three operations produce a sum whose length is one digit, but when both augend and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a carry. When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher-order pair of significant bits. A combinational circuit that performs the addition of two bits is called a *half-adder*. One that performs the addition of three bits (two significant bits and previous carry) is called a *full-adder*. The name of the former stems from the fact that two half-adders can be employed to implement a full-adder.

### 7.3.1 The Half-Adder

A half-adder is a combinational circuit with two binary inputs (augend and addend bits) and two binary outputs (sum and carry bits). It adds the two inputs (A and B) and produces the sum (S) and the carry (C) bits. It is an arithmetic circuit used to perform the arithmetic operation of addition of two single bit words. The truth table and block diagram of a half-adder are shown in Figure 7.2.

Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(a) Truth table
(b) Block diagram

Figure 7.2 Half-adder.

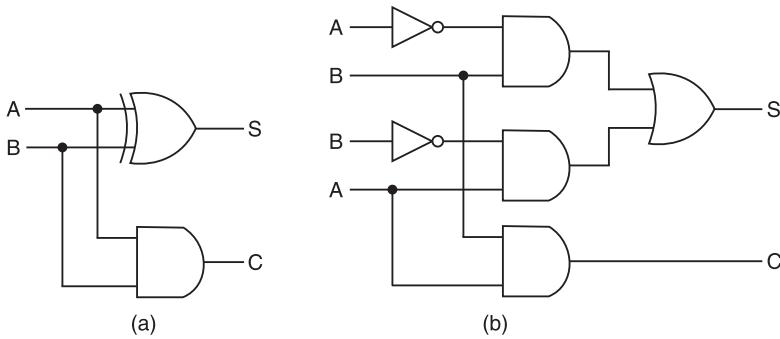
The sum (S) bit and the carry (C) bit, according to the rules of binary addition, are given by:  
The sum (S) is the X-OR of A and B (It represents the LSB of the sum). Therefore,

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

The carry (C) is the AND of A and B (It is 0 unless both the inputs are 1). Therefore,

$$C = AB$$

A half-adder can, therefore, be realized by using one X-OR gate and one AND gate as shown in Figure 7.3a. Realization using AOI logic is shown in Figure 7.3b.

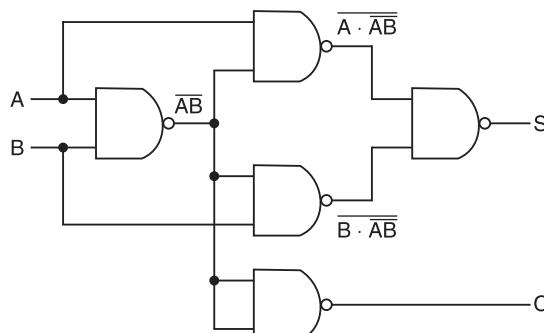


**Figure 7.3** Logic diagrams of half-adder.

A half-adder can also be realized in universal logic by using either only NAND gates or only NOR gates as shown in Figures 7.4 and 7.5 respectively.

#### NAND logic

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

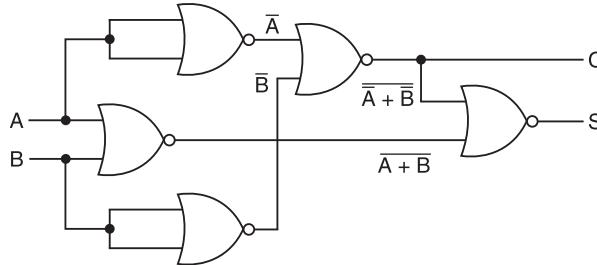


**Figure 7.4** Logic diagram of a half-adder using only 2-input NAND gates.

#### NOR logic

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

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



**Figure 7.5** Logic diagram of a half-adder using only 2-input NOR gates.

### 7.3.2 The Full-Adder

A full-adder is a combinational circuit that adds two bits and a carry and outputs a sum bit and a carry bit. When we want to add two binary numbers, each having two or more bits, the LSBs can be added by using a half-adder. The carry resulted from the addition of the LSBs is carried over to the next significant column and added to the two bits in that column. So, in the second and higher columns, the two data bits of that column and the carry bit generated from the addition in the previous column need to be added.

The full-adder adds the bits A and B and the carry from the previous column called the carry-in  $C_{in}$  and outputs the sum bit S and the carry bit called the carry-out  $C_{out}$ . The variable S gives the value of the least significant bit of the sum. The variable  $C_{out}$  gives the output carry. The block diagram and the truth table of a full-adder are shown in Figure 7.6. The eight rows under the input variables designate all possible combinations of 1s and 0s that these variables may have. The 1s and 0s for the output variables are determined from the arithmetic sum of the input bits. When all the bits are 0s, the output is 0. The S output is equal to 1 when only 1 input is equal to 1 or when all the inputs are equal to 1. The  $C_{out}$  has a carry of 1 if two or three inputs are equal to 1.

Inputs			Sum	Carry
A	B	$C_{in}$	S	$C_{out}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) Truth table
(b) Block diagram

**Figure 7.6** Full-adder.

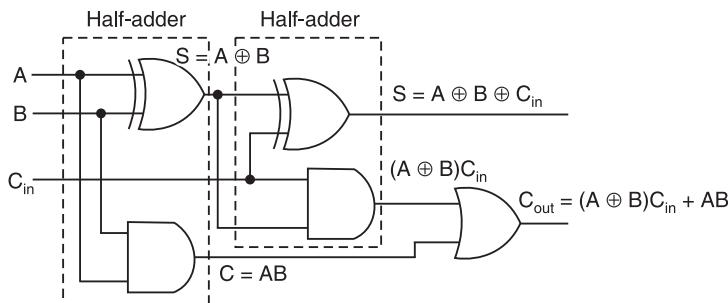
From the truth table, a circuit that will produce the correct sum and carry bits in response to every possible combination of A, B, and  $C_{in}$  is described by

$$\begin{aligned} S &= \overline{ABC}_{in} + \overline{AB}\overline{C}_{in} + \overline{A}\overline{BC}_{in} + ABC_{in} \\ &= (\overline{A}\overline{B} + \overline{AB})\overline{C}_{in} + (AB + \overline{AB})C_{in} = (A \oplus B)\overline{C}_{in} + (A \oplus B)C_{in} = A \oplus B \oplus C_{in} \end{aligned}$$

and

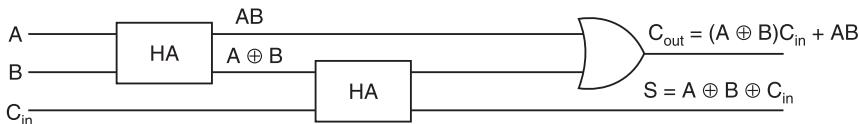
$$C_{out} = \overline{ABC}_{in} + \overline{AB}\overline{C}_{in} + AB\overline{C}_{in} + ABC_{in} = AB + (A \oplus B)C_{in} = AB + AC_{in} + BC_{in}$$

The sum term of the full-adder is the X-OR of A, B, and  $C_{in}$ , i.e. the sum bit is the modulo sum of the data bits in that column and the carry from the previous column. The logic diagram of the full-adder using two X-OR gates and two AND gates (i.e. two half-adders) and one OR gate is shown in Figure 7.7.



**Figure 7.7** Logic diagram of a full-adder using two half-adders.

The block diagram of a full-adder using two half-adders is shown in Figure 7.8.



**Figure 7.8** Block diagram of a full-adder using two half-adders.

Even though a full-adder can be constructed using two half-adders as shown in Figure 7.7, the disadvantage is that the bits must propagate through several gates in succession, which makes the total propagation delay greater than that of the full-adder circuit using AOI logic shown in Figure 7.9.

The full-adder can also be realized using universal logic, i.e. either only NAND gates or only NOR gates as shown in Figures 7.10 and 7.11 respectively.

### NAND logic

We know that

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

Then

$$S = A \oplus B \oplus C_{in} = \overline{(A \oplus B) \cdot \overline{(A \oplus B)C_{in}} \cdot \overline{C_{in}} \cdot \overline{(A \oplus B)C_{in}}}$$

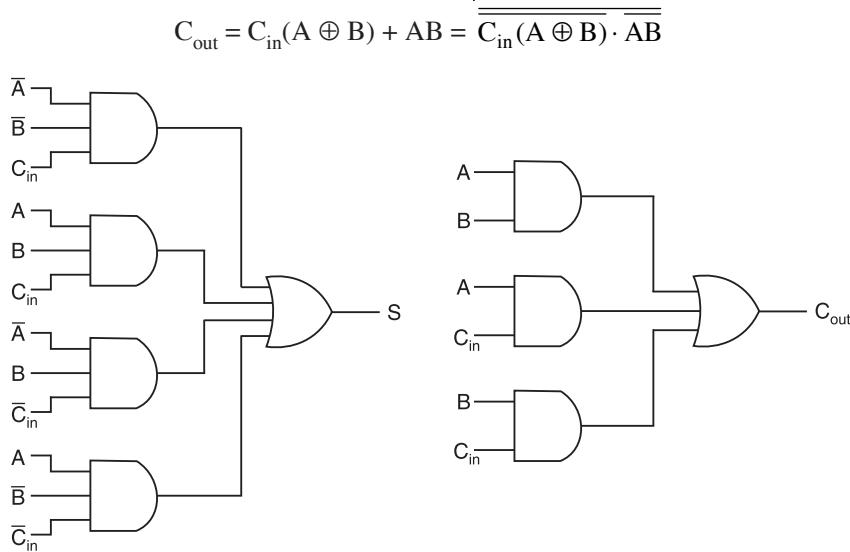


Figure 7.9 Sum and carry bits of a full-adder using AOI logic.

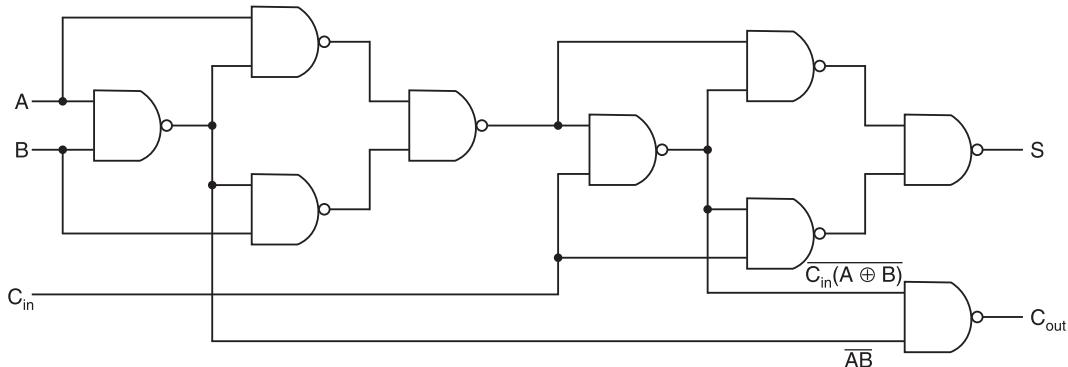


Figure 7.10 Logic diagram of a full-adder using only 2-input NAND gates.

### NOR logic

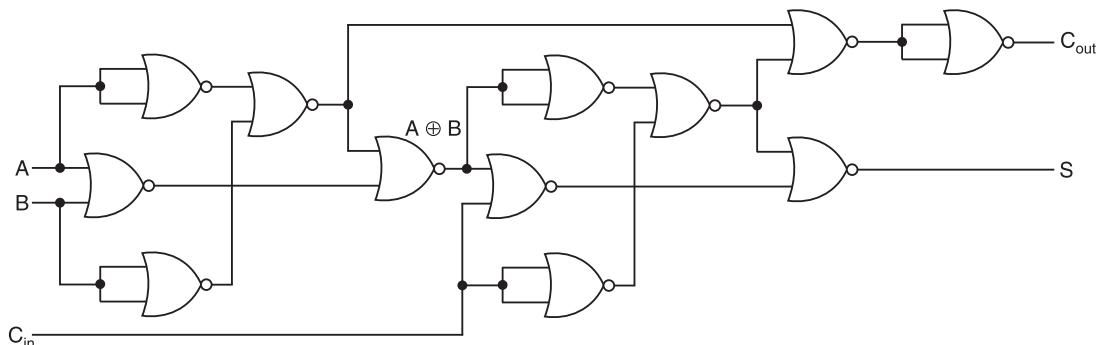
We know that

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

Then

$$S = A \oplus B \oplus C_{\text{in}} = \overline{\overline{(A \oplus B)} + \overline{C_{\text{in}}}} + \overline{\overline{(A \oplus B)} + \overline{\overline{C_{\text{in}}}}}$$

$$C_{\text{out}} = AB + C_{\text{in}}(A \oplus B) = \overline{\overline{A} + \overline{B}} + \overline{\overline{C_{\text{in}}} + \overline{A \oplus B}}$$



**Figure 7.11** Logic diagram of a full-adder using only 2-input NOR gates.

## 7.4 SUBTRACTORS

The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend. By this method, the subtraction operation becomes an addition operation and instead of having a separate circuit for subtraction, the adder itself can be used to perform subtraction. This results in reduction of hardware. In subtraction, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit. If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position. The fact that a 1 has been borrowed must be conveyed to the next higher pair of bits by means of a signal coming out (output) of a given stage and going into (input) the next higher stage. Just as there are half- and full-adders, there are half- and full-subtractors.

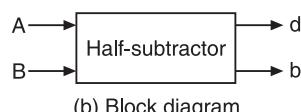
### 7.4.1 The Half-Subtractor

A half-subtractor is a combinational circuit that subtracts one bit from the other and produces the difference. It also has an output to specify if a 1 has been borrowed. It is used to subtract the LSB of the subtrahend from the LSB of the minuend when one binary number is subtracted from the other.

A half-subtractor shown in Figure 7.12 is a combinational circuit with two inputs A and B and two outputs d and b. d indicates the difference and b is the output signal generated that informs the next stage that a 1 has been borrowed. We know that, when a bit B is subtracted from another bit A, a difference bit (d) and a borrow bit (b) result according to the rules given as follows.

Inputs		Outputs	
A	B	d	b
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	1

(a) Truth table



(b) Block diagram

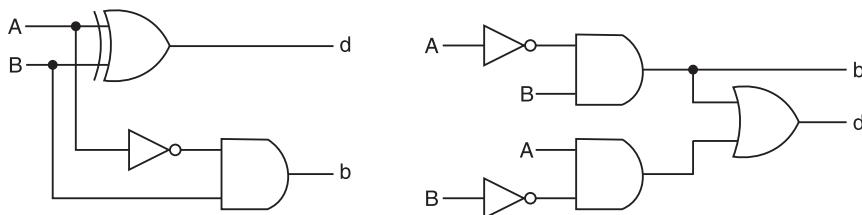
**Figure 7.12** Half-subtractor.

The output borrow  $b$  is a 0 as long as  $A \geq B$ . It is a 1 for  $A = 0$  and  $B = 1$ . The  $d$  output is the result of the arithmetic operation  $2b + A - B$ .

A circuit that produces the correct difference and borrow bits in response to every possible combination of the two 1-bit numbers is, therefore, described by

$$d = A\bar{B} + \bar{A}B = A \oplus B \text{ and } b = \bar{A}\bar{B}$$

That is, the difference bit is obtained by X-ORing the two inputs, and the borrow bit is obtained by ANDing the complement of the minuend with the subtrahend. Figure 7.13 shows two logic diagrams of a half-subtractor—one using an X-OR gate together with one each NOT gate and AND gate and the other using the AOI gates. Note that the logic for  $d$  is exactly the same as the logic for output  $S$  in the half-adder.



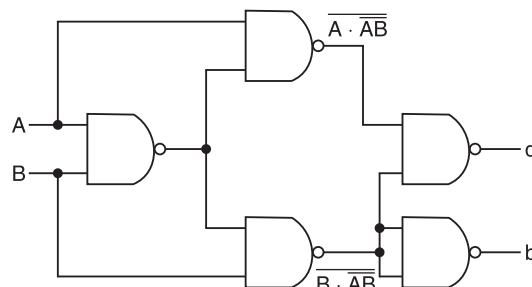
**Figure 7.13** Logic diagrams of a half-subtractor.

A half-subtractor can also be realized using universal logic—either using only NAND gates or using only NOR gates—as shown in Figures 7.14 and 7.15 respectively.

#### NAND logic

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

$$b = \overline{A}B = B(\overline{A} + \overline{B}) = B(\overline{A} \cdot \overline{B}) = \overline{B} \cdot \overline{A}B$$



**Figure 7.14** Logic diagram of a half-subtractor using only 2-input NAND gates.

#### NOR logic

$$\begin{aligned} d &= A \oplus B = \overline{A}B + \overline{A}B = \overline{A}B + \overline{B}B + \overline{A}B + A\overline{A} \\ &= \overline{B}(A + B) + \overline{A}(A + B) = \overline{B} + \overline{A} + B + A + A + B \\ d &= \overline{A}B = \overline{A}(A + B) = \overline{A}(A + B) = A + (\overline{A} + B) \end{aligned}$$

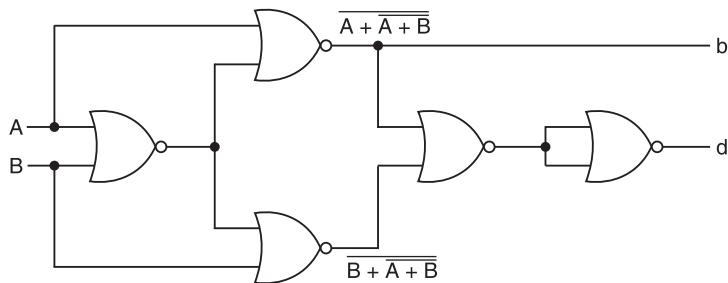


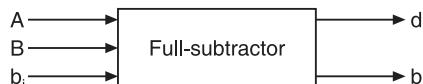
Figure 7.15 Logic diagram of a half-subtractor using only 2-input NOR gates.

#### 7.4.2 The Full-Subtractor

The half-subtractor can be used only for LSB subtraction. If there is a borrow during the subtraction of the LSBs, it affects the subtraction in the next higher column; the subtrahend bit is subtracted from the minuend bit, considering the borrow from that column used for the subtraction in the preceding column. Such a subtraction is performed by a full-subtractor. It subtracts one bit (B) from another bit (A), when already there is a borrow  $b_i$  from this column for the subtraction in the preceding column, and outputs the difference bit (d) and the borrow bit (b) required from the next column. So a full-subtractor is a combinational circuit with three inputs (A, B,  $b_i$ ) and two outputs d and b. The two outputs present the difference and output borrow. The 1s and 0s for the output variables are determined from the subtraction of  $A - B - b_i$ . The truth table and the block diagram of a full-subtractor are shown in Figure 7.16.

Inputs			Difference	Borrow
A	B	$b_i$	d	b
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

(a) Truth table



(b) Block diagram

Figure 7.16 Full-subtractor.

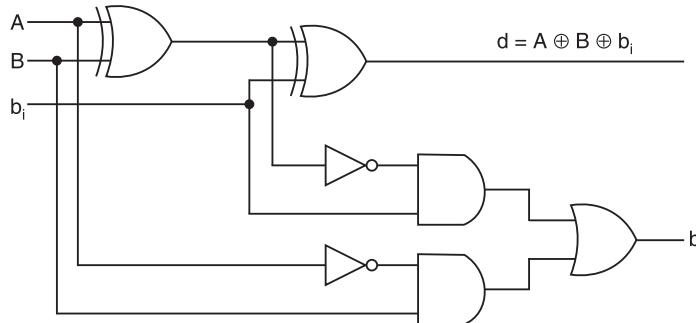
From the truth table, a circuit that will produce the correct difference and borrow bits in response to every possible combination of A, B, and  $b_i$  is described by

$$\begin{aligned} d &= \overline{A}\overline{B}b_i + \overline{A}\overline{B}\overline{b}_i + \overline{A}B\overline{b}_i + ABb_i \\ &= b_i(AB + \overline{A}\overline{B}) + \overline{b}_i(\overline{A}B + \overline{A}\overline{B}) \\ &= b_i(A \oplus B) + \overline{b}_i(A \oplus B) = A \oplus B \oplus b_i \end{aligned}$$

and

$$\begin{aligned} b &= \overline{A}\overline{B}b_i + \overline{A}\overline{B}\overline{b}_i + \overline{A}Bb_i + AB\overline{b}_i = \overline{A}B(b_i + \overline{b}_i) + (AB + \overline{A}\overline{B})b_i \\ &= \overline{A}B + (A \oplus B)b_i \end{aligned}$$

A full-subtractor can, therefore, be realized using X-OR gates and AOI gates as shown in Figure 7.17.

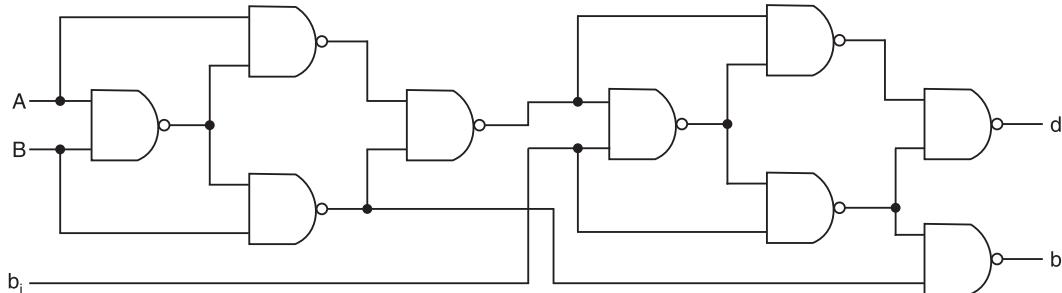


**Figure 7.17** Logic diagram of a full-subtractor.

The full subtractor can also be realized in universal logic using either only NAND gates or only NOR gates as shown in Figures 7.18 and 7.19 respectively.

#### NAND logic

$$\begin{aligned}
 d &= A \oplus B \oplus b_i = \overline{(A \oplus B) \oplus b_i} = \overline{(A \oplus B)(\overline{A \oplus B})b_i} \cdot \overline{b_i(A \oplus B)b_i} \\
 b &= \overline{AB} + b_i(\overline{A \oplus B}) = \overline{\overline{AB} + b_i(\overline{A \oplus B})} \\
 &= \overline{\overline{AB} \cdot \overline{b_i(A \oplus B)}} = \overline{B(\overline{A} + \overline{B})} \cdot \overline{b_i(\overline{b_i} + (\overline{A} \oplus \overline{B}))} \\
 &= \overline{B \cdot \overline{AB} \cdot b_i[\overline{b_i} \cdot (\overline{A} \oplus \overline{B})]}
 \end{aligned}$$

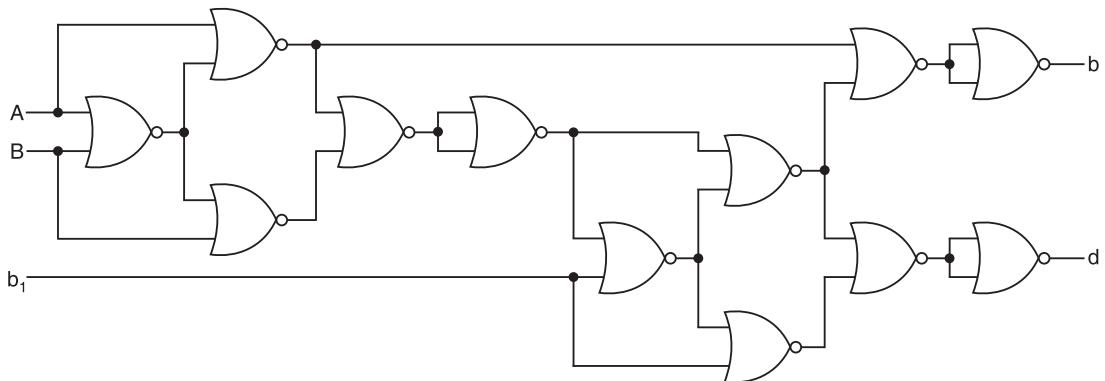


**Figure 7.18** Logic diagram of a full-subtractor using only 2-input NAND gates.

#### NOR logic

$$\begin{aligned}
 d &= A \oplus B \oplus b_i = \overline{(A \oplus B) \oplus b_i} \\
 &= \overline{(A \oplus B)b_i + (A \oplus B)\overline{b_i}} \\
 &= \overline{[(A \oplus B) + (A \oplus B)\overline{b_i}][b_i + (A \oplus B)\overline{b_i}]}
 \end{aligned}$$

$$\begin{aligned}
 &= \overline{(A \oplus B)} + \overline{(A \oplus B) + b_i} + \overline{b_i + (A \oplus B) + b_i} \\
 &= \overline{(A \oplus B)} + \overline{(A \oplus B) + b_i} + \overline{b_i} + \overline{(A \oplus B) + b_i} \\
 b &= \overline{AB} + b_i(\overline{A} \oplus \overline{B}) \\
 &= \overline{A(A+B)} + (\overline{A} \oplus \overline{B})[(A \oplus B) + b_i] \\
 &= \overline{A} + \overline{\overline{A+B}} + (A \oplus B) + (A \oplus B) + b_i
 \end{aligned}$$

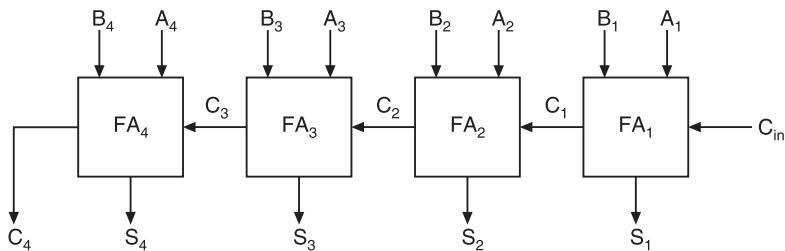


**Figure 7.19** Logic diagram of a full subtractor using only 2-input NOR gates.

## 7.5 BINARY PARALLEL ADDER

A binary parallel adder is a digital circuit that adds two binary numbers in parallel form and produces the arithmetic sum of those numbers in parallel form. It consists of full adders connected in a chain, with the output carry from each full-adder connected to the input carry of the next full-adder in the chain.

Figure 7.20 shows the interconnection of four full-adder (FA) circuits to provide a 4-bit parallel adder. The augend bits of A and addend bits of B are designated by subscript numbers from right to left, with subscript 1 denoting the lower-order bit. The carries are connected in a chain through the full-adders. The input carry to the adder is  $C_{in}$  and the output carry is  $C_4$ . The S outputs generate the required sum bits. When the 4-bit full-adder circuit is enclosed within an IC package, it has four terminals for the augend bits, four terminals for the addend bits, four terminals



**Figure 7.20** Logic diagram of a 4-bit binary parallel adder.

for the sum bits, and two terminals for the input and output carries. An  $n$ -bit parallel adder requires  $n$ -full adders. It can be constructed from 4-bit, 2-bit, and 1-bit full adder ICs by cascading several packages. The output carry from one package must be connected to the input carry of the one with the next higher-order bits. The 4-bit full adder is a typical example of an MSI function.

### 7.5.1 The Ripple Carry Adder

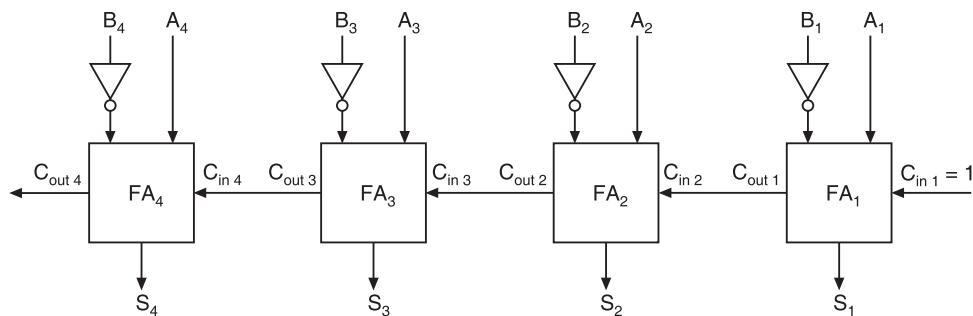
In the parallel adder discussed above, the carry-out of each stage is connected to the carry-in of the next stage. The sum and carry-out bits of any stage cannot be produced, until some time after the carry-in of that stage occurs. This is due to the propagation delays in the logic circuitry, which lead to a time delay in the addition process. The carry propagation delay for each full-adder is the time between the application of the carry-in and the occurrence of the carry-out.

Referring to the 4-bit parallel adder in Figure 7.20, we see that the sum ( $S_1$ ) and carry-out ( $C_1$ ) bits given by  $FA_1$  are not valid, until after the propagation delay of  $FA_1$ . Similarly, the sum  $S_2$  and carry-out ( $C_2$ ) bits given by  $FA_2$  are not valid until after the cumulative propagation delay of two full adders ( $FA_1$  and  $FA_2$ ), and so on. At each stage, the sum bit is not valid until after the carry bits in all the preceding stages are valid. In effect, carry bits must propagate or ripple through all stages before the most significant sum bit is valid. Thus, the total sum (the parallel output) is not valid until after the cumulative delay of all the adders.

The parallel adder in which the carry-out of each full-adder is the carry-in to the next most significant adder as shown in Figure 7.20 is called a *ripple carry adder*. The greater the number of bits that a ripple carry adder must add, the greater the time required for it to perform a valid addition. If two numbers are added such that no carries occur between stages, then the add time is simply the propagation time through a single full-adder.

## 7.6 4-BIT PARALLEL SUBTRACTOR

The subtraction of binary numbers can be carried out most conveniently by means of complements as discussed in Chapter 1. Remember that the subtraction  $A - B$  can be done by taking the 2's complement of  $B$  and adding it to  $A$ . The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits. The 1's complement can be implemented with inverters as shown in Figure 7.21.



**Figure 7.21** Logic diagram of a 4-bit parallel subtractor.

## 7.7 BINARY ADDER-SUBTRACTOR

Figure 7.22 shows a 4-bit adder-subtractor circuit. Here the addition and subtraction operations are combined into one circuit with one common binary adder. This is done by including an X-OR gate with each full-adder. The mode input  $M$  controls the operation. When  $M = 0$ , the circuit is an adder, and when  $M = 1$ , the circuit becomes a subtractor. Each X-OR gate receives input  $M$  and one of the inputs of  $B$ . When  $M = 0$ , we have  $B \oplus 0 = B$ . The full-adder receives the value of  $B$ , the input carry is 0 and the circuit performs  $A + B$ . When  $M = 1$ , we have  $B \oplus 1 = \bar{B}$  and  $C_1 = 1$ . The  $B$  inputs are complemented and a 1 is added through the input carry. The circuit performs the operation  $A$  plus the 2's complement of  $B$ .

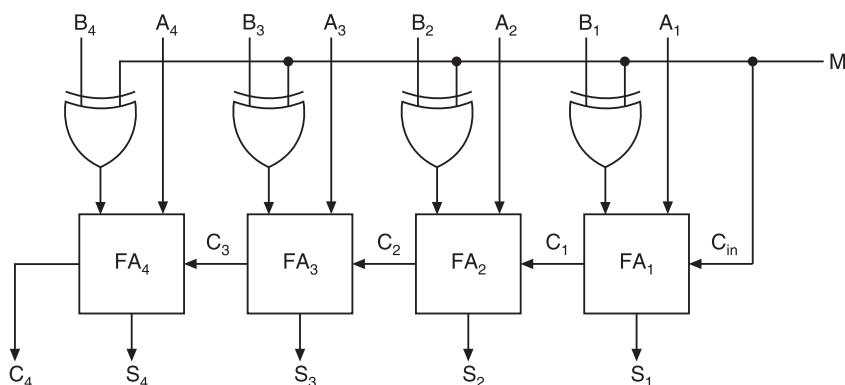


Figure 7.22 Logic diagram of a 4-bit binary adder-subtractor.

## 7.8 THE LOOK-AHEAD-CARRY ADDER

In the case of the parallel adder, the speed with which an addition can be performed is governed by the time required for the carries to propagate or ripple through all of the stages of the adder. The look-ahead-carry adder speeds up the process by eliminating this ripple carry delay. It examines all the input bits simultaneously and also generates the carry-in bits for all the stages simultaneously.

The method of speeding up the addition process is based on the two additional functions of the full-adder, called the *carry generate* and *carry propagate* functions.

Consider one full adder stage; say the  $n$ th stage of a parallel adder shown in Figure 7.23. We know that it is made of two half-adders and that the half-adder contains an X-OR gate to produce the sum and an AND gate to produce the carry. If both the bits  $A_n$  and  $B_n$  are 1s, a carry has to be generated in this stage regardless of whether the input carry  $C_{in}$  is a 0 or a 1. This is called generated carry, expressed as  $G_n = A_n \cdot B_n$  which has to appear at the output through the OR gate as shown in Figure 7.23.

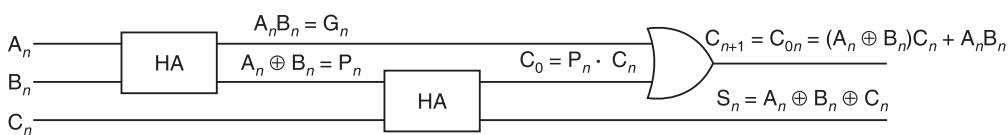


Figure 7.23 A full adder ( $n$ th stage of a parallel adder).

There is another possibility of producing a carry out. X-OR gate inside the half-adder at the input produces an intermediary sum bit—call it  $P_n$ —which is expressed as  $P_n = A_n \oplus B_n$ . Next  $P_n$  and  $C_n$  are added using the X-OR gate inside the second half adder to produce the final sum bit  $S_n = P_n \oplus C_n = A_n \oplus B_n \oplus C_n$  and output carry  $C_0 = P_n \cdot C_n = (A_n \oplus B_n)C_n$  which becomes input carry for the  $(n + 1)$ th stage.

Consider the case of both  $P_n$  and  $C_n$  being 1. The input carry  $C_n$  has to be propagated to the output only if  $P_n$  is 1. If  $P_n$  is 0, even if  $C_n$  is 1, the AND gate in the second half-adder will inhibit  $C_n$ . We may thus call  $P_n$  as the propagated carry as this is associated with enabling propagation of  $C_n$  to the carry output of the  $n$ th stage which is denoted as  $C_{n+1}$  or  $C_{0n}$ . So, we can say that the carryout of the  $n$ th stage is 1 when either  $G_n = 1$  or  $P_n \cdot C_n = 1$  or both  $G_n$  and  $P_n \cdot C_n$  are equal to 1.

For the final sum and carry outputs of the  $n$ th stage, we get the following Boolean expressions.

$$S_n = P_n \oplus C_n \text{ where } P_n = A_n \oplus B_n$$

$$C_{on} = C_{n+1} = G_n + P_n C_n \text{ where } G_n = A_n \cdot B_n$$

Observe the recursive nature of the expression for the output carry at the  $n$ th stage which becomes the input carry for the  $(n + 1)$ st stage. By successive substitution, it is possible to express the output carry of a higher significant stage in terms of the applied input variables  $A$ ,  $B$  and the carry-in to the LSB adder. The carry-in to each stage is the carry-out of the previous stage.

Based on these, the expressions for the carry-outs of various full adders are as follows:

$$C_1 = G_0 + P_0 \cdot C_0$$

$$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$$

$$C_3 = G_2 + P_2 \cdot C_2 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

$$C_4 = G_3 + P_3 \cdot C_3 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

The general expression for  $n$  stages designated as 0 through  $(n - 1)$  would be

$$C_n = G_{n-1} + P_{n-1} \cdot C_{n-1} = G_{n-1} + P_{n-1} \cdot G_{n-2} + P_{n-1} \cdot P_{n-2} \cdot G_{n-3} + \dots + P_{n-1} \cdot \dots \cdot P_0 \cdot C_0$$

Observe that the final output carry is expressed as a function of the input variables in SOP form, which is a two-level AND-OR or equivalent NAND-NAND form. To produce the output carry for any particular stage, it is clear that it requires only that much time required for the signals to pass through two levels only. Hence the circuit for look-ahead-carry introduces a delay corresponding to two gate levels. The block diagram of a 4 stage look-ahead-carry parallel adder is shown in Figure 7.24.

Observe that the full look-ahead-scheme requires the use of OR gate with  $(n + 1)$  inputs and AND gates with number of inputs varying from 2 to  $(n + 1)$ .

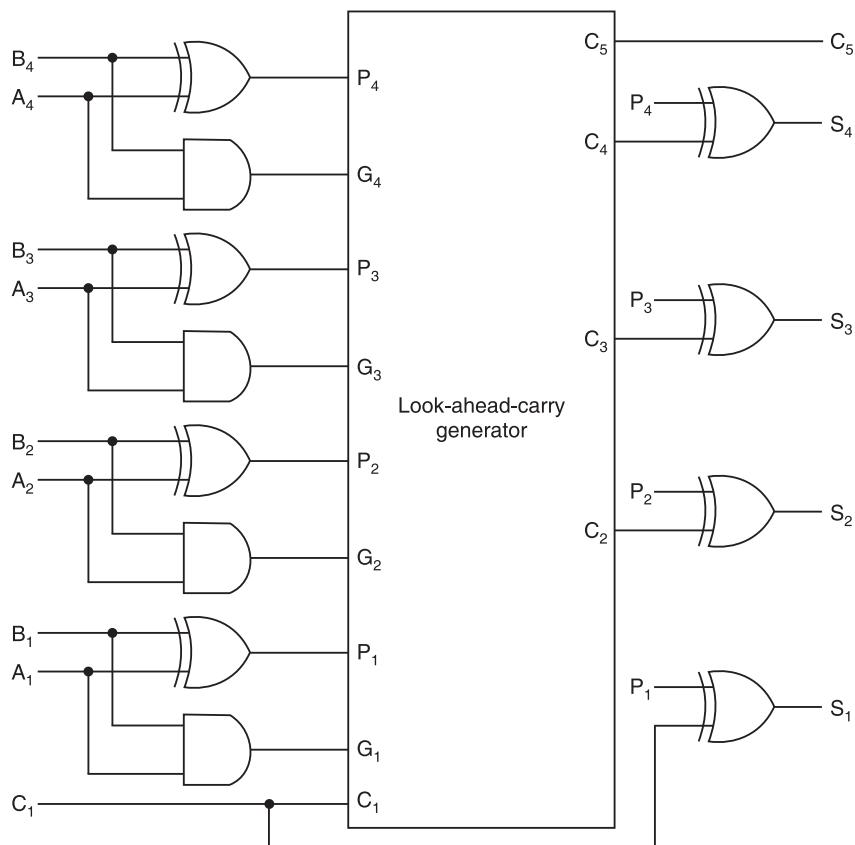


Figure 7.24 Logic diagram of a 4-bit look-ahead-carry adder.

## 7.9 IC PARALLEL ADDERS

Several parallel adders are available as ICs. The most common one is a 4-bit parallel adder IC that contains four interconnected full-adders and a look-ahead-carry circuit needed for high speed operation. The 7483A, the 74LS83A, the 74283, and the 74LS283 are all TTL 4-bit parallel adder chips. Figure 7.25 shows the functional symbol for the 74LS83 4-bit parallel adder (and its equivalents). The inputs to this IC are two 4-bit numbers,  $A_3A_2A_1A_0$  and  $B_3B_2B_1B_0$  and the carry  $C_0$  into the LSB position; the outputs are the sum bits  $S_3S_2S_1S_0$  and the carry  $C_4$  out of the MSB position. The sum bits are often labelled  $\Sigma_3\Sigma_2\Sigma_1\Sigma_0$ .

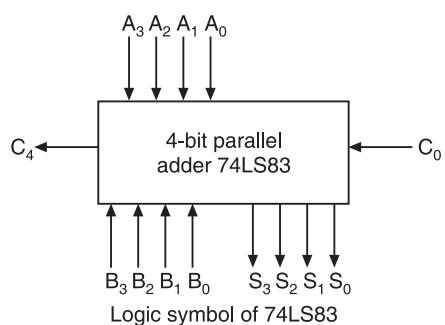


Figure 7.25 Logic symbol of 74LS83.

### 7.9.1 Cascading IC Parallel Adders

The addition of large binary numbers can be accomplished by cascading two or more parallel adder chips. When two 74LS83 chips are cascaded to add two 8-bit numbers, the first adder adds the 4 LSBs of the numbers. The  $C_4$  output of this adder is connected as the input carry to the first position of the second adder which adds the 4 MSBs of the numbers. The eight sum outputs represent the resultant sum of the two 8-bit numbers. The  $C_8$  is the carry-out of the last position (MSB) of the second adder. The  $C_8$  can be used as an overflow bit or as a carry into another adder stage if still larger binary numbers are to be handled.

## 7.10 2'S COMPLEMENT ADDITION AND SUBTRACTION USING PARALLEL ADDERS

Most modern computers use the 2's complement system to represent negative numbers and to perform subtraction. Both the addition and subtraction operations of signed numbers can be performed using only the addition operation, if we use the 2's complement form to represent negative numbers.

Figure 7.26 shows a complete circuit that can perform both addition and subtraction in the 2's complement. This adder/subtractor circuit is controlled by the control signal  $\overline{\text{ADD/SUB}}$ . When the  $\overline{\text{ADD/SUB}}$  level is HIGH, the circuit performs the addition of the numbers stored in registers A and B. When the  $\overline{\text{ADD/SUB}}$  level is LOW, the circuit subtracts the number in register B from the number in register A. The operation is described as follows:

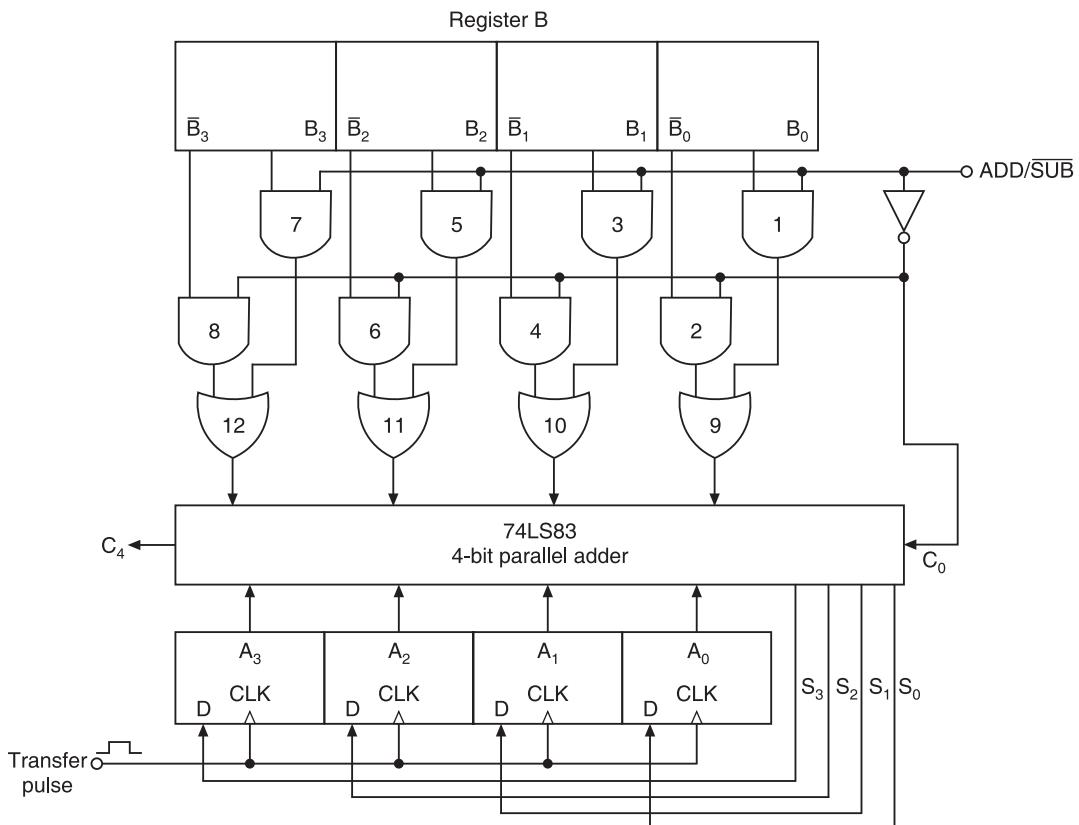
When  $\overline{\text{ADD/SUB}}$  is a 1:

1. AND gates 1, 3, 5, and 7 are enabled, allowing  $B_0$ ,  $B_1$ ,  $B_2$ , and  $B_3$  to pass to the OR gates 9, 10, 11, and 12. AND gates 2, 4, 6, and 8 are disabled, blocking  $\bar{B}_0$ ,  $\bar{B}_1$ ,  $\bar{B}_2$ , and  $\bar{B}_3$  from reaching the OR gates 9, 10, 11, and 12.
2. The levels  $B_0$  to  $B_3$  pass through the OR gates to the 4-bit parallel adder, to be added to the bits  $A_0$  to  $A_3$ . The sum appears at the outputs  $S_0$  to  $S_3$ .
3.  $\overline{\text{ADD/SUB}} = 1$  causes no carry into the adder.

When  $\overline{\text{ADD/SUB}}$  is a 0:

1. AND gates 1, 3, 5, and 7 are disabled, blocking  $B_0$ ,  $B_1$ ,  $B_2$ , and  $B_3$  from reaching the OR gates 9, 10, 11, and 12. AND gates 2, 4, 6, and 8 are enabled allowing  $\bar{B}_0$ ,  $\bar{B}_1$ ,  $\bar{B}_2$ , and  $\bar{B}_3$  to pass to the OR gates.
2. The levels  $\bar{B}_0$  to  $\bar{B}_3$  pass through the OR gates into the 4-bit parallel adder, to be added to bits  $A_0$  to  $A_3$ . The  $C_0$  is now 1. Thus, the number in register B is converted to its 2's complement form.
3. The difference appears at the outputs  $S_0$  to  $S_3$ .

Circuits like the adder/subtractor of Figure 7.26 are used in computers because they provide a relatively simple means for adding and subtracting signed binary numbers. In most computers, the output is usually transferred into the register A (accumulator) so that the results of the addition or subtraction always end up stored in the register A. This is accomplished by applying a transfer pulse to the CLK inputs of register A.



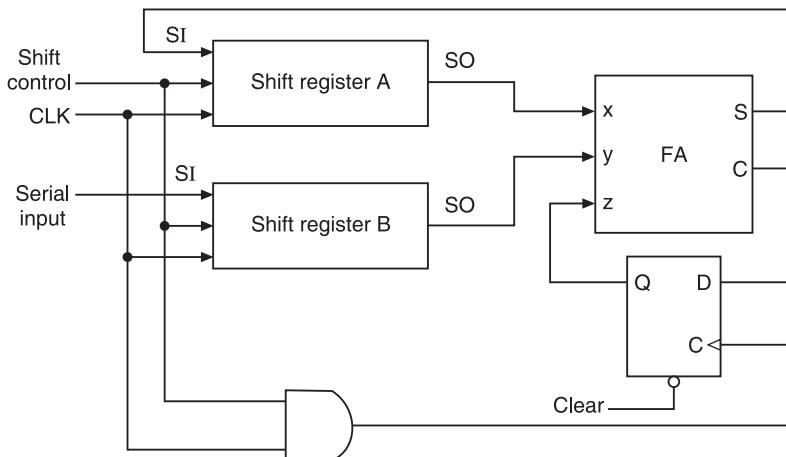
**Figure 7.26** Logic diagram of a parallel adder/subtractor using 2's complement system.

## 7.11 SERIAL ADDER

A serial adder is used to add binary numbers in serial form. The two binary numbers to be added serially are stored in two shift registers A and B. Bits are added one pair at a time through a single full adder (FA) circuit as shown in Figure 7.27. The carry out of the full-adder is transferred to a D flip-flop. The output of this flip-flop is then used as the carry input for the next pair of significant bits. The sum bit from the S output of the full adder could be transferred to a third shift register. By shifting the sum into A while the bits of A are shifted out, it is possible to use one register for storing both augend and the sum bits. The serial input register B can be used to transfer a new binary number while the addend bits are shifted out during the addition.

The operation of the serial adder is as follows. Initially register A holds the augend, register B holds the addend and the carry flip-flop is cleared to 0. The outputs (SO) of A and B provide a pair of significant bits for the full-adder at  $x$  and  $y$ . The shift control enables both registers and carry flip-flop, so, at the clock pulse both registers are shifted once to the right, the sum bit from S enters the left most flip-flop of A, and the output carry is transferred into flip-flop Q. The shift control enables the registers for a number of clock pulses equal to the number of bits of the registers. For each succeeding clock pulse a new sum bit is transferred to A, a new carry is transferred to Q, and

both registers are shifted once to the right. This process continues until the shift control is disabled. Thus the addition is accomplished by passing each pair of bits together with the previous carry through a single full adder circuit and transferring the sum, one bit at a time, into register A.



**Figure 7.27** Logic diagram of a serial adder.

Initially, register A and the carry flip-flop are cleared to 0 and then the first number is added from B. While B is shifted through the full adder, a second number is transferred to it through its serial input. The second number is then added to the content of register A while a third number is transferred serially into register B. This can be repeated to form the addition of two, three, or more numbers and accumulate their sum in register A.

### 7.11.1 Difference between Serial and Parallel Adders

The parallel adder uses registers with parallel load, whereas the serial adder uses shift registers. The number of full adder circuits in the parallel adder is equal to the number of bits in the binary numbers, whereas the serial adder requires only one full adder circuit and a carry flip-flop. Excluding the registers, the parallel adder is a combinational circuit, whereas the serial adder is a sequential circuit. The sequential circuit in the serial adder consists of a full-adder and a flip-flop that stores the output carry.

## 7.12 BCD ADDER

The BCD addition process has been discussed in Chapter 1. It is briefly reviewed here.

1. Add the 4-bit BCD code groups for each decimal digit position using ordinary binary addition.
2. For those positions where the sum is 9 or less, the sum is in proper BCD form and no correction is needed.
3. When the sum of two digits is greater than 9, a correction of 0110 should be added to that sum, to produce the proper BCD result. This will produce a carry to be added to the next decimal position.

A BCD adder circuit must be able to operate in accordance with the above steps. In other words, the circuit must be able to do the following.

1. Add two 4-bit BCD code groups, using straight binary addition.
2. Determine, if the sum of this addition is greater than 1001 (decimal 9); if it is, add 0110 (decimal 6) to this sum and generate a carry to the next decimal position.

The first requirement is easily met by using a 4-bit binary parallel adder such as the 74LS83 IC. For example, if the two BCD code groups  $A_3A_2A_1A_0$  and  $B_3B_2B_1B_0$  are applied to a 4-bit parallel adder, the adder will output  $S_4S_3S_2S_1S_0$ , where  $S_4$  is actually  $C_4$ , the carry-out of the MSB bits.

The sum output  $S_4S_3S_2S_1S_0$  can range anywhere from 00000 to 10010 (when both the BCD code groups are 1001 = 9). The circuitry for a BCD adder must include the logic needed to detect whenever the sum is greater than 01001, so that the correction can be added in. Those cases, where the sum is greater than 1001 are listed in Table 7.1.

**Table 7.1**

<b>S<sub>4</sub></b>	<b>S<sub>3</sub></b>	<b>S<sub>2</sub></b>	<b>S<sub>1</sub></b>	<b>S<sub>0</sub></b>	<b>Decimal number</b>
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18

Let us define a logic output X that will go HIGH only when the sum is greater than 01001 (i.e. for the cases in Table 7.1). If we examine these cases, we see that X will be HIGH for either of the following conditions.

1. Whenever  $S_4 = 1$  (sum greater than 15)
2. Whenever  $S_3 = 1$  and either  $S_2$  or  $S_1$  or both are 1 (sums 10 to 15)

This condition can be expressed as

$$X = S_4 + S_3(S_2 + S_1)$$

Whenever  $X = 1$ , it is necessary to add the correction factor 0110 to the sum bits, and to generate a carry. Figure 7.28 shows the complete circuitry for a BCD adder, including the logic circuit implementation for X.

The circuit consists of three basic parts. The two BCD code groups  $A_3A_2A_1A_0$  and  $B_3B_2B_1B_0$  are added together in the upper 4-bit adder, to produce the sum  $S_4S_3S_2S_1S_0$ . The logic gates shown implement the expression for X. The lower 4-bit adder will add the correction 0110 to the sum bits, only when X = 1, producing the final BCD sum output represented by  $\Sigma_3\Sigma_2\Sigma_1\Sigma_0$ . The X is also the carry-out that is produced when the sum is greater than 01001. Of course, when X = 0, there is no carry and no addition of 0110. In such cases,  $\Sigma_3\Sigma_2\Sigma_1\Sigma_0 = S_3S_2S_1S_0$ .

Two or more BCD adders can be connected in cascade when two or more digit decimal numbers are to be added. The carry-out of the first BCD adder is connected as the carry-in of the

second BCD adder, the carry-out of the second BCD adder is connected as the carry-in of the third BCD adder and so on.

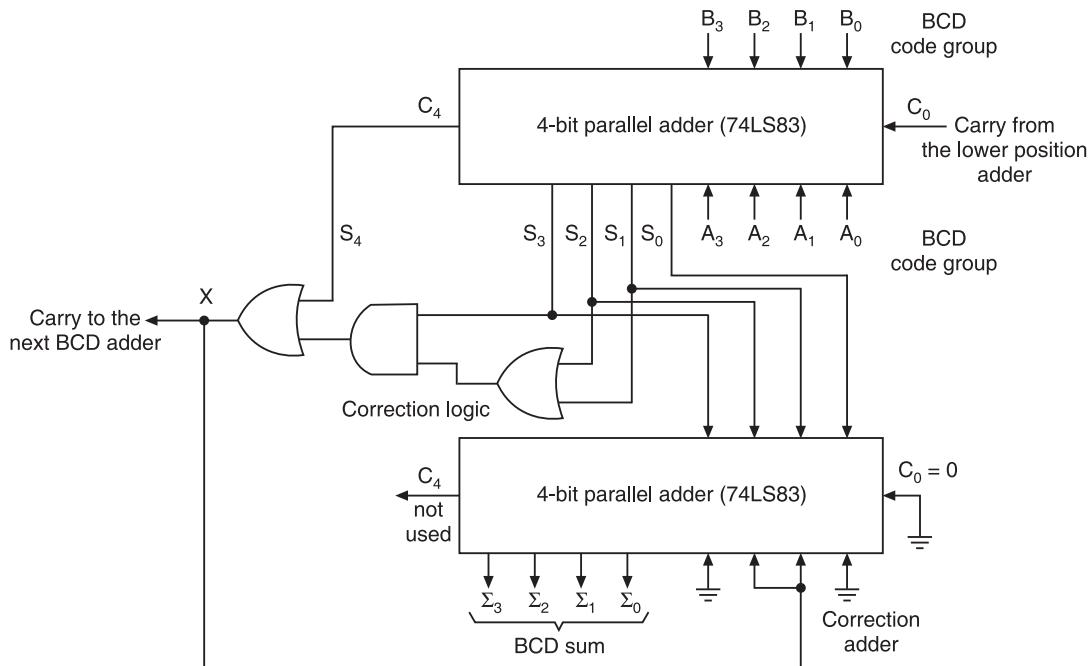


Figure 7.28 Logic diagram of a BCD adder using two 4-bit adders and a correction-detector circuit.

### 7.13 EXCESS-3 (XS-3) ADDER

To perform Excess-3 additions, we have to

1. Add two XS-3 code groups.
2. If carry = 1, add 0011 (3) to the sum of those two code groups.

If carry = 0, subtract 0011(3), i.e. add 1101 (13 in decimal) to the sum of those two code groups.

**EXAMPLE 7.1** (a) Add 9 and 5 and (b) 4 and 3 in XS-3.

**Solution**

(a) $  \begin{array}{r}  1 & 1 & 0 & 0 & \text{9 in XS-3} \\  + & 1 & 0 & 0 & 0 & \text{5 in XS-3} \\  \hline  1 & 0 & 1 & 0 & 0 & \text{There is a carry} \\  + & 0 & 0 & 1 & 1 & \text{add 3 to each group} \\  \hline  0 & 1 & 0 & 0 & 1 & 4 \text{ in XS-3} \\  (1) & (4) & & & &  \end{array}  $	(b) $  \begin{array}{r}  0 & 1 & 1 & 1 & \text{4 in XS-3} \\  + & 0 & 1 & 1 & 0 & \text{3 in XS-3} \\  \hline  1 & 1 & 0 & 1 & & \text{no carry} \\  + & 1 & 1 & 0 & 1 & \text{Subtract 3 (i.e. add 13)} \\  \hline  1 & 1 & 0 & 1 & 0 & 7 \text{ in XS-3} \\  (7) & & & & &  \end{array}  $
---	--

Implementation of XS-3 adder using 4-bit binary adders is shown in Figure 7.29. The augend ( $A_3A_2A_1A_0$ ) and addend ( $B_3B_2B_1B_0$ ) in XS-3 are added using the 4-bit parallel adder. If the carry is a 1, then 0011 (3) is added to the sum bits  $S_3, S_2, S_1, S_0$  of the upper adder in the lower

4-bit parallel adder. If the carry is a 0, then 1101(13) is added to the sum bits (This is equivalent to subtracting 0011(3) from the sum bits. The correct sum in XS-3 is obtained as shown in Figure 7.29.

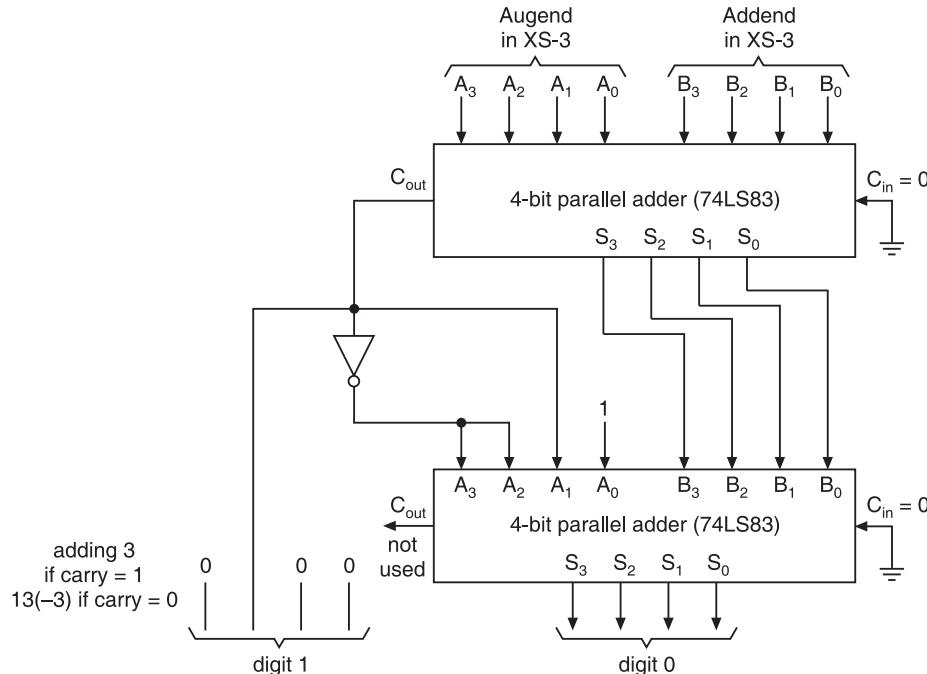


Figure 7.29 Logic diagram of an XS-3 adder.

## 7.14 EXCESS-3 (XS-3) SUBTRACTOR

To perform Excess-3 subtraction, we have to

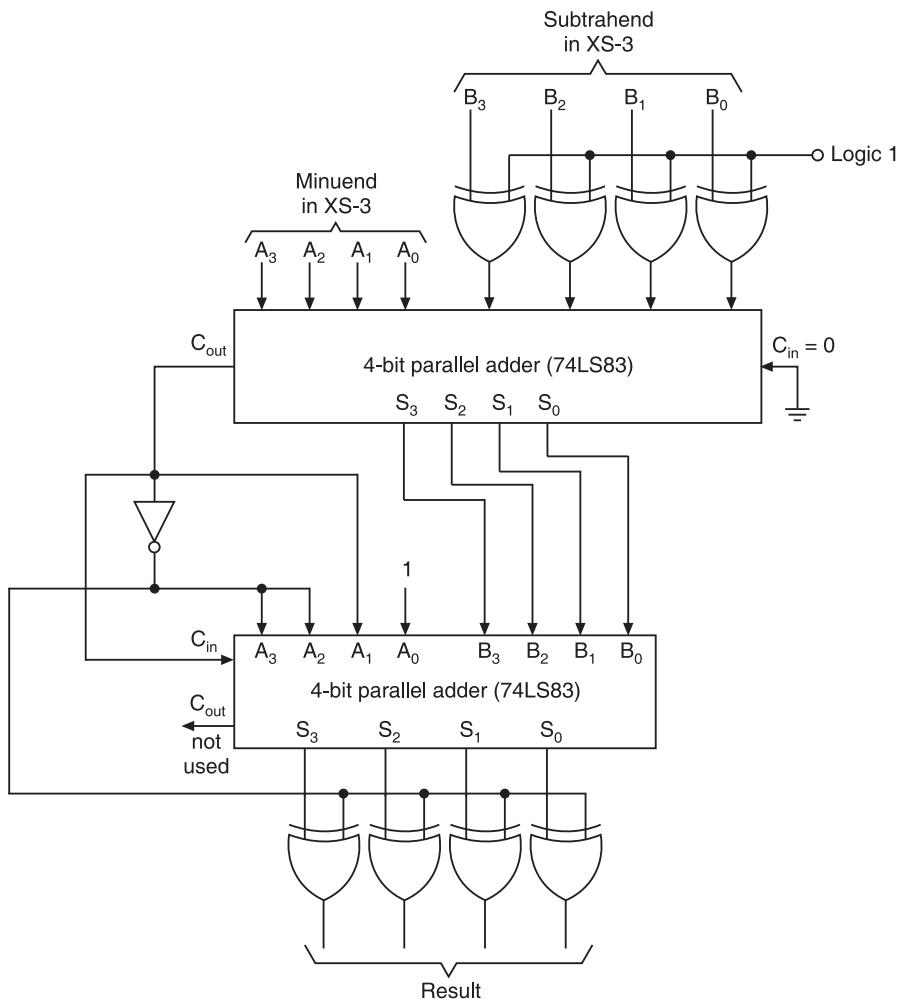
1. Complement the subtrahend.
2. Add the complemented subtrahend to the minuend.
3. If carry = 1, result is positive. Add 3 and end around carry to the result. If carry = 0, the result is negative. Subtract 3, i.e. add 13 and take the 1's complement of the result.

**EXAMPLE 7.2** Perform: (a) 9 – 4 and (b) 4 – 9 in XS-3.

**Solution**

(a) $  \begin{array}{r}  1 \ 1 \ 0 \ 0 \quad 9 \text{ in XS-3} \\  + 1 \ 0 \ 0 \ 0 \quad \text{Complement of } 4 \text{ in XS-3} \\  \hline  1 \ 0 \ 1 \ 0 \quad \text{There is a carry} \\  + 0 \ 0 \ 1 \ 1 \quad \text{Add } 0011(3) \\  \hline  0 \ 1 \ 1 \ 1 \quad \text{End around carry} \\  \hline  \xrightarrow{1} \ 1 \ 0 \ 0 \ 0 \quad 5 \text{ in XS-3}  \end{array}  $	(b) $  \begin{array}{r}  0 \ 1 \ 1 \ 1 \quad 4 \text{ in XS-3} \\  + 0 \ 0 \ 1 \ 1 \quad \text{Complement of } 9 \text{ in XS-3} \\  \hline  1 \ 0 \ 1 \ 0 \quad \text{There is no carry} \\  + 1 \ 1 \ 0 \ 1 \quad \text{Subtract } 3, \text{ i.e. add } 13 \\  \hline  1 \ 0 \ 1 \ 1 \ 1 \quad \text{Complement the result} \\  \hline  1 \ 0 \ 0 \ 0 \quad 5 \text{ in XS-3}  \end{array}  $
--	---

Implementation of the XS-3 subtractor using two 4-bit parallel adders is shown in Figure 7.30. The minuend and the 1's complement of the subtrahend in XS-3 are added in the upper 4-bit parallel adder. If the carry-out from the upper adder is a 0, then 1101 is added to the sum bits of the upper adder in the lower adder and the sum bits of the lower adder are complemented to get the result. If the carry-out from the upper adder is a 1, then 3 = 0011 is added to the sum bits of the lower adder and the sum bits of the lower adder give the result.



**Figure 7.30** Logic diagram of an XS-3 subtractor using 4-bit binary adders.

## 7.15 BINARY MULTIPLIERS

In Chapter 2 we discussed binary multiplication by the paper and pencil method. The paper and pencil method is modified somewhat in digital machines because a binary adder can add only two binary numbers at a time.

In a binary multiplier, instead of adding all the partial products at the end, they are added two at a time and their sum accumulated in a register (the accumulator register). In addition, when the multiplier bit is a 0, 0s are not written down and added because it does not affect the final result. Instead, the multiplicand is shifted left by one bit.

The multiplication of 1110 by 1001 using this process is illustrated below.

Multiplicand: 1 1 1 0

Multiplier: 1 0 0 1

1 1 1 0	The LSB of the multiplier is a 1; write down the multiplicand; shift the multiplicand one position to the left (1 1 1 0 0).
1 1 1 0	The second multiplier bit is a 0; write down the previous result 1 1 1 0; shift the multiplicand to the left again (1 1 1 0 0 0).
1 1 1 0	The third multiplier bit is a 0; write down the previous result 1 1 1 0; shift the multiplicand to the left again (1 1 1 0 0 0 0).
+ 1 1 1 0 0 0 0	The fourth multiplier bit is a 1; write down the new multiplicand; add it to the first partial product to obtain the final product.

1 1 1 1 1 0

This multiplication process can be performed by the serial multiplier circuit shown in Figure 7.31, which multiplies two 4-bit numbers to produce an 8-bit product. The circuit consists of the following elements.

**X register:** A 4-bit shift register that stores the multiplier—it will shift right on the falling edge of the clock. Note that 0s are shifted in from the left.

**B register:** An 8-bit register that stores the multiplicand; it will shift left on the falling edge of the clock. Note that 0s are shifted in from the right.

**A register:** An 8-bit register, i.e. the accumulator that accumulates the partial products.

**Adder:** An 8-bit parallel adder that produces the sum of A and B registers. The adder outputs  $S_7$  through  $S_0$  are connected to the D inputs of the accumulator so that the sum can be transferred to the accumulator only when a clock pulse gets through the AND gate.

The circuit operation can be described by going through each step in the multiplication of 1110 by 1001. The complete process requires 4 clock cycles.

1. *Before the first clock pulse:* Prior to the occurrence of the first clock pulse, the register A is loaded with 00000000, the register B with the multiplicand 00001110, and the register X with the multiplier 1001. We can assume that each of these registers is loaded using its asynchronous inputs (i.e. PRESET and CLEAR). The output of the adder will be the sum of A and B, that is, 00001110.
2. *First clock pulse:* Since the LSB of the multiplier ( $X_0$ ) is a 1, the first clock pulse gets through the AND gate and its positive going transition transfers the sum outputs into the accumulator. The subsequent negative going transition causes the X and B registers to shift right and left, respectively. This, of course, produces a new sum of A and B.
3. *Second clock pulse:* The second bit of the original multiplier is now in  $X_0$ . Since this bit is a 0, the second clock pulse is inhibited from reaching the accumulator. Thus, the sum outputs are not transferred into the accumulator and the number in the accumulator does not change. The negative going transition of the clock pulse will again shift the X and B registers. Again a new sum is produced.

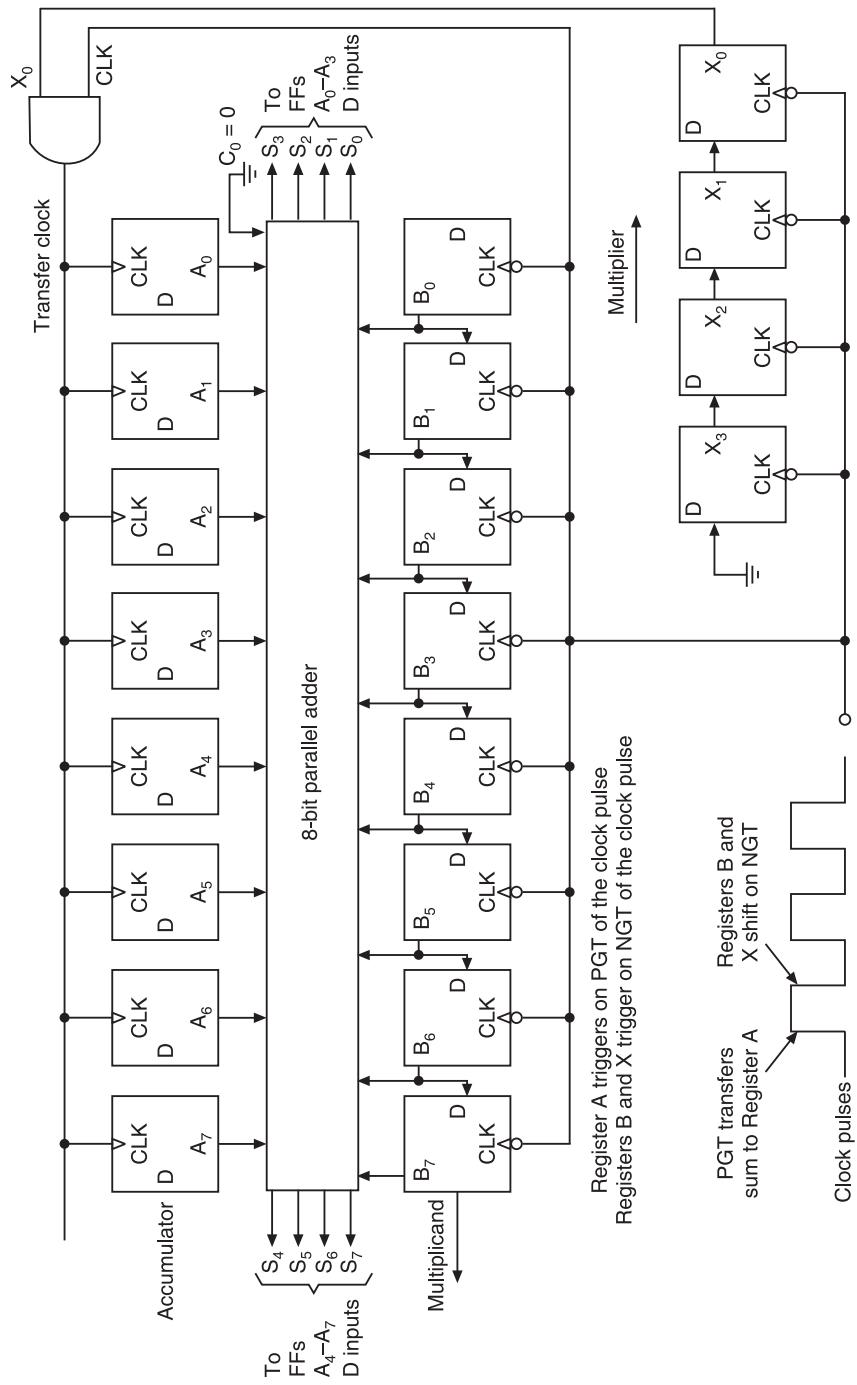


Figure 7.31 Logic diagram of a binary multiplier.

4. *Third clock pulse:* The third bit of the original multiplier is now in  $X_0$ ; since this bit is a 0, the third clock pulse is inhibited from reaching the accumulator. Thus, the sum outputs are not transferred into the accumulator and the number in the accumulator does not change. The negative going transition of the clock pulse will again shift the X and B registers. Again a new sum is produced.
5. *Fourth clock pulse:* The last bit of the original multiplier is now in  $X_0$ , and since it is a 1, the positive going transition of the fourth pulse transfers the sum into the accumulator. The accumulator now holds the final product. The negative going transition of the clock pulse shifts X and B again. Note that, X is now 0000, since all the multiplier bits have been shifted out.

## 7.16 CODE CONVERTERS

The availability of a large variety of codes for the same discrete elements of information results in the use of different codes by different digital systems. It is sometimes necessary to use the output of one system as the input to another. A conversion circuit must be inserted between the two systems if each uses different codes for the same information. Thus a code converter is a logic circuit whose inputs are bit patterns representing numbers (or characters) in one code and whose outputs are the corresponding representations in a different code. It makes two systems compatible even though each uses a different binary code. Code converters are usually multiple output circuits.

To convert from binary code A to binary code B, the input lines must supply the bit combination of elements as specified by code A and the output lines must generate the corresponding bit combination of code B. A combinational circuit performs this transformation by means of logic gates.

For example, a binary-to-Gray code converter has four binary input lines  $B_4$ ,  $B_3$ ,  $B_2$ , and  $B_1$ , and four Gray code output lines  $G_4$ ,  $G_3$ ,  $G_2$ , and  $G_1$ . When the input is 0010, for instance, the output should be 0011 and so forth. To design a code converter, we use a code table treating it as a truth table to express each output as a Boolean algebraic function of all the inputs.

In this example of binary-to-gray code conversion, we can treat the binary to the Gray code table as four truth tables to derive expressions for  $G_4$ ,  $G_3$ ,  $G_2$ , and  $G_1$ . Each of these four expressions would, in general, contain all the four input variables  $B_4$ ,  $B_3$ ,  $B_2$ , and  $B_1$ . Thus, this code converter is actually equivalent to four logic circuits, one for each of the truth tables.

The logic expressions derived for the code converter can be simplified using the usual techniques, including ‘don’t cares’ if present. Even if the input is an unweighted code, the same cell numbering method which we used earlier can be used, but the cell numbers must correspond to the input combinations as if they were an 8421 weighted code. For example, in Excess-3 to BCD conversion, number ABCD = 0110, which represents  $3_{10}$  is assigned the cell number 6 and not the cell number 3. Be careful to determine which input combinations, if any, will never occur and can be treated as don’t cares. Of course, it is the input bit patterns, and not the output bit patterns that determine don’t cares.

Integrated circuits (ICs) are available to convert data from one form to another. Binary-to-BCD conversions are most often encountered in connection with computer applications. Numerical data transmitted in BCD form from input devices must be converted to binary, so that arithmetic

operations can be performed on it. The binary results of arithmetic operations must be converted to BCD for transmission to output devices. Therefore, conversions are often accomplished by using the major components of the computer system itself rather than special converter circuits. Conversion tables may be stored in the ROM. In some systems, conversions are accomplished by the computer itself, through execution of a specially designed program. This is called software conversion, as opposed to the hardware conversion performed by logic circuits.

### 7.16.1 Design of a 4-bit Binary-to-Gray Code Converter

The input to the 4-bit binary-to-Gray code converter circuit is a 4-bit binary and the output is a 4-bit Gray code. There are 16 possible combinations of 4-bit binary input and all of them are valid. Hence no don't cares. The 4-bit binary and the corresponding Gray code are shown in the conversion table (Figure 7.32a). From the conversion table, we observe that the expressions for the outputs  $G_4$ ,  $G_3$ ,  $G_2$ , and  $G_1$  are as follows:

$$G_4 = \Sigma m(8, 9, 10, 11, 12, 13, 14, 15)$$

$$G_3 = \Sigma m(4, 5, 6, 7, 8, 9, 10, 11)$$

$$G_2 = \Sigma m(2, 3, 4, 5, 10, 11, 12, 13)$$

$$G_1 = \Sigma m(1, 2, 5, 6, 9, 10, 13, 14)$$

The K-maps for  $G_4$ ,  $G_3$ ,  $G_2$ , and  $G_1$  and their minimization are shown in Figure 7.32b. The minimal expressions for the outputs obtained from the K-map are:

$$G_4 = B_4$$

$$G_3 = \overline{B}_4 B_3 + B_4 \overline{B}_3 = B_4 \oplus B_3$$

$$G_2 = \overline{B}_3 B_2 + B_3 \overline{B}_2 = B_3 \oplus B_2$$

$$G_1 = \overline{B}_2 B_1 + B_2 \overline{B}_1 = B_2 \oplus B_1$$

So, the conversion can be achieved by using three X-OR gates as shown in the logic diagram in Figure 7.32c.

4-bit binary				4-bit Gray			
$B_4$	$B_3$	$B_2$	$B_1$	$G_4$	$G_3$	$G_2$	$G_1$
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

(a) Conversion table

Figure 7.32 4-bit binary-to-Gray code converter (Contd.)

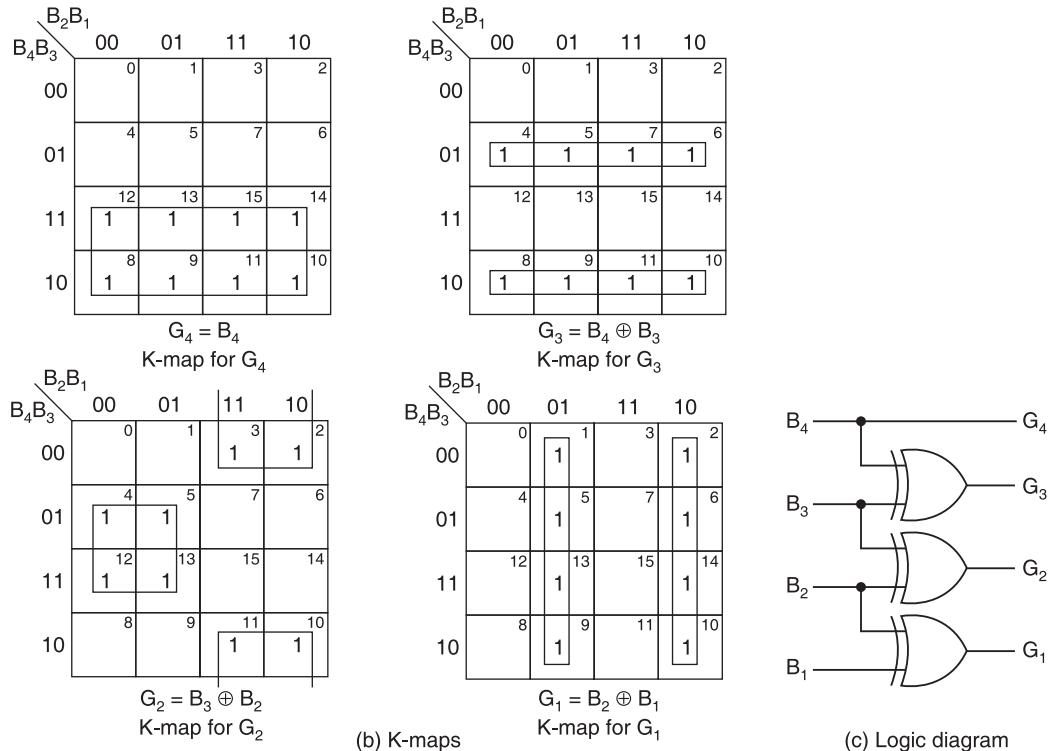


Figure 7.32 4-bit binary-to-Gray code converter.

### 7.16.2 Design of a 4-bit Gray-to-Binary Code Converter

The input to the 4-bit Gray code converter circuit is a 4-bit Gray code and the output is a 4-bit binary. There are 16 possible combinations of 4-bit Gray input and all of them are valid. Hence no don't cares. The 4-bit input Gray code and the corresponding output binary numbers are shown in the conversion table of Figure 7.33a. From the conversion table we observe that the expressions for the outputs  $B_4$ ,  $B_3$ ,  $B_2$  and  $B_1$  are:

$$B_4 = \Sigma m(12, 13, 15, 14, 10, 11, 9, 8) = \Sigma m(8, 9, 10, 11, 12, 13, 14, 15)$$

$$B_3 = \Sigma m(6, 7, 5, 4, 10, 11, 9, 8) = \Sigma m(4, 5, 6, 7, 8, 9, 10, 11)$$

$$B_2 = \Sigma m(3, 2, 5, 4, 15, 14, 9, 8) = \Sigma m(2, 3, 4, 5, 8, 9, 14, 15)$$

$$B_1 = \Sigma m(1, 2, 7, 4, 13, 14, 11, 8) = \Sigma m(1, 2, 4, 7, 8, 11, 13, 14)$$

Drawing the K-maps for  $B_4$ ,  $B_3$ ,  $B_2$  and  $B_1$  in terms of  $G_4$ ,  $G_3$ ,  $G_2$ , and  $G_1$  as shown in Figure 7.33b and simplifying them, the minimal expressions for the outputs are as follows:

$$B_4 = G_4$$

$$B_3 = \overline{G}_4 G_3 + G_4 \overline{G}_3 = G_4 \oplus G_3$$

$$B_2 = \overline{G}_4 G_3 \overline{G}_2 + \overline{G}_4 \overline{G}_3 G_2 + G_4 \overline{G}_3 \overline{G}_2 + G_4 G_3 G_2$$

$$= \overline{G}_4 (G_3 \oplus G_2) + G_4 (G_3 \oplus G_2) = G_4 \oplus G_3 \oplus G_2 = B_3 \oplus G_2$$

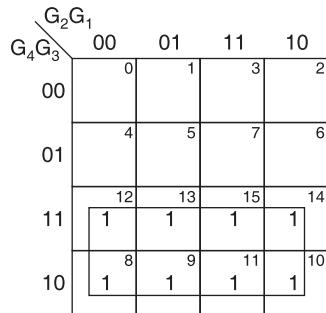
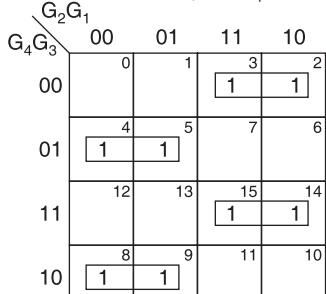
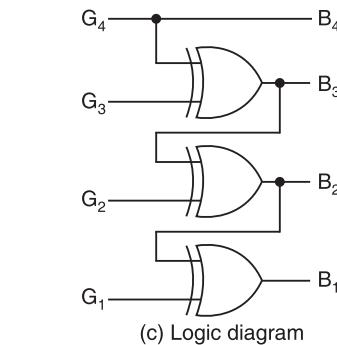
$$B_1 = \overline{G}_4 \overline{G}_3 \overline{G}_2 G_1 + \overline{G}_4 \overline{G}_3 G_2 \overline{G}_1 + \overline{G}_4 G_3 \overline{G}_2 G_1 + \overline{G}_4 G_3 \overline{G}_2 \overline{G}_1 + G_4 G_3 \overline{G}_2 G_1 \\ + G_4 G_3 G_2 \overline{G}_1 + G_4 \overline{G}_3 G_2 G_1 + G_4 \overline{G}_3 \overline{G}_2 \overline{G}_1$$

$$\begin{aligned}
 &= \overline{G_4} \overline{G_3} (G_2 \oplus G_1) + G_4 G_3 (G_2 \oplus G_1) + \overline{G_4} G_3 (\overline{G_2} \oplus G_1) + G_4 \overline{G}_3 (\overline{G_2} \oplus G_1) \\
 &= (G_2 \oplus G_1)(\overline{G_4} \oplus G_3) + (\overline{G_2} \oplus G_1)(G_4 \oplus G_3) \\
 &= G_4 \oplus G_3 \oplus G_2 \oplus G_1 \\
 &= B_2 \oplus G_1
 \end{aligned}$$

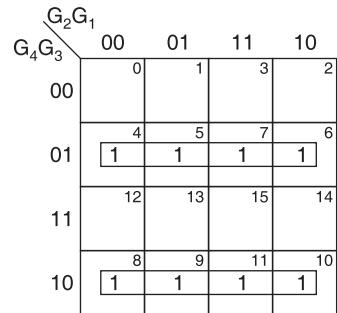
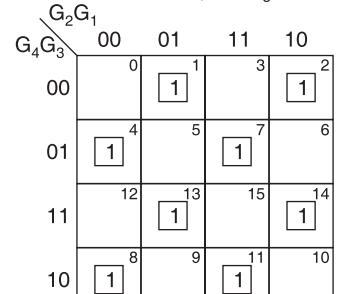
Based on the above expressions, a logic circuit can be drawn as shown in Figure 7.33c.

4-bit Gray				4-bit binary			
G <sub>4</sub>	G <sub>3</sub>	G <sub>2</sub>	G <sub>1</sub>	B <sub>4</sub>	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

(a) Conversion table

 $B_4 = G_4 \oplus G_3 \oplus G_2 \oplus G_1$   
K-map for  $B_4$  $B_2 = G_4 \oplus G_3 \oplus G_2$   
K-map for  $B_2$ 

(c) Logic diagram

 $B_3 = G_4 \oplus G_3$   
K-map for  $B_3$  $B_1 = G_4 \oplus G_3 \oplus G_2 \oplus G_1$   
K-map for  $B_1$ **Figure 7.33** 4-bit Gray-to-binary code converter.

### 7.16.3 Design of a 4-bit Binary-to-BCD Code Converter

The input is a 4-bit binary. There are 16 possible combinations of 4-bit binary inputs (representing 0–15) and all are valid. Hence there are no don't cares. Since the input is of 4 bits (i.e. a maximum of 2 decimal digits), the output has to be an 8-bit one; but since the first three bits will all be a 0 for all combinations of inputs, the output can be treated as a 5-bit one. The conversion is shown in the conversion table in Figure 7.34a. From the conversion table, we observe that the expressions for BCD outputs are as follows:

$$A = \Sigma m(10, 11, 12, 13, 14, 15)$$

$$B = \Sigma m(8, 9)$$

$$C = \Sigma m(4, 5, 6, 7, 14, 15)$$

$$D = \Sigma m(2, 3, 6, 7, 12, 13)$$

$$E = \Sigma m(1, 3, 5, 7, 9, 11, 13, 15)$$

Drawing the K-maps for the outputs and minimizing them as shown in Figure 7.34c the minimal expressions for the BCD outputs A, B, C, D, and E in terms of the 4-bit binary inputs  $B_4$ ,  $B_3$ ,  $B_2$ , and  $B_1$  are as follows:

$$A = B_4 B_3 + B_4 B_2$$

$$B = B_4 \bar{B}_3 \bar{B}_2$$

$$C = \bar{B}_4 B_3 + B_3 B_2$$

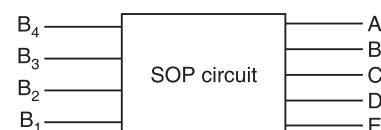
$$D = B_4 B_3 \bar{B}_2 + \bar{B}_4 B_2$$

$$E = B_1$$

A logic diagram can be drawn based on the above minimal expressions.

Decimal	4-bit binary				BCD output				
	$B_4$	$B_3$	$B_2$	$B_1$	A	B	C	D	E
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

(a) Conversion table



(b) Block diagram

**Figure 7.34** 4-bit binary-to-BCD code converter (*Contd.*)

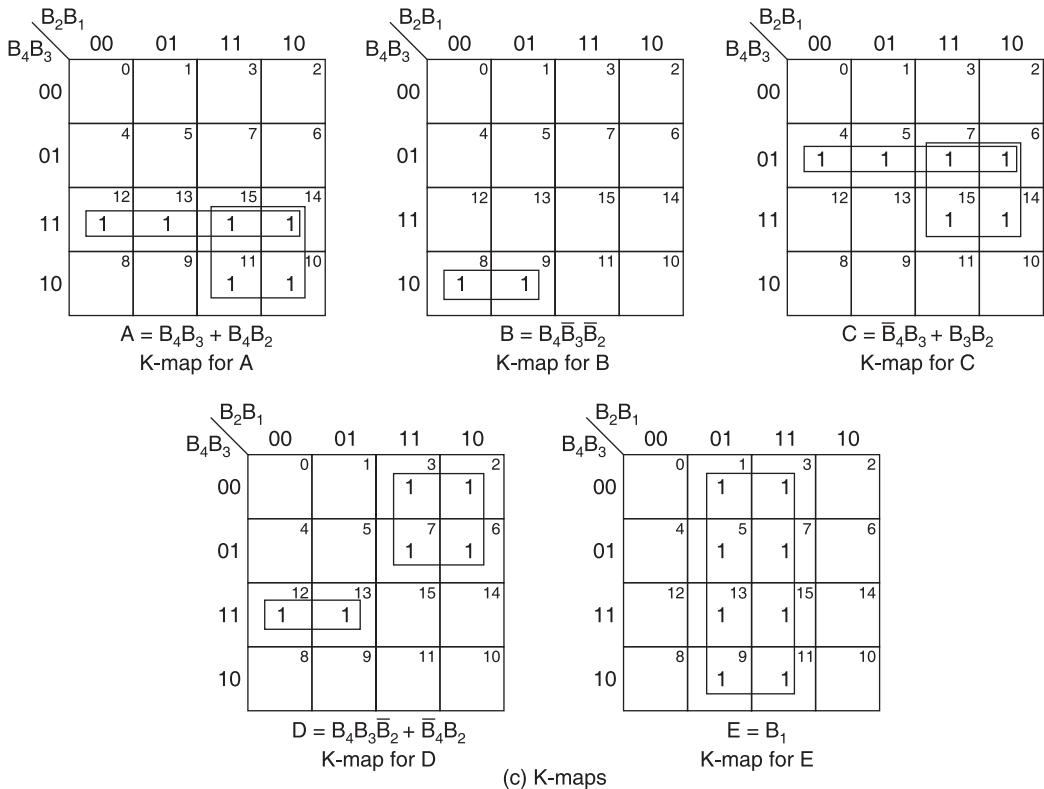


Figure 7.34 4-bit binary-to-BCD code converter.

#### 7.16.4 Design of a 4-bit BCD-to-XS-3 Code Converter

BCD means 8421 BCD. The 4-bit input BCD code ( $B_4 B_3 B_2 B_1$ ) and the corresponding output XS-3 code( $X_4 X_3 X_2 X_1$ ) numbers are shown in the conversion table in Figure 7.35a. The input combinations 1010, 1011, 1100, 1101, 1110, and 1111 are invalid in BCD. So they are treated as don't cares.

8421 code				XS-3 code			
$B_4$	$B_3$	$B_2$	$B_1$	$X_4$	$X_3$	$X_2$	$X_1$
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

(a) Conversion table

$$\begin{aligned}
 X_4 &= \Sigma m(5, 6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15) \\
 X_3 &= \Sigma m(1, 2, 3, 4, 9) + d(10, 11, 12, 13, 14, 15) \\
 X_2 &= \Sigma m(0, 3, 4, 7, 8) + d(10, 11, 12, 13, 14, 15) \\
 X_1 &= \Sigma m(0, 2, 4, 6, 8) + d(10, 11, 12, 13, 14, 15)
 \end{aligned}$$

The minimal expressions are

$$\begin{aligned}
 X_4 &= B_4 + B_3B_2 + B_3B_1 \\
 X_3 &= B_3\bar{B}_2\bar{B}_1 + \bar{B}_3B_1 + \bar{B}_3B_2 \\
 X_2 &= \bar{B}_2\bar{B}_1 + B_2B_1 \\
 X_1 &= \bar{B}_1
 \end{aligned}$$

(b) Minimal expressions

Figure 7.35 4-bit BCD-to-XS-3 code converter (Contd.)

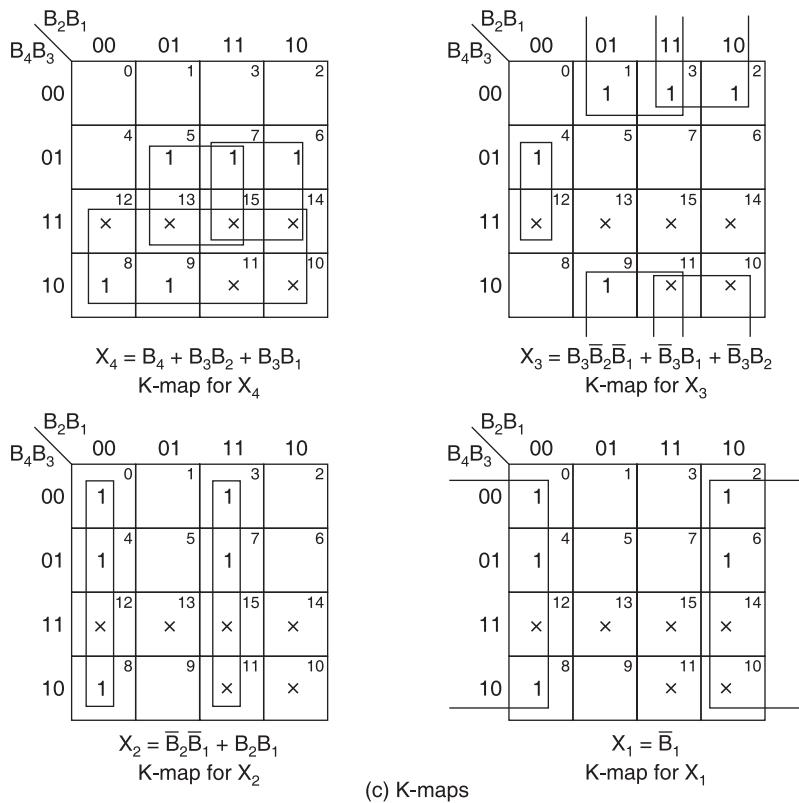


Figure 7.35 4-bit BCD-to-XS-3 code converter.

The expressions for the outputs  $X_4$ ,  $X_3$ ,  $X_2$  and  $X_1$  are shown in Figure 7.35b. Drawing K-maps for the outputs  $X_4$ ,  $X_3$ ,  $X_2$  and  $X_1$  in terms of the inputs  $B_4$ ,  $B_3$ ,  $B_2$ , and  $B_1$  and simplifying them, as shown in Figure 7.35c the minimal expressions for  $X_4$ ,  $X_3$ ,  $X_2$ , and  $X_1$  are as shown in Figure 7.35b. A logic diagram can be drawn based on those minimal expressions.

### 7.16.5 Design of a BCD-to-Gray Code Converter

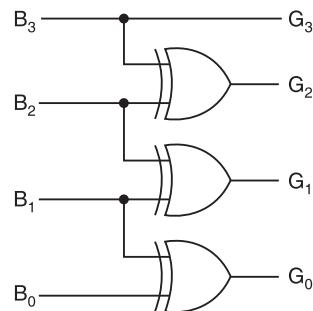
The BCD to Gray code conversion table is shown in Figure 7.36a. For a 4-bit BCD code minterms 10, 11, 12, 13, 14, and 15 are don't cares. So the expressions for the Gray code outputs in terms of BCD inputs are as follows:

$$\begin{aligned} G_3 &= \Sigma m(8, 9) + d(10, 11, 12, 13, 14, 15) \\ G_2 &= \Sigma m(4, 5, 6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15) \\ G_1 &= \Sigma m(2, 3, 4, 5) + d(10, 11, 12, 13, 14, 15) \\ G_0 &= \Sigma m(1, 2, 5, 6, 9) + d(10, 11, 12, 13, 14, 15) \end{aligned}$$

The K-maps for  $G_3$ ,  $G_2$ ,  $G_1$ , and  $G_0$ , their minimization, and the minimal expressions obtained from them are shown in Figure 7.37. The logic diagram of the BCD to Gray code converter based on those minimal expressions is shown in Figure 7.36b.

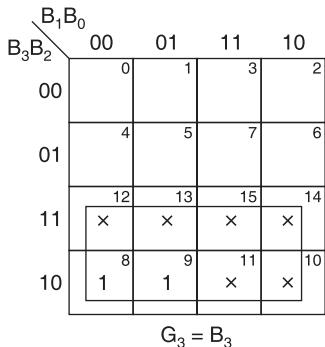
BCD code				Gray code			
B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	G <sub>3</sub>	G <sub>2</sub>	G <sub>1</sub>	G <sub>0</sub>
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1

(a) BCD-to-Gray code conversion table

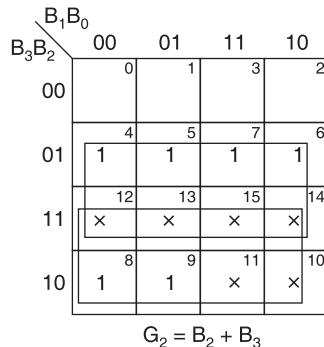


(b) Logic diagram

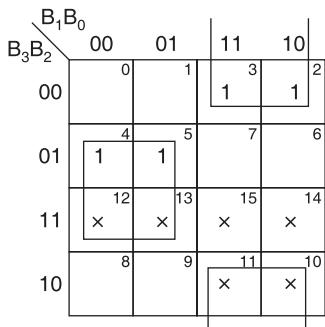
Figure 7.36 BCD-to-Gray code converter.



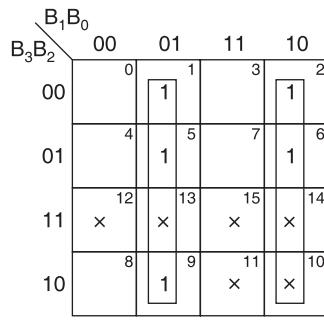
$$G_3 = B_3$$



$$G_2 = B_2 + B_3$$



$$G_1 = B_2 \bar{B}_1 + \bar{B}_2 B_1 = B_2 \oplus B_1$$



$$G_0 = \bar{B}_1 B_0 + B_1 \cdot \bar{B}_0 = B_1 \oplus B_0$$

Figure 7.37 K-maps for a BCD-to-Gray code converter.

#### 7.16.6 Design of an SOP Circuit to Detect the Decimal Numbers 5 through 12 in a 4-bit Gray Code Input

The input to the SOP circuit is a 4-bit Gray code. Let the input Gray code be ABCD. There are 16 possible combinations of 4-bit Gray code. All of them are valid and hence there are no don't cares.

The truth table of the SOP circuit is shown in Figure 7.38a. Looking at the truth table of the SOP circuit, we observe that the output is 1 for the input combinations corresponding to minterms 7, 5, 4, 12, 13, 15, 14, and 10 (i.e. corresponding to the Gray code of decimal numbers 5, 6, 7, 8, 9, 10, 11 and 12). So the expression for the output is

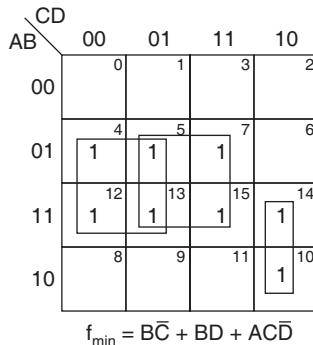
$$f = \Sigma m(7, 5, 4, 12, 13, 15, 14, 10) = \Sigma m(4, 5, 7, 10, 12, 13, 14, 15)$$

The K-map for f, its minimization, the minimal expression obtained from it and its realization in NAND logic are shown in Figures 7.38b and c respectively.

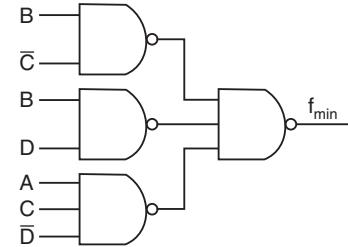
$$f_{\min} = B\bar{C} + BD + A\bar{C}\bar{D} = \overline{\overline{B}\bar{C}} \cdot \overline{BD} \cdot \overline{\overline{A}\bar{C}\bar{D}}$$

Decimal number	4-bit Gray code				Output f
	A	B	C	D	
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	1	0
3	0	0	1	0	0
4	0	1	1	0	0
5	0	1	1	1	1
6	0	1	0	1	1
7	0	1	0	0	1
8	1	1	0	0	1
9	1	1	0	1	1
10	1	1	1	1	1
11	1	1	1	0	1
12	1	0	1	0	1
13	1	0	1	1	0
14	1	0	0	1	0
15	1	0	0	0	0

(a) Truth table



(b) K-map



(c) NAND logic

**Figure 7.38** Truth table, K-map and logic diagram for the SOP circuit.

#### 7.16.7 Design of an SOP Circuit to Detect the Decimal Numbers 0, 2, 4, 6, and 8 in a 4-bit 5211 BCD Code Input

The input to the SOP circuit is a 5211 BCD code. Let it be ABCD. It is a 4-bit input. Therefore, there are 16 possible combinations of inputs, out of which only the 10 combinations shown in the truth table of Figure 7.39a are used to code the decimal digits in 5211 code. The remaining 6 combinations 0010, 0100, 0110, 1001, 1011, and 1101 are invalid. So, the corresponding outputs are don't cares (i.e. minterms 2, 4, 6, 9, 11, and 13 are don't cares). Looking at the truth table of the SOP circuit shown in Figure 7.39a we observe that the output is 1 for the input combinations corresponding to minterms 0, 3, 7, 10, 14 (i.e. corresponding to 5211 code of decimal numbers 0, 2, 4, 6, and 8).

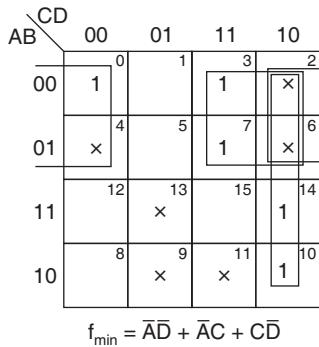
So the problem may be stated as

$$F = \Sigma m(0, 3, 7, 10, 14) + d(2, 4, 6, 9, 11, 13)$$

The K-map, its minimization, the minimal expression obtained from it and the realization of the minimal expression in NAND logic are shown in Figure 7.39.

$$f_{\min} = \overline{AD} + \overline{AC} + \overline{CD} = \overline{\overline{AD}} \cdot \overline{\overline{AC}} \cdot \overline{\overline{CD}}$$

Decimal number	5211 code				Output f
	A	B	C	D	
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	1	1
3	0	1	0	1	0
4	0	1	1	1	1
5	1	0	0	0	0
6	1	0	1	0	1
7	1	1	0	0	0
8	1	1	1	0	1
9	1	1	1	1	0



(a) Truth table

(b) K-map

(c) Logic diagram

Figure 7.39 Truth table, K-map and logic diagram for the SOP circuit.

### 7.16.8 Design of a Combinational Circuit to Produce the 2's Complement of a 4-bit Binary Number

The 4-bit binary input combinations and their 2's complement versions are shown in the truth table in Figure 7.40a. From the truth table, the expressions for outputs E, F, G, and H are:

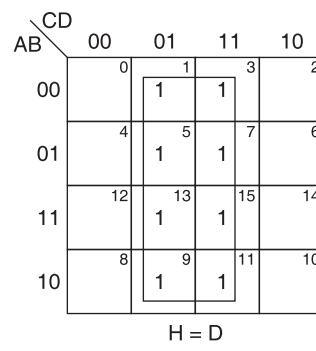
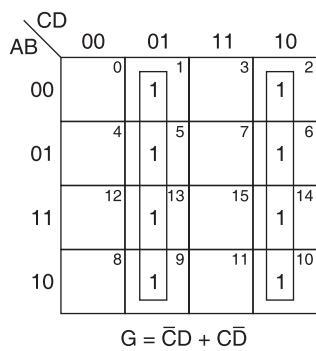
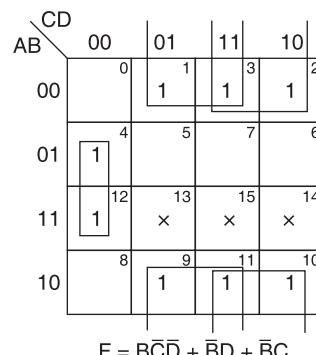
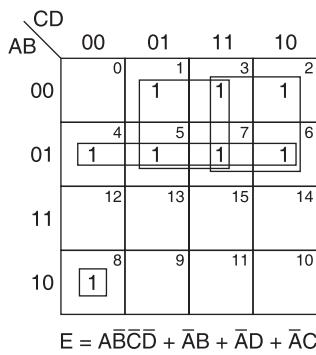
$$\begin{aligned} E &= \Sigma m(1, 2, 3, 4, 5, 6, 7, 8), & F &= \Sigma m(1, 2, 3, 4, 9, 10, 11, 12), \\ G &= \Sigma m(1, 2, 5, 6, 9, 10, 13, 14), & H &= \Sigma m(1, 3, 5, 7, 9, 11, 13, 15) \end{aligned}$$

The K-maps for the output expressions, their minimization and the minimal expressions obtained from them are shown in Figure 7.40b.

A	B	C	D	Input				Output			
				E	F	G	H				
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1
0	0	1	0	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0	1	1
0	1	0	0	1	1	0	0	0	0	0	0
0	1	0	1	1	0	1	0	1	1	1	1
0	1	1	0	1	0	0	1	0	1	0	0
0	1	1	1	1	0	0	0	0	0	1	1
1	0	0	0	0	1	0	1	1	1	1	1
1	0	0	1	0	0	1	0	1	1	0	0
1	0	1	1	0	0	1	0	0	1	0	1
1	1	0	0	0	0	1	0	1	0	0	0
1	1	0	1	0	0	0	1	1	1	1	1
1	1	1	0	0	0	0	1	0	1	0	0
1	1	1	1	0	0	0	0	0	0	1	0

(a) Conversion table

Figure 7.40 Conversion table and K-maps for the circuit (Contd.)



(b) K-maps

**Figure 7.40** Conversion table and K-maps for the circuit.

After simplification the minimal expressions are

$$E = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B} + \bar{A}\bar{D} + \bar{A}\bar{C}, \quad F = \bar{B}\bar{C}\bar{D} + \bar{B}\bar{D} + \bar{B}\bar{C}, \quad G = \bar{C}\bar{D} + C\bar{D}, \quad H = D$$

A logic circuit can be drawn based on these minimal expressions.

### 7.16.9 Design of a Circuit to Detect the Decimal Numbers 0, 1, 4, 6, 7, and 8 in a 4-bit XS-3 Code Input

The input to the circuit is a 4-bit Excess-3 code. There are 16 possible combinations of 4-bit inputs. Out of these, 10 combinations shown in the truth table in Figure 7.41a represent valid Excess-3 code. The remaining 6 combinations (0000, 0001, 0010, 1101, 1110, 1111) are invalid. Hence, the corresponding outputs are don't cares. Looking at the truth table of the SOP circuit shown in Figure 7.41a we observe that the output is 1 for the input combinations corresponding to minterms 3, 4, 7, 9, 10, and 11 (i.e. corresponding to the XS-3 code of decimal numbers 0, 1, 4, 6, 7, and 8).

So the Boolean expression for the output of the circuit in terms of minterms is

$$f = \sum m(3, 4, 7, 9, 10, 11) + d(0, 1, 2, 13, 14, 15)$$

For a minimal design, we find the minimal SOP form, the minimal POS form and then take the minimal of these two minimals.

The K-maps in SOP and POS forms, their minimization, the minimal expressions obtained from them and the actual minimal circuit in NOR logic are shown in Figures 7.41b and c respectively. SOP minimal is

$$f_{\min} = CD + AD + AC + \overline{ACD}$$

POS minimal is

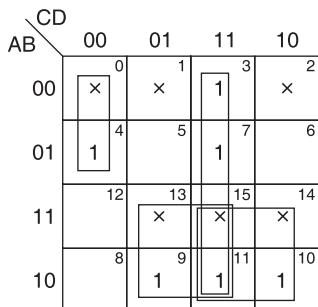
$$f_{\min} = (\overline{A} + C + D)(\overline{B} + C + \overline{D})(\overline{B} + \overline{C} + D)$$

For minimal circuit in NOR logic the minimal expression is written as

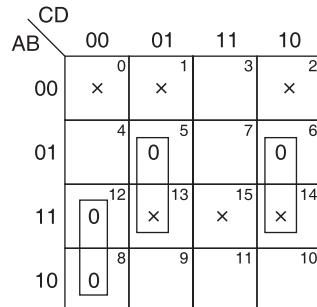
$$f_{\min} = (\overline{A} + C + D)(\overline{B} + C + \overline{D})(\overline{B} + \overline{C} + D) = \overline{(\overline{A} + C + D)} + \overline{(\overline{B} + C + \overline{D})} + \overline{(\overline{B} + \overline{C} + D)}$$

Decimal number	4-bit excess-3				Output f
	A	B	C	D	
0	0	0	1	1	1
1	0	1	0	0	1
2	0	1	0	1	0
3	0	1	1	0	0
4	0	1	1	1	1
5	1	0	0	0	0
6	1	0	0	1	1
7	1	0	1	0	1
8	1	0	1	1	1
9	1	1	0	0	0

(a) Truth table

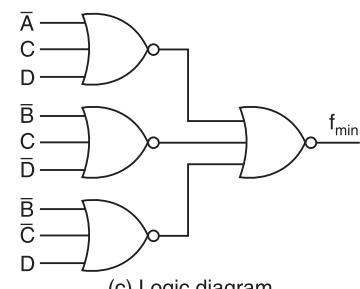


$$f_{\min} = CD + AD + AC + \overline{ACD}$$



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

(b) K-maps



(c) Logic diagram

Figure 7.41 Truth table, K-maps and logic diagram for the circuit.

**EXAMPLE 7.3** Design a minimal circuit to produce an output of 1, when its input is a 2421 code representing an even decimal number less than 10.

**Solution**

The input to the circuit is a 4-bit 2421 code. There are 16 possible combinations of 4-bit input, out of which the 10 combinations shown in the truth table are valid for the 2421 code, and the remaining 6 combinations 0101, 0110, 0111, 1000, 1001 and 1010 are invalid and hence the corresponding outputs are don't cares. Looking at the truth table of the SOP circuit shown in Figure 7.42a we observe that the output is 1 for the input combinations corresponding to minterms 0, 2, 4, 12, and 14 (i.e. corresponding to the 2421 code of decimal numbers 0, 2, 4, 6, and 8).

So the Boolean expression for the output is

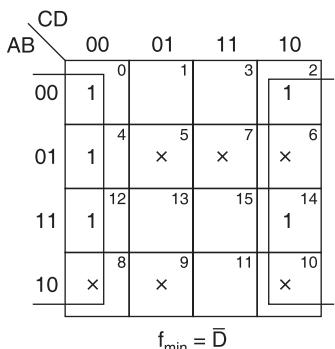
$$f = \Sigma m(0, 2, 4, 12, 14) + d(5, 6, 7, 8, 9, 10)$$

The truth table, the K-maps in SOP and POS forms, their minimization, the minimal expressions obtained from them and the logic diagram based on the real minimal expression are shown in Figures 7.42a, b, and c respectively. In this case, both the SOP and POS minimal forms are the same.

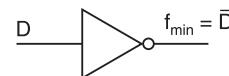
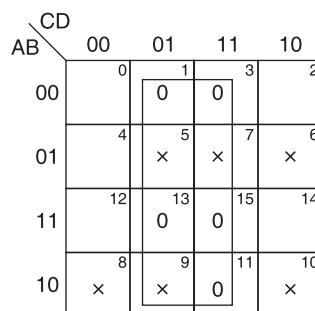
The minimal expression is  $f_{\min} = \bar{D}$

Decimal number	2421 code				Output f
	A	B	C	D	
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	1	0	1	1	0
6	1	1	0	0	1
7	1	1	0	1	0
8	1	1	1	0	1
9	1	1	1	1	0

(a) Truth table



(b) K-maps



(c) Logic diagram

**Figure 7.42** Example 7.3.

**EXAMPLE 7.4** Design a combinational circuit that accepts a 3-bit BCD number and generates an output binary number equal to the square of the input number.

#### Solution

The square of a 3-bit number is a 6-bit number. The truth table of the combinational circuit and the minimal expressions for the outputs obtained after algebraic simplification are shown in Figures 7.43a and b respectively. A logic diagram can be drawn based on the above expressions.

Inputs			Outputs					
A	B	C	X <sub>6</sub>	X <sub>5</sub>	X <sub>4</sub>	X <sub>3</sub>	X <sub>2</sub>	X <sub>1</sub>
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	0	1
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	0	0	1

(a) Truth table

$$X_1 = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC = C$$

$$X_2 = 0$$

$$X_3 = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} = B\bar{C}$$

$$X_4 = \bar{A}BC + A\bar{B}C = C(A \oplus B)$$

$$X_5 = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + ABC = A(\bar{B} + C)$$

$$X_6 = ABC + ABC = AB$$

(b) Minimal expressions

Figure 7.43 Example 7.4.

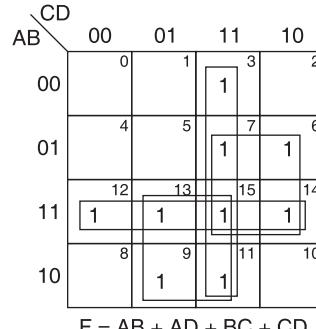
**EXAMPLE 7.5** Design a logic circuit with 4 inputs A, B, C, D that will produce output ‘1’ only whenever two adjacent input variables are 1s. A and D are also to be treated as adjacent. Implement it using universal logic.

### Solution

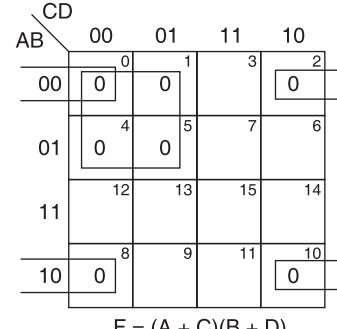
The truth table of the logic circuit is shown in Figure 7.44a.

Input				Output
A	B	C	D	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

(a) Truth table

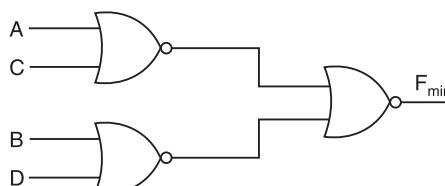


$$F = AB + AD + BC + CD$$



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

(b) K-maps



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

(c) Logic diagram

Figure 7.44 Example 7.5.

From the truth table, we observe that the expression for the output is given by:

In SOP form

$$F = \Sigma m(3, 6, 7, 9, 11, 12, 13, 14, 15)$$

In POS form

$$F = \Pi M(0, 1, 2, 4, 5, 8, 10)$$

The K-maps in SOP and POS forms, their minimization and the minimal expressions obtained from them are shown in Figure 7.44b.

SOP minimal is

$$F_{\min} = AB + AD + BC + CD$$

POS minimal is

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

The SOP form requires 12 gate inputs whereas the POS form requires only 6 gate inputs for realization. To realize in universal logic the minimal expression can be written as

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

A logic diagram based on the real minimal expression can be drawn as shown in Figure 7.44c.

**EXAMPLE 7.6**  $A_8 A_4 A_2 A_1$  is an 8421 BCD input to a logic circuit whose output is a 1 when  $A_8 = 0$ ,  $A_4 = 0$  and  $A_2 = 1$ , or when  $A_8 = 0$  and  $A_4 = 1$ . Design the simplest possible logic circuit.

### Solution

Denote the non-complemented variable by a 1 and the complemented variable by a 0. Based on the statement of the problem, the expression for the output is

$$f = \overline{A}_8 \overline{A}_4 A_2 + \overline{A}_8 A_4$$

Now expand it to get it in the standard SOP form. Therefore,

$$\begin{aligned} f &= \overline{A}_8 \overline{A}_4 A_2 (A_1 + \overline{A}_1) + \overline{A}_8 A_4 (A_2 + \overline{A}_2)(A_1 + \overline{A}_1) \\ &= 001X + 01XX \\ &= 0010 + 0011 + 0100 + 0101 + 0110 + 0111 \\ &= \Sigma m(2, 3, 4, 5, 6, 7) \end{aligned}$$

The input is a 4-bit BCD. So there are 6 invalid combinations (corresponding to minterms 10, 11, 12, 13, 14, 15) and the corresponding outputs are don't cares. So, the Boolean expression is

$$f = \Sigma m(2, 3, 4, 5, 6, 7) + d(10, 11, 12, 13, 14, 15)$$

Obtain the minimal expressions in SOP and POS forms and implement the minimal of these minimals. The K-maps in SOP and POS forms, their minimization, the minimal expressions obtained from each, and the logic diagram based on the minimal expression are shown in Figures 7.45a and b.

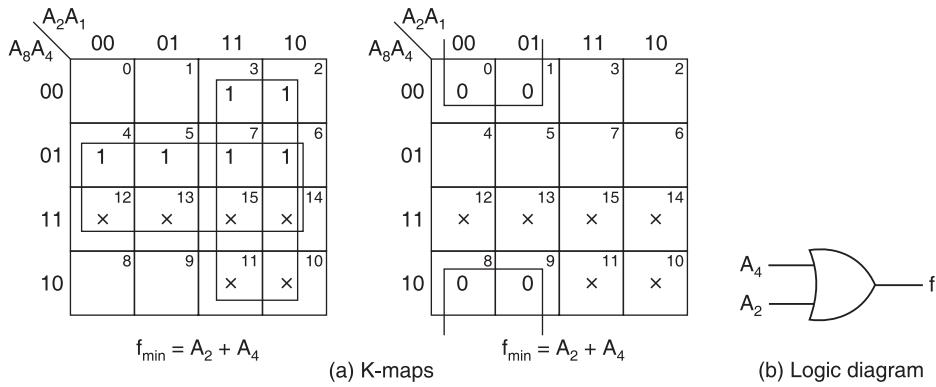


Figure 7.45 Example 7.6.

**EXAMPLE 7.7** Design each of the following circuits that can be built using AOI logic and outputs a 1 when:

- (a) A 4-bit hexadecimal input is an odd number from 0 to 9.
- (b) A 4-bit BCD code translated to a number that uses the upper right segment of a seven-segment display.

**Solution**

(a) The output is a 1 when the input is a 4-bit hexadecimal odd number from 0 to 9. There are 16 possible combinations of inputs, and all are valid. The output is a 1 only for the input combinations 0001, 0011, 0101, 0111, and 1001. For all other combinations of inputs, the output is a 0. The problem may thus be stated as  $f = \sum m(1, 3, 5, 7, 9)$ . The SOP K-map, its minimization, the minimal expression obtained from it, and the realization of the minimal expression in AOI logic are shown in Figures 7.46a and b.

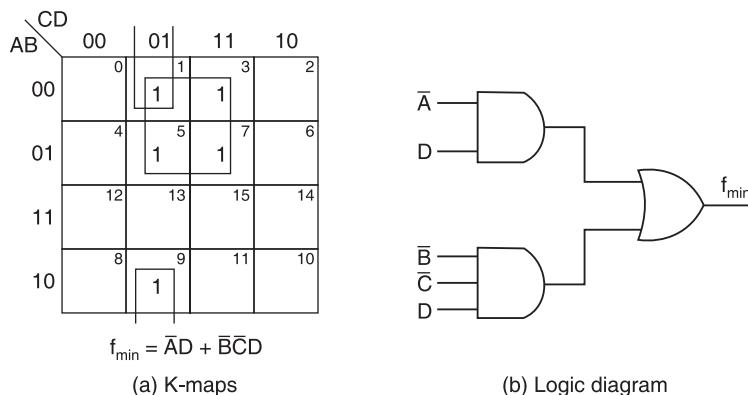


Figure 7.46 Example 7.7a.

- (b) We see from Figure 7.47a that display of digits 0, 1, 2, 3, 4, 7, 8, and 9 requires the upper-right segment of the seven-segment display. Since the input is a 4-bit BCD, inputs 1010

through 1111 are invalid, and therefore, the corresponding outputs are don't cares. The problem may be stated as

$$f = \Sigma m(0, 1, 2, 3, 4, 7, 8, 9) + d(10, 11, 12, 13, 14, 15)$$

The K-map, its minimization, the minimal expression obtained from it and the realization of the minimal expression using AOI logic are shown in Figures 7.47b and c respectively.

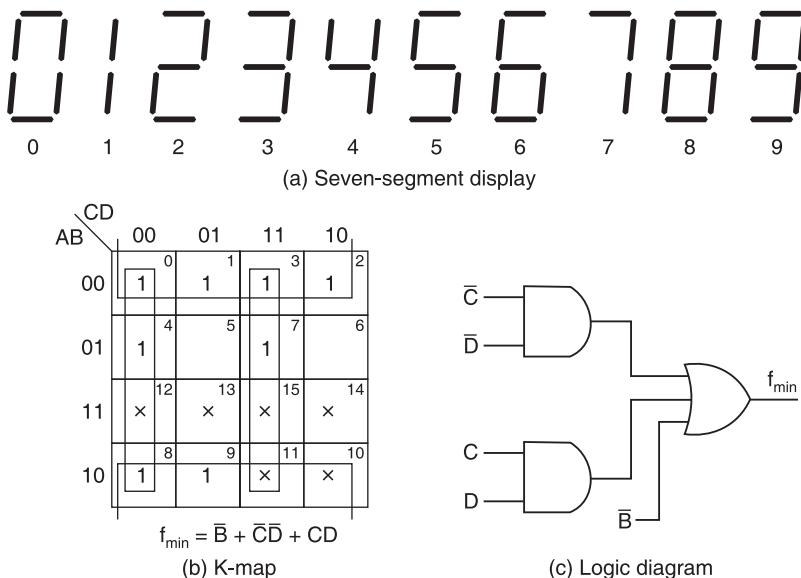


Figure 7.47 Example 7.7b.

### 7.16.10 Code Converters Using ICs

Figure 7.48a shows the use of 74184, a ROM device programmed as a BCD-to-binary converter. Figure 7.48b shows the use of 74185, a ROM device programmed as a binary-to-BCD converter.

Figure 7.49 shows the use of ICs for obtaining 9's and 10's complement of a number.

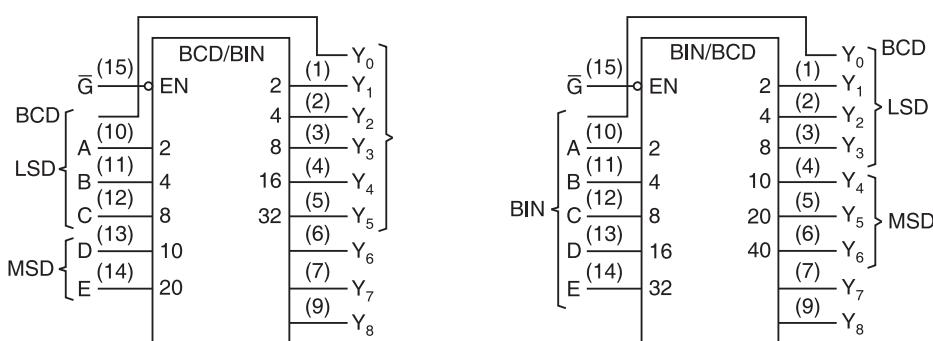


Figure 7.48 BCD-to-binary and binary-to-BCD conversion using ICs.

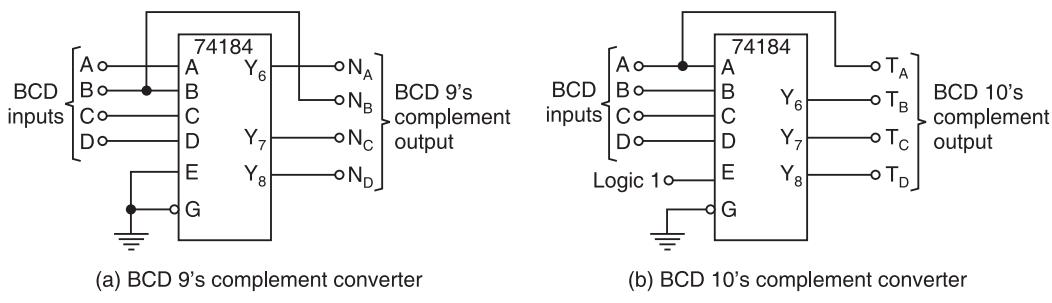


Figure 7.49 9's and 10's complement converters using ICs.

## 7.17 PARITY BIT GENERATORS/CHECKERS

Exclusive-OR functions are very useful in systems requiring error detection and correction codes. Binary data, when transmitted and processed, is susceptible to noise that can alter its 1s to 0s and 0s to 1s. To detect such errors, an additional bit called the *parity bit* is added to the data bits and the word containing the data bits and the parity bit is transmitted. At the receiving end the number of 1s in the word received are counted and the error, if any, is detected. This parity check, however, detects only single bit errors.

The circuit that generates the parity bit in the transmitter is called a parity generator. The circuit that checks the parity in the receiver is called a parity checker.

A parity bit, a 0 or a 1 is attached to the data bits such that the total number of 1s in the word is even for even parity and odd for odd parity. The parity bit can be attached to the code group either at the beginning or at the end depending on system design. A given system operates with either even or odd parity but not both. So, a word always contains either an even or an odd number of 1s.

At the receiving end, if the word received has an even number of 1s in the odd parity system or an odd number of 1s in the even parity system, it implies that an error has occurred.

In order to check or generate the proper parity bit in a given code word, the basic principle used is, “the modulo sum of an even number of 1s is always a 0 and the modulo sum of an odd number of 1s is always a 1”. Therefore, in order to check for an error, all the bits in the received word are added. If the modulo sum is a 0 for an odd parity system or a 1 for an even parity system, an error is detected.

To generate an even parity bit, the four data bits are added using three X-OR gates. The sum bit will be the parity bit. Figure 7.50a shows the logic diagram of an even parity generator.

To generate an odd parity bit, the four data bits are added using three X-OR gates and the sum bit is inverted. Figure 7.50b shows the logic diagram of an odd parity generator. Figures 7.50c and d show an even bit parity checker and an odd parity checker, respectively. Also, Figure 7.50e shows the logic symbol of IC 74180, a 9-bit parity generator/checker. Figure 7.50f gives the truth table operation of this IC. This device can be used to check for odd or even parity in a 9-bit code (8 data bits and one parity bit), or it can be used to generate a 9-bit odd or even parity code.

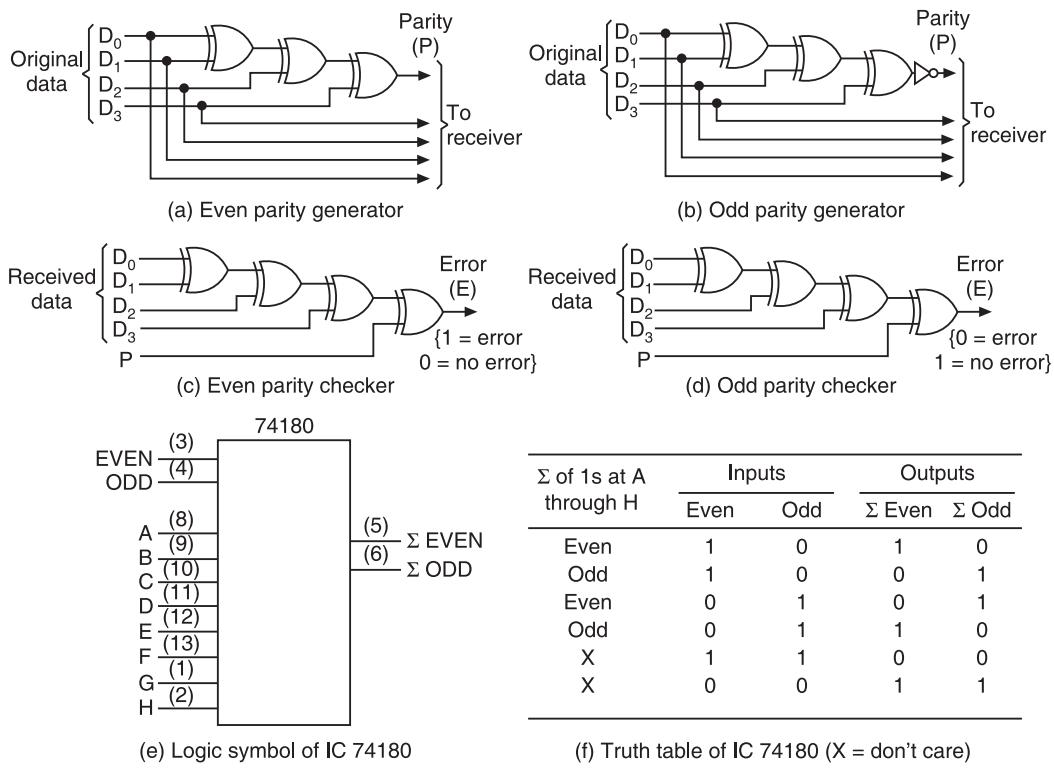


Figure 7.50 Parity bit generator/checker.

### 7.17.1 Parallel Parity Bit Generator for Hamming Code

Consider the 7-bit Hamming code ( $P_1 P_2 D_3 P_4 D_5 D_6 D_7$ ) used to correct single bit errors. Each word of the code contained 7 bits numbered 1–7. The bits 1, 2, and 4 are labelled as parity bits and called  $P_1$ ,  $P_2$ , and  $P_4$ . The message bits in the word are labelled  $D_3$ ,  $D_5$ ,  $D_6$  and  $D_7$ . Each parity bit is chosen in such a way that together with three other message bits it forms a 4-bit word with even parity, i.e.  $P_1$  is chosen as a 0 or 1 so that  $P_1 D_3 D_5 D_7$  must have even parity.  $P_2$  is chosen as a 0 or 1 so that  $P_2 D_3 D_6 D_7$  must have even parity.  $P_4$  is chosen as a 0 or 1 so that  $P_4 D_5 D_6 D_7$  must have even parity.

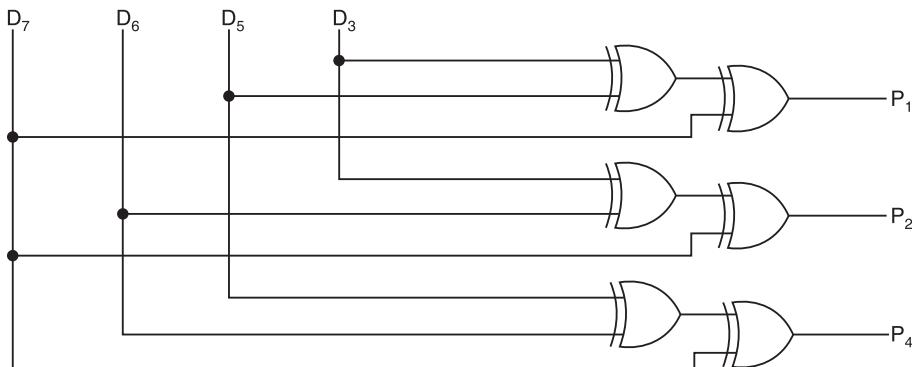


Figure 7.51 Parallel 3-bit parity generator for Hamming code.

Thus we would like to generate a parity bit with the three message bits given. The parity bit will be 1 whenever the number of 1's in the 3-message bits is odd, i.e. a single 1 or three 1s. So  $P_1$  is generated using data bits  $D_3, D_5, D_7$ .  $P_2$  is generated using data bits  $D_3, D_6, D_7$ .  $P_4$  is generated using data bits  $D_5, D_6, D_7$ . So two X-OR gates are to be used to generate each parity bit. The logic diagram of a parallel parity bit generator is shown in Figure 7.51.

### 7.17.2 Design of an Even Parity Bit Generator for a 4-bit Input

Let the 4-bit input be A B C D. For even parity, a parity bit 1 is added such that the total number of 1s in the 4-bit input and the parity bit together is even. The truth table, the K-map, and the logic diagram for even parity bit generator are shown in Figures 7.52a, b, and c respectively.

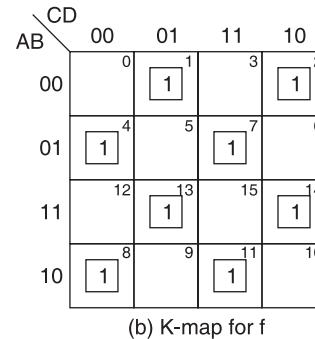
From the K-map we see that no minimization is possible. From the truth table, we observe that the parity bit is a 1, only if the total number of 1s in the four data bits is odd. Therefore, the parity bit is the modulo sum of the four data bits. Hence,

$$f = A \oplus B \oplus C \oplus D$$

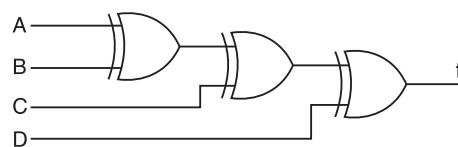
Therefore, three X-OR gates are required to realize the above expression as shown in Figure 7.52c. It can also be realized by using twelve 2-input NAND gates.

4-bit data input				Output parity bit (f)
A	B	C	D	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

(a) Truth table



(b) K-map for f



(c) Logic diagram

**Figure 7.52** Even parity bit generator.

From the K-map, the output is

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

### 7.17.3 Design of an Odd Parity Bit Generator for a 4-bit Input

An odd parity bit generator outputs a 1, when the number of 1s in the data bits is even, so that the total number of 1s in the data bits and the parity bit together is odd.

The expression for the even parity generator is

$$f = A \oplus B \oplus C \oplus D$$

Since odd parity is the complement of even parity, we get for odd parity generator

$$\bar{f} = \overline{A \oplus B \oplus C \oplus D}$$

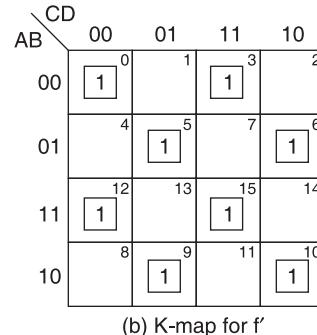
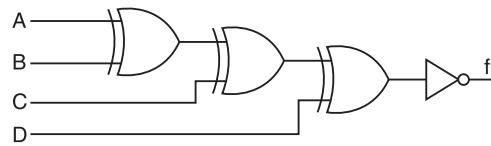
So, we put an inverter at the output of an even parity bit generator. The same result is directly obtained as follows.

The truth table, the K-map, and the logic diagram for the odd parity generator are shown in Figures 7.53a, b, and c, respectively.

From the K-map, we see that no minimization is possible. If the expression for the parity bit is implemented as it is, 40 gate inputs are required. To reduce the cost, the expression may be manipulated in terms of X-OR gates and implemented.

4-bit data input				Output parity bit $\bar{f}$
A	B	C	D	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

(a) Truth table

(b) K-map for  $f'$ 

(c) Logic diagram

**Figure 7.53** Odd parity bit generator.

From the K-map, the output is

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

#### EXAMPLE 7.8

- (a) Make a 9-bit odd parity checker using a 74180 and an inverter.
- (b) Make a 10-bit even parity generator using a 74180 and an inverter.
- (c) Make a 16-bit even parity checker using two 74180s.

**Solution**

(a) 8 of the 9-bits are applied at A-H inputs and the ninth bit, I, is applied to the ODD input.  
The circuit is shown in Figure 7.54a.

(b) The 9-bit word consisting of A through I is converted to a 10-bit word with even parity.  
The circuit is shown in Figure 7.54b.

(c) The 16-bit even parity checker is shown in Figure 7.54c.

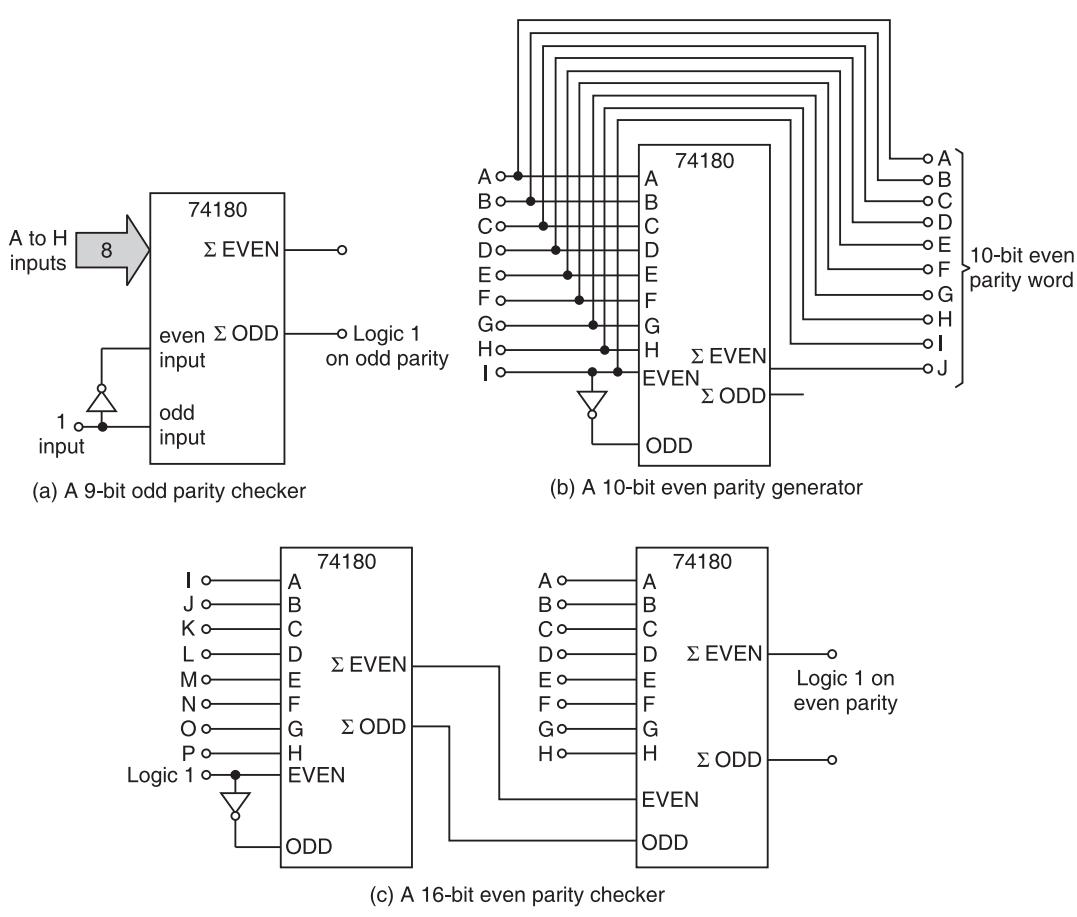


Figure 7.54 Example 7.8: Logic circuits.

## 7.18 COMPARATORS

A comparator is a logic circuit used to compare the magnitudes of two binary numbers. Depending on the design, it may either simply provide an output that is active (goes HIGH for example) when the two numbers are equal, or additionally provide outputs that signify which of the numbers is greater when equality does not hold.

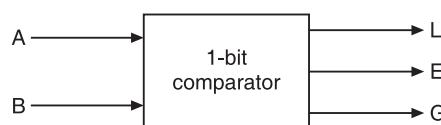
The X-NOR gate (coincidence gate) is a basic comparator, because its output is a 1 only if its two input bits are equal, i.e. the output is a 1 if and only if the input bits coincide.

Two binary numbers are equal, if and only if all their corresponding bits coincide. For example, two 4-bit binary numbers,  $A_3A_2A_1A_0$  and  $B_3B_2B_1B_0$  are equal, if and only if,  $A_3 = B_3$ ,  $A_2 = B_2$ ,  $A_1 = B_1$  and  $A_0 = B_0$ . Thus, equality holds when  $A_3$  coincides with  $B_3$ ,  $A_2$  coincides with  $B_2$ ,  $A_1$  coincides with  $B_1$ , and  $A_0$  coincides with  $B_0$ . The implementation of this logic,

$$\text{EQUALITY} = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$

is straightforward. It is obvious that this circuit can be expanded or compressed to accommodate binary numbers with any other number of bits.

The block diagram of a 1-bit comparator which can be used as a module for comparison of larger numbers is shown in Figure 7.55.



**Figure 7.55** Block diagram of a 1-bit comparator.

### 7.18.1 1-bit Magnitude Comparator

The logic for a 1-bit magnitude comparator: Let the 1-bit numbers be  $A = A_0$  and  $B = B_0$ . If  $A_0 = 1$  and  $B_0 = 0$ , then  $A > B$ .

Therefore,

$$A > B : G = A_0 \bar{B}_0$$

If  $A_0 = 0$  and  $B_0 = 1$ , then  $A < B$ .

Therefore,

$$A < B : L = \bar{A}_0 B_0$$

If  $A_0$  and  $B_0$  coincide, i.e.  $A_0 = B_0 = 0$  or if  $A_0 = B_0 = 1$ , then  $A = B$ .

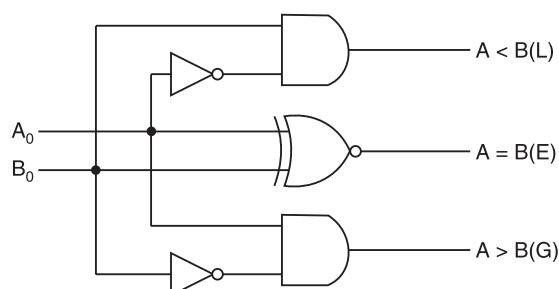
Therefore,

$$A = B : E = A_0 \odot B_0$$

The truth table and the logic diagram for the 1-bit comparator are shown in Figure 7.56. The logic expressions for  $G$ ,  $L$ , and  $E$  can also be obtained from the truth table.

$A_0$	$B_0$	$L$	$E$	$G$
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

(a) Truth table



(b) Logic diagram

**Figure 7.56** 1-bit comparator.

### 7.18.2 2-bit Magnitude Comparator

The logic for a 2-bit magnitude comparator: Let the two 2-bit numbers be  $A = A_1A_0$  and  $B = B_1B_0$ .

1. If  $A_1 = 1$  and  $B_1 = 0$ , then  $A > B$  or
2. If  $A_1$  and  $B_1$  coincide and  $A_0 = 1$  and  $B_0 = 0$ , then  $A > B$ . So the logic expression for  $A > B$  is

$$A > B : G = A_1\bar{B}_1 + (A_1 \odot B_1)A_0\bar{B}_0$$

1. If  $A_1 = 0$  and  $B_1 = 1$ , then  $A < B$  or

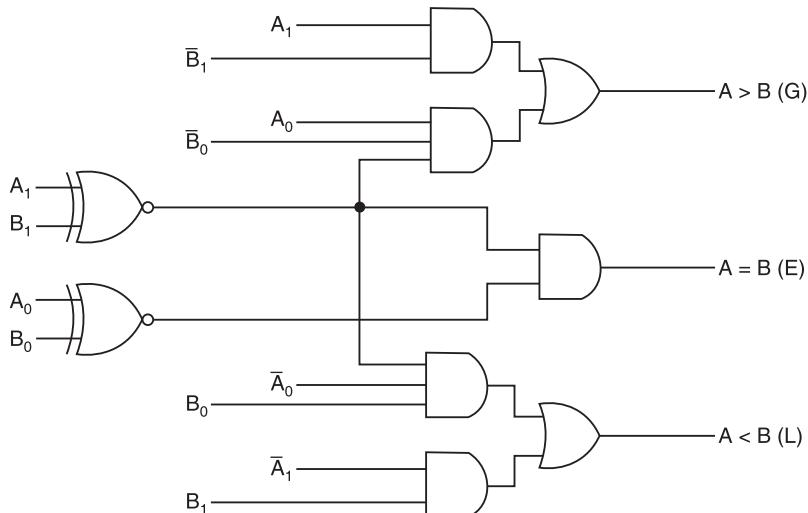
2. If  $A_1$  and  $B_1$  coincide and  $A_0 = 0$  and  $B_0 = 1$ , then  $A < B$ . So the expression for  $A < B$  is

$$A < B : L = \bar{A}_1B_1 + (A_1 \odot B_1)\bar{A}_0B_0$$

If  $A_1$  and  $B_1$  coincide and if  $A_0$  and  $B_0$  coincide then  $A = B$ . So the expression for  $A = B$  is

$$A = B : E = (A_1 \odot B_1)(A_0 \odot B_0)$$

The logic diagram for a 2-bit comparator is as shown in Figure 7.57.



**Figure 7.57** Logic diagram of a 2-bit magnitude comparator.

### 7.18.3 4-bit Magnitude Comparator

The logic for a 4-bit magnitude comparator: Let the two 4-bit numbers be  $A = A_3A_2A_1A_0$  and  $B = B_3B_2B_1B_0$ .

1. If  $A_3 = 1$  and  $B_3 = 0$ , then  $A > B$ . Or
2. If  $A_3$  and  $B_3$  coincide, and if  $A_2 = 1$  and  $B_2 = 0$ , then  $A > B$ . Or
3. If  $A_3$  and  $B_3$  coincide, and if  $A_2$  and  $B_2$  coincide, and if  $A_1 = 1$  and  $B_1 = 0$ , then  $A > B$ . Or
4. If  $A_3$  and  $B_3$  coincide, and if  $A_2$  and  $B_2$  coincide, and if  $A_1$  and  $B_1$  coincide, and if  $A_0 = 1$  and  $B_0 = 0$ , then  $A > B$ .

From these statements, we see that the logic expression for  $A > B$  can be written as

$$(A > B) : G = A_3\bar{B}_3 + (A_3 \odot B_3)A_2\bar{B}_2 + (A_3 \odot B_3)(A_2 \odot B_2)A_1\bar{B}_1 \\ + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)A_0\bar{B}_0$$

Similarly, the logic expression for  $A < B$  can be written as

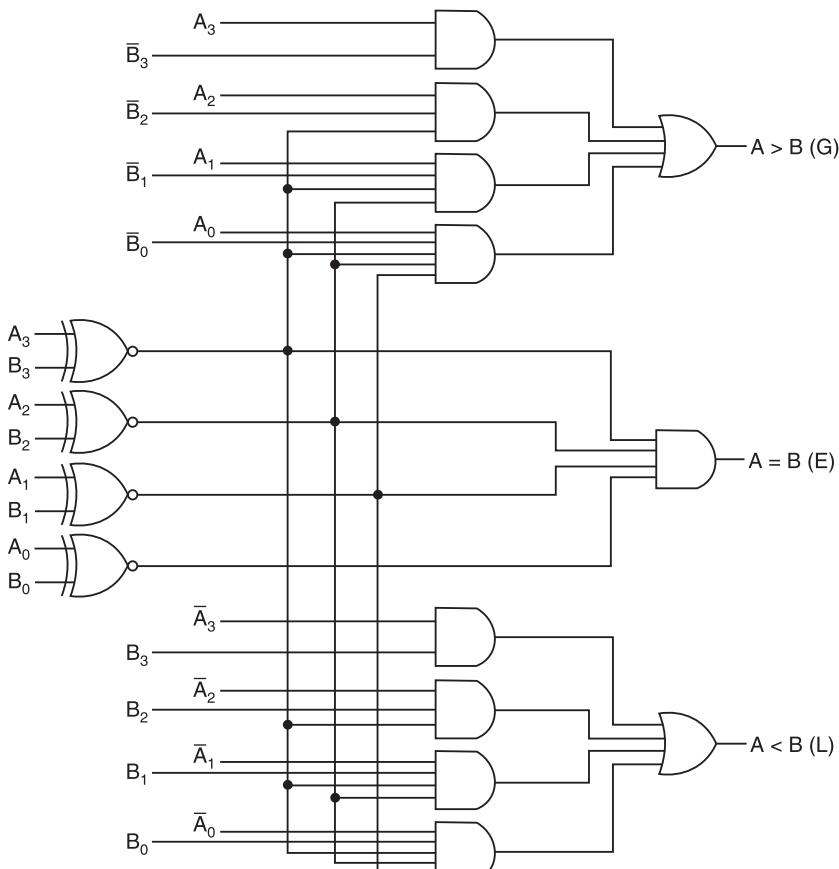
$$(A < B): L = \bar{A}_3B_3 + (A_3 \odot B_3)\bar{A}_2B_2 + (A_3 \odot B_3)(A_2 \odot B_2)\bar{A}_1B_1 \\ + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)\bar{A}_0B_0$$

If  $A_3$  and  $B_3$  coincide and if  $A_2$  and  $B_2$  coincide and if  $A_1$  and  $B_1$  coincide and if  $A_0$  and  $B_0$  coincide, then  $A = B$ .

So the expression for  $A = B$  can be written as

$$(A = B): E = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$

Figure 7.58 shows the logic diagram of a comparator that implements the logic we have described. Note that, it provides three active-HIGH outputs:  $A > B$ ,  $A < B$ , and  $A = B$ .

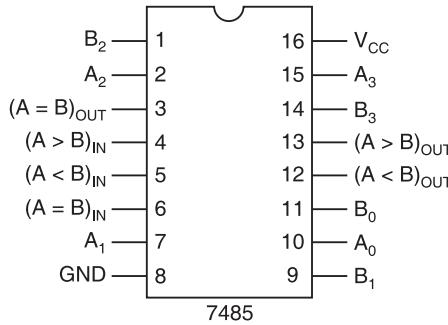


**Figure 7.58** Logic diagram of a 4-bit magnitude comparator.

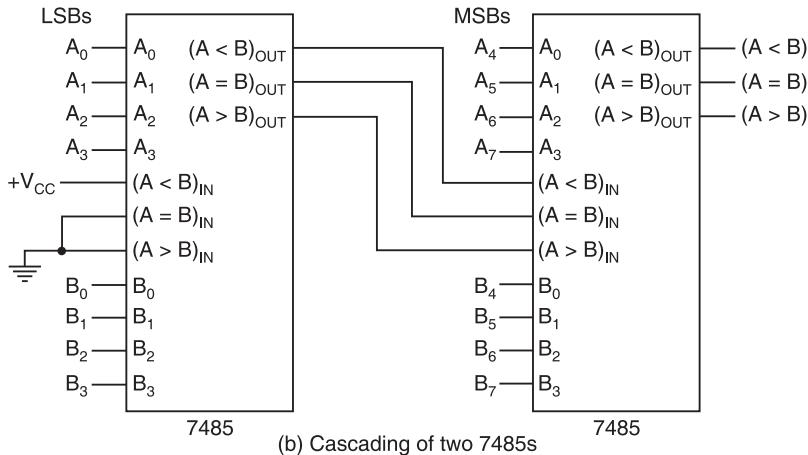
## 7.19 IC COMPARATOR

Figure 7.59a shows the pin diagram of IC 7485, a 4-bit comparator. Pins labelled  $(A < B)_{IN}$ ,  $(A = B)_{IN}$ , and  $(A > B)_{IN}$  are used for cascading. Figure 7.59b shows how two 4-bit comparators

are cascaded to perform 8-bit comparisons. The  $(A < B)_{OUT}$ ,  $(A = B)_{OUT}$  and  $(A > B)_{OUT}$  outputs from the lower order comparator used for the least significant 4 bits, are connected to the  $(A < B)_{IN}$ ,  $(A = B)_{IN}$ , and  $(A > B)_{IN}$  inputs of the higher-order comparator. Note that,  $(A < B)_{IN}$  input of the lower order comparator is connected to  $V_{CC}$ , and  $(A = B)_{IN}$  and  $(A > B)_{IN}$  inputs of the lower order comparator are connected to ground.



(a) Pin diagram of 7485



(b) Cascading of two 7485s

Figure 7.59 Pin diagram and cascading of 7485 4-bit comparators.

**EXAMPLE 7.9** Design a 5-bit comparator using a single 7485 and one gate.

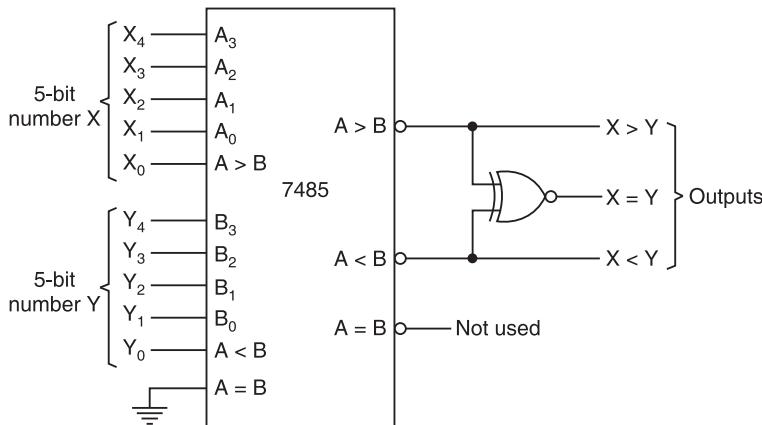
**Solution**

The circuit is shown in Figure 7.60. The two 5-bit numbers to be compared are  $X_4X_3X_2X_1X_0$  and  $Y_4Y_3Y_2Y_1Y_0$ .

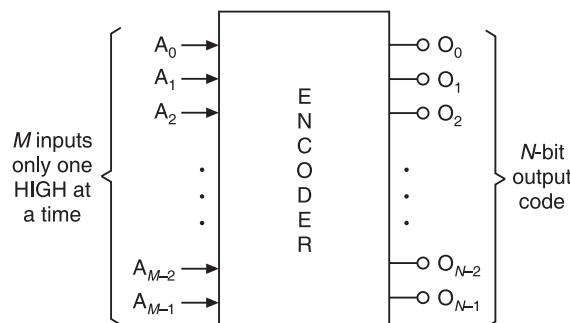
## 7.20 ENCODERS

An encoder is a device whose inputs are decimal digits and/or alphabetic characters and whose outputs are the coded representation of those inputs, i.e. an encoder is a device which converts familiar numbers or symbols into coded format. In other words, an encoder may be said to be a combinational logic circuit that performs the ‘reverse’ operation of the decoder. The opposite of the decoding process is called encoding, i.e. encoding is a process of converting familiar numbers

or symbols into a coded format. An encoder has a number of input lines, only one of which is activated at a given time, and produces an  $N$ -bit output code depending on which input is activated. Figure 7.61 shows the block diagram of an encoder with  $M$  inputs and  $N$  outputs. Here the inputs are active HIGH, which means they are normally LOW.



**Figure 7.60** Example 7.9: Use of 7485 as a 5-bit comparator.



**Figure 7.61** Block diagram of encoder.

### 7.20.1 Octal-to-Binary Encoder

An octal-to-binary encoder (8-line to 3-line encoder) accepts 8 input lines and produces a 3-bit output code corresponding to the activated input. Figure 7.62 shows the truth table and the logic circuit for an octal-to-binary encoder with active HIGH inputs.

From the truth table, we see that  $A_2$  is a 1 if any of the digits  $D_4$  or  $D_5$  or  $D_6$  or  $D_7$  is a 1. Therefore,

$$A_2 = D_4 + D_5 + D_6 + D_7$$

Similarly,

$$A_1 = D_2 + D_3 + D_6 + D_7$$

and

$$A_0 = D_1 + D_3 + D_5 + D_7$$

We see that  $D_0$  is not present in any of the expressions. So,  $D_0$  is a don't care.

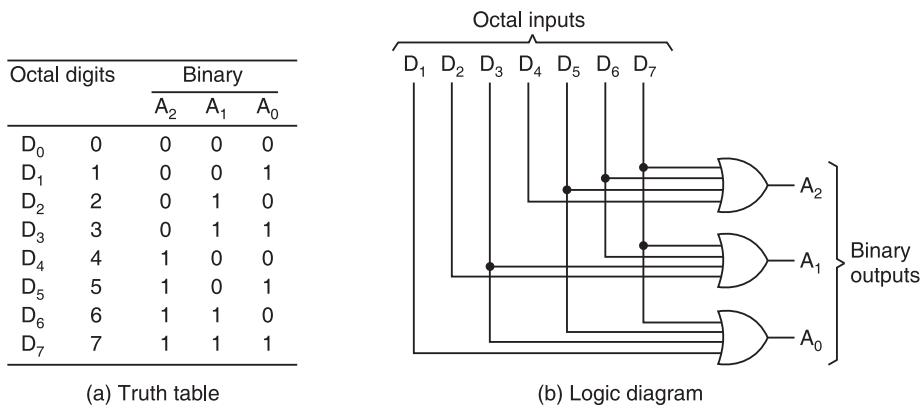


Figure 7.62 Octal-to-binary encoder.

### 7.20.2 Decimal-to-BCD Encoder

This type of encoder has 10 inputs—one for each decimal digit, and 4 outputs corresponding to the BCD code as shown in Figure 7.63a. This is a basic 10-line to 4-line encoder. The BCD code is

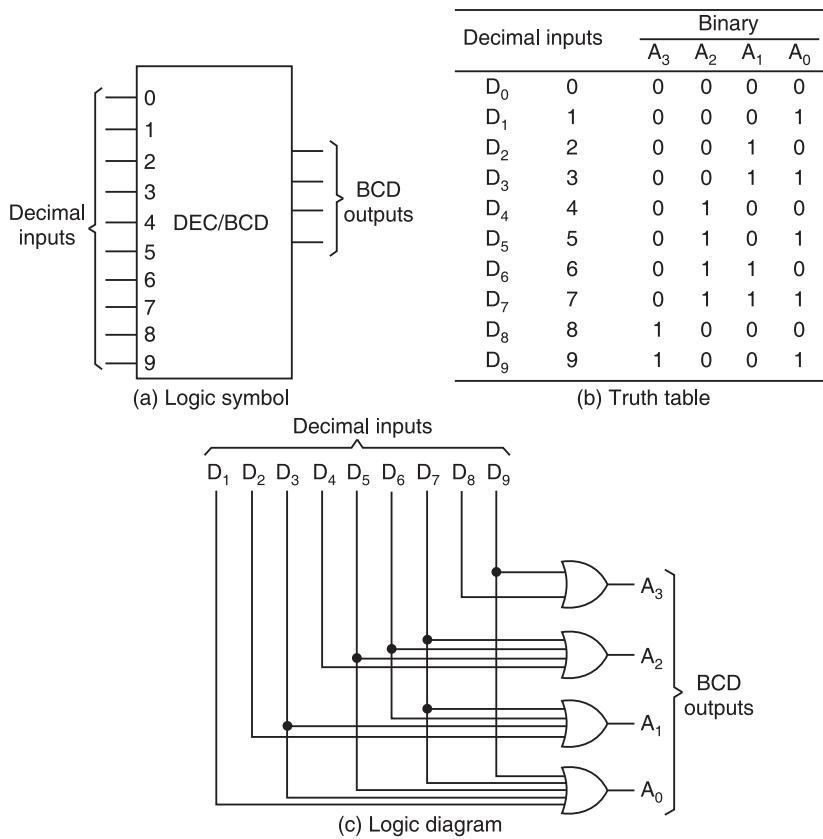


Figure 7.63 Decimal-to-BCD encoder.

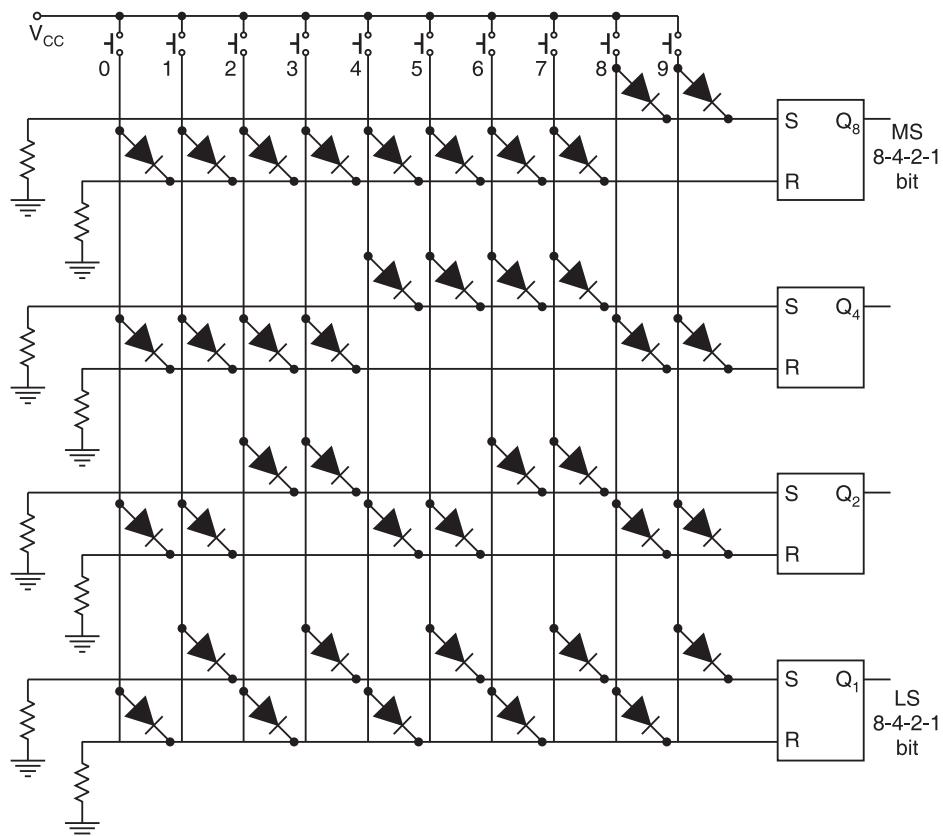
listed in the truth table (Figure 7.63b) and from this we can determine the relationships between each BCD bit and the decimal digits. There is no explicit input for a decimal 0. The BCD output is 0000 when the decimal inputs 1–9 are all 0.

The logic circuit of the encoder is shown in Figure 7.63c. From the table, we get

$$\begin{aligned} A_3 &= D_8 + D_9 \\ A_2 &= D_4 + D_5 + D_6 + D_7 \\ A_1 &= D_2 + D_3 + D_6 + D_7 \\ A_0 &= D_1 + D_3 + D_5 + D_7 + D_9 \end{aligned}$$

## 7.21 KEYBOARD ENCODERS

Figure 7.64 shows a typical keyboard encoder consisting of a diode matrix, used to encode the 10 decimal digits in 8-4-2-1 BCD.



**Figure 7.64** A keyboard encoder employing a diode matrix.

The S-R flip-flops are used to store the BCD output. When a key corresponding to one of the decimal digits is pressed, a positive voltage forward biases the selected diodes connected to the SET(S) and RESET(R) inputs of the flip-flops. The diodes are so arranged that each flip-flop sets

or resets, as necessary to produce the 4-bit code corresponding to the decimal digit. For example, when the key 7 is pressed, the diodes connected to the S inputs of  $Q_4$ ,  $Q_2$ , and  $Q_1$  are forward biased, as is that connected to the R input of  $Q_3$ . Thus, the output is 0111. Note that, the diode configuration at each S and R input is essentially a diode OR gate. Diode matrix encoders are found on printed circuit boards of many devices having a keyboard as the means of data entry.

## 7.22 PRIORITY ENCODERS

The encoders discussed so far will operate correctly, provided that one and only one decimal input is HIGH at any given time. In some practical systems, two or more decimal inputs may inadvertently become HIGH at the same time. For example, a person operating a keyboard might press a second key before releasing the first. Let us say he presses key 3 before releasing key 4. In such a case the output will be  $7_{10}$  (0111) instead of being  $4_{10}$  or  $3_{10}$ .

A priority encoder is a logic circuit that responds to just one input in accordance with some priority system, among all those that may be simultaneously HIGH. The most common priority system is based on the relative magnitudes of the inputs; whichever decimal input is the largest, is the one that is encoded. Thus, in the above example, a priority encoder would encode decimal 4 if both 3 and 4 are simultaneously HIGH.

In some practical applications, priority encoders may have several inputs that are routinely HIGH at the same time, and the principal function of the encoder in those cases is to select the input with the highest priority. This function is called *arbitration*. A common example is found in computer systems, where there are numerous input devices and several of which may attempt to supply data to the computer at the same time. A priority encoder is used to enable that input device which has the highest priority among those competing for access to the computer at the same time.

### 7.22.1 4-Input Priority Encoder

The truth table of a 4-input priority encoder is given in Figure 7.65a. In addition to the outputs A and B, the circuit has a third output designated by V. This is a valid bit indicator that is set to 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0. The other two outputs are not specified when V equals 0 and are specified as don't care conditions. According to the truth table, the higher the subscript number, the higher the priority of the input. Input  $D_3$  has the highest priority. So regardless of the values of other inputs, when this input is 1, the output for AB is 11 (binary 3).  $D_2$  has the next priority level. The output is 10 if  $D_2 = 1$  provided that  $D_3 = 0$  regardless of the values of the other two lower priority inputs. The output for  $D_1$  is generated only if higher priority inputs are 0, and so on down the priority levels.

From the truth table we observe that

$$A = D_3 + \bar{D}_3 D_2 = D_3 + D_2$$

and

$$B = D_3 + \bar{D}_3 \bar{D}_2 D_1 = D_3 + \bar{D}_2 D_1$$

$$V = D_3 + D_2 + D_1 + D_0$$

The condition for output V is an OR function of all the input variables.

The maps for simplifying outputs A and B and the corresponding logic diagram are shown in Figures 7.65b, c and d. Although the table has only five rows, when each  $\times$  in a row is first

replaced by 0 and then by 1, we obtain all 16 possible input combinations. The minterms for the two functions A and B are derived from the table as

$$A = \Sigma m(1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15)$$

$$B = \Sigma m(1, 3, 4, 5, 7, 9, 11, 12, 13, 15)$$

The same values of A and B mentioned above are obtained from the K-map.

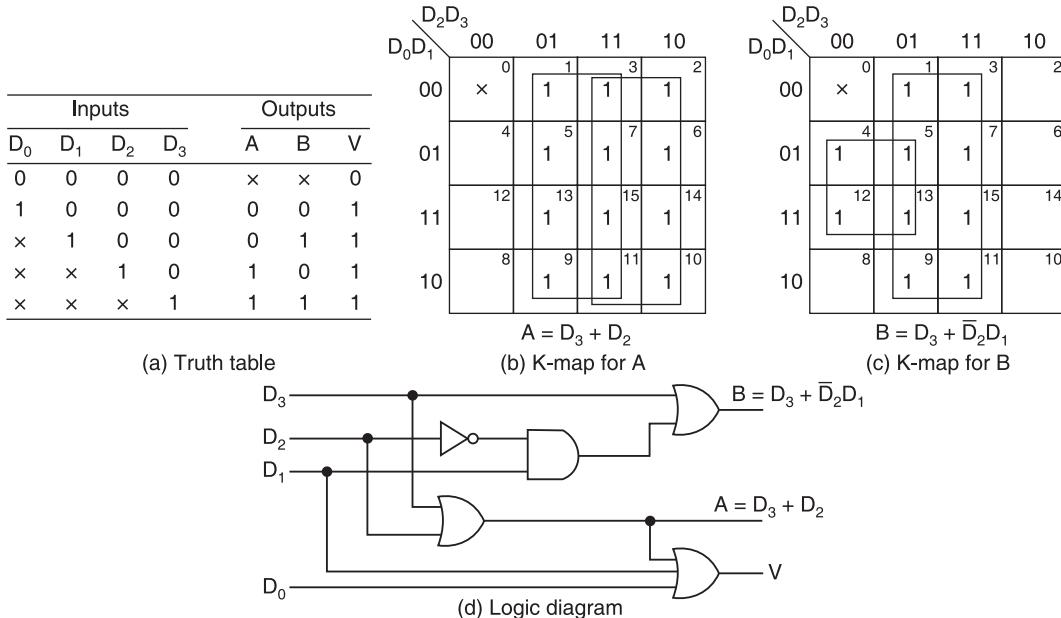


Figure 7.65 4-bit priority encoder.

### 7.22.2 Decimal-to-BCD Priority Encoder

This type of encoder performs the same basic function of encoding the decimal digits into 4-bit BCD outputs, as that performed by a normal decimal-to-BCD encoder. It, however, offers the additional facility of providing priority. That is, it produces a BCD output corresponding to the highest order decimal digit appearing on the inputs and ignores all others.

Now, let us look at the requirements for the priority detection logic. The purpose of this logic circuitry is to prevent a lower-order digit input from disrupting the encoding of a higher-order digit. This is accomplished by using inhibit gates. Referring to the truth table and logic diagram of the decimal-to-BCD encoder of Figure 7.63, note that A<sub>0</sub> is HIGH when D<sub>1</sub>, D<sub>3</sub>, D<sub>5</sub>, D<sub>7</sub>, or D<sub>9</sub> is HIGH.

In the priority encoder, input digit 1 will be allowed to activate the A<sub>0</sub> output, only if no higher order digits other than those that also activate A<sub>0</sub> are HIGH, i.e. D<sub>2</sub>, D<sub>4</sub>, D<sub>6</sub>, and D<sub>8</sub> must be LOW. If any of those are HIGH, then A<sub>0</sub> will be LOW. This can be stated as follows:

A<sub>0</sub> is HIGH if:

D<sub>1</sub> is HIGH and D<sub>2</sub>, D<sub>4</sub>, D<sub>6</sub>, and D<sub>8</sub> are LOW. Or

D<sub>3</sub> is HIGH and D<sub>4</sub>, D<sub>6</sub>, and D<sub>8</sub> are LOW. Or

D<sub>5</sub> is HIGH and D<sub>6</sub> and D<sub>8</sub> are LOW. Or

D<sub>7</sub> is HIGH and D<sub>8</sub> is LOW. Or

D<sub>9</sub> is HIGH.

Thus,

$$A_0 = D_1 \bar{D}_2 \bar{D}_4 \bar{D}_6 \bar{D}_8 + D_3 \bar{D}_4 \bar{D}_6 \bar{D}_8 + D_5 \bar{D}_6 \bar{D}_8 + D_7 \bar{D}_8 + D_9$$

Similarly,  $A_1$  is HIGH when  $D_2$ ,  $D_3$ ,  $D_6$ , or  $D_7$  is HIGH. So, in the priority encoder  $A_1$  will be HIGH if:

$D_2$  is HIGH and  $D_4$ ,  $D_5$ ,  $D_8$ , and  $D_9$  are LOW. Or

$D_3$  is HIGH and  $D_4$ ,  $D_5$ ,  $D_8$ , and  $D_9$  are LOW. Or

$D_e$  is HIGH and  $D_o$  and  $D_o$  are LOW. Or

D<sub>7</sub> is HIGH and D<sub>6</sub> and D<sub>5</sub> are LOW.

Thus,

$$A_1 = D_2 \bar{D}_4 \bar{D}_5 \bar{D}_8 \bar{D}_9 + D_3 \bar{D}_4 \bar{D}_5 \bar{D}_8 \bar{D}_9 + D_6 \bar{D}_8 \bar{D}_9 + D_7 \bar{D}_8 \bar{D}_9$$

Also,  $A_2$  is HIGH when  $D_4$ ,  $D_5$ ,  $D_6$ , or  $D_7$  is HIGH. So, in the priority encoder  $A_2$  will become HIGH if:

$D_4$  is HIGH and  $D_8$  and  $D_9$  are LOW. Or

$D_5$  is HIGH and  $D_8$  and  $D_9$  are LOW. Or

$D_c$  is HIGH and  $D_o$  and  $D_o$  are LOW. Or

$D_7$  is HIGH and  $D_6$  and  $D_5$  are LOW.

Thus

$$A_2 = D_4 \bar{D}_8 \bar{D}_0 + D_5 \bar{D}_8 \bar{D}_0 + D_6 \bar{D}_8 \bar{D}_0 + D_7 \bar{D}_8 \bar{D}_0$$

Finally,  $A_3$  is HIGH if  $D_8$  is HIGH or if  $D_9$  is HIGH. So, in the priority encoder  $A_3$  will be HIGH if  $D_8$  is HIGH OR  $D_9$  is HIGH.

Thus,

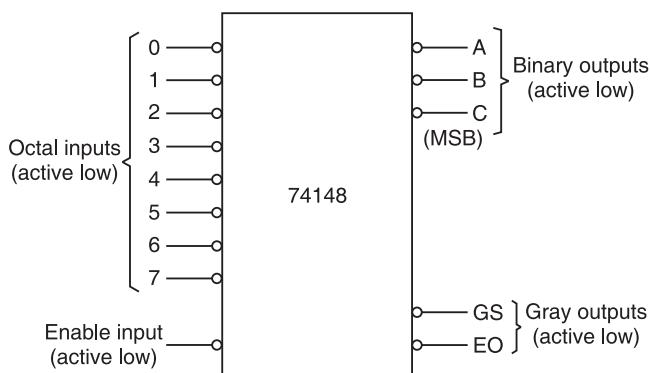
$$A_3 = D_8 + D_0$$

The truth table operation of the priority encoder is shown in Table 7.2. The inputs and outputs are active low. The truth table clearly shows that the magnitudes of the decimal inputs determine their priorities. If any decimal input is active, it is encoded provided all higher value inputs are inactive regardless of the states of all lower value inputs.

**Table 7.2** Truth table of a decimal-to-BCD priority encoder

### 7.22.3 Octal-to-Binary Priority Encoder

The octal code is often used at the inputs of digital circuits that require manual entering of long binary words. Priority encoder 74148 IC has been designed to achieve this operation. Its pin diagram is given in Figure 7.66. This circuit has active LOW inputs and active LOW outputs. The enable input and the Gray outputs which are also active LOW are used to cascade circuits to handle more inputs. A hexadecimal-to-binary encoder which is also a very useful circuit because of the widespread use of the hexadecimal code in computers, microprocessors, etc. can be designed using this facility.



**Figure 7.66** Pin diagram of an octal-to-binary priority encoder (IC 74148).

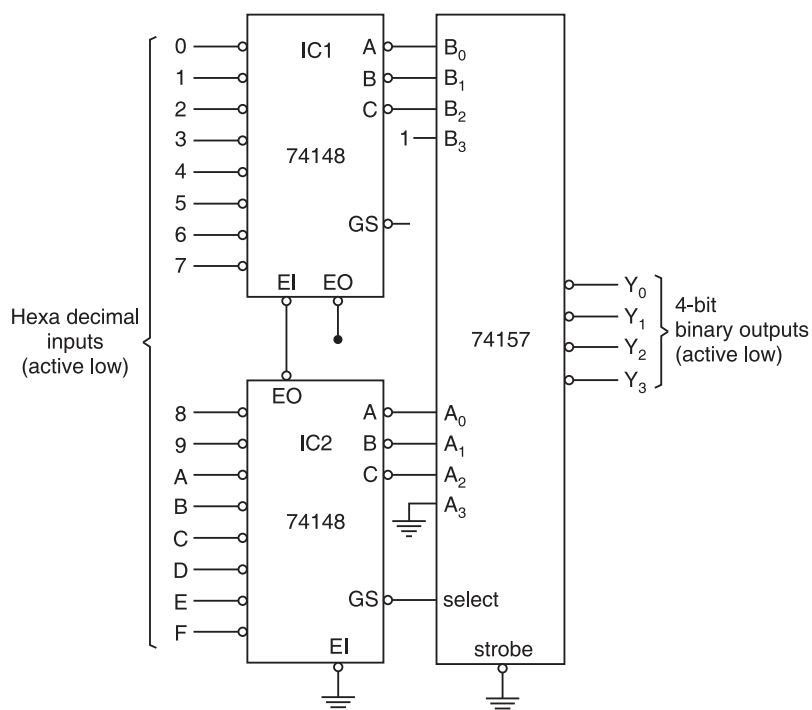
**EXAMPLE 7.10** Design a hexadecimal-to-binary encoder using 74148 encoders and 74157 multiplexer.

#### Solution

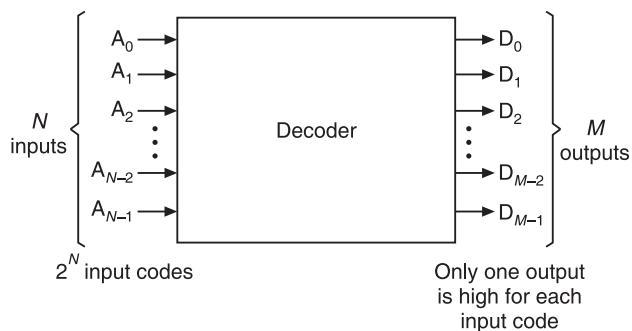
Since there are 16 symbols (0–F) in the hexadecimal number system, two 74148 encoders are required. Hexadecimal inputs 0–7 are applied to IC1 input lines and hexadecimal inputs 8–F to IC2 input lines. Whenever one of the inputs of IC2 is active (LOW), IC1 must be disabled. On the other hand, if all the inputs of IC2 are HIGH, then IC1 must be enabled. This is achieved by connecting the EO line of IC2 to the EI line of IC1. A quad 2:1 multiplexer is required to get the proper 4-bit binary outputs. The complete circuit is shown in Figure 7.67. The GS output of 74148 goes LOW whenever one of its inputs is active. Therefore, the GS of IC2 is connected to the SELECT input of 74157. The 74157 selects its A inputs if the SELECT input is LOW, otherwise B inputs are selected. The outputs of the multiplexer are the required binary outputs and are active LOW. This circuit is also a priority encoder.

## 7.23 DECODERS

A decoder is a logic circuit that converts an  $N$ -bit binary input code into  $M$  output lines such that only one output line is activated for each one of the possible combinations of inputs. In other words, we can say that a decoder identifies or recognizes or detects a particular code. Figure 7.68 shows the general decoder diagram with  $N$  inputs and  $M$  outputs. Since each of the  $N$  inputs can be a 0 or a 1, there are  $2^N$  possible input combinations or codes. For each of these input combinations,



**Figure 7.67** Hexadecimal-to-binary encoder using 74148s and 74157.



**Figure 7.68** General block diagram of a decoder.

only one of the  $M$  outputs will be active (HIGH), all the other outputs will remain inactive (LOW). Some decoders are designed to produce active LOW output, while all the other outputs remain HIGH.

Some decoders do not utilize all of the  $2^N$  possible input codes. For example, a BCD to decimal decoder has a 4-bit input code and 10 output lines that correspond to the 10 BCD code groups 0000 through 1001. Decoders of this type are often designed so that if any of the unused codes are applied to the input, none of the outputs will be activated.

### 7.23.1 3-Line-to-8-Line Decoder

Figure 7.69a shows the circuitry for a decoder with three inputs and eight outputs. It uses all AND gates, and therefore, the outputs are active-HIGH. For active-LOW outputs, NAND gates are used. The truth table of the decoder is shown in Figure 7.69b. This decoder can be referred to in several ways. It can be called a 3-line to 8-line decoder because it has three input lines and eight output lines. It is also called a binary-to-octal decoder because it takes a 3-bit binary input code and activates one of the eight (octal) outputs corresponding to that code. It is also referred to as a *1-of-8 decoder* because only one of the eight outputs is activated at one time.

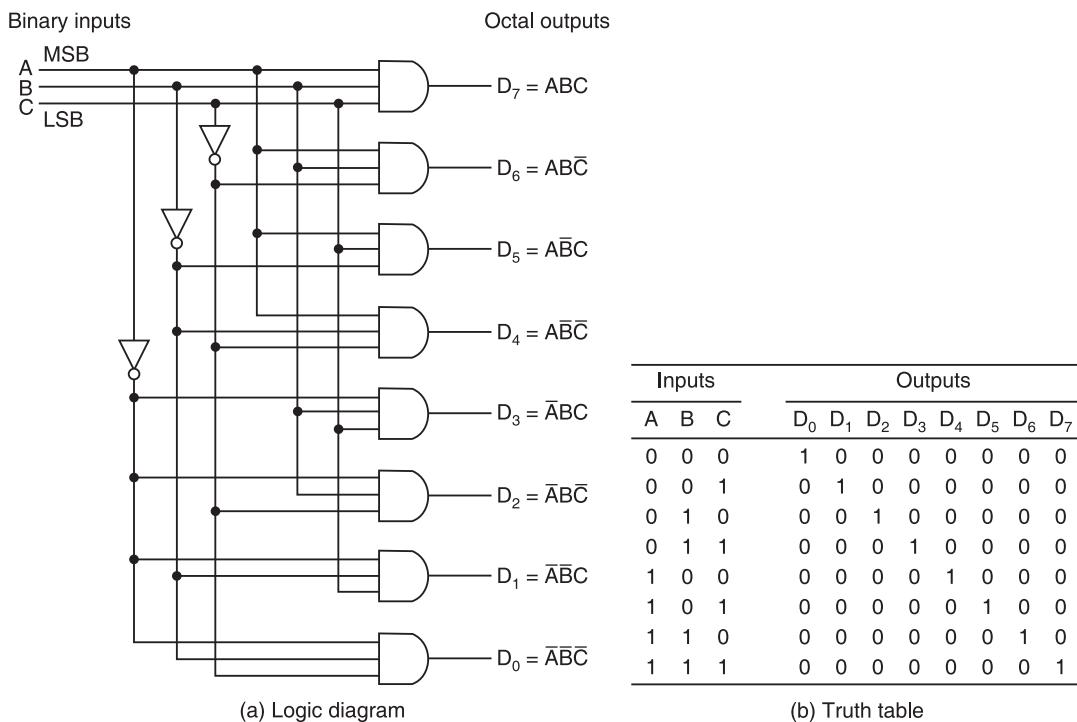


Figure 7.69 3-line to 8-line decoder.

### 7.23.2 Enable Inputs

Some decoders have one or more ENABLE inputs that are used to control the operation of the decoder. For example, in the 3-line to 8-line decoder, if a common ENABLE line is connected to the fourth input of each gate, a particular output as determined by the A, B, C input code will go HIGH only when the ENABLE line is held HIGH. When the ENABLE is held LOW, however, all the outputs will be forced to the LOW state regardless of the levels at the A, B, and C inputs. Thus, the decoder is enabled only when the ENABLE is HIGH. IC74 LS138 is a 3-line to 8-line decoder.

### 7.23.3 BCD-to-Decimal Decoder (7442)

The BCD to-decimal Decoder is also called a 4-line to 10-line or 4-to-10 decoder or 1-of-10 decoder. It has 4-input lines (for A<sub>3</sub>, A<sub>2</sub>, A<sub>1</sub>, A<sub>0</sub>) and 10 output lines (for D<sub>0</sub>, D<sub>1</sub>, D<sub>2</sub>, D<sub>3</sub>, D<sub>4</sub>, D<sub>5</sub>,

$D_6, D_7, D_8, D_9$ ). Only one output line is active at time. 6 of the 16 input combinations are invalid and for input combinations that are invalid for BCD none of the outputs will be activated. The inputs and outputs can be active high or active low. IC 7442 is a BCD to decimal decoder with active low inputs and outputs. The TTL 7445 IC is a BCD-to-decimal decoder/driver. The term driver is added to its description because this IC has open collector outputs that can operate at higher current and voltage limits than a normal TTL output. It makes 7445 suitable for directly driving loads such as indicator LEDs or lamps, relays or DC motors.

#### 7.23.4 2-Line-to-4-Line Decoder with NAND Gates

Some decoders are constructed with NAND gates. Since a NAND gate produces the AND operation with an inverted output, it becomes more economical to generate the decoder minterms in their complemented form. A 2-to-4 line decoder with an enable input is constructed with NAND gates as shown in Figure 7.70. The circuit operates with complemented outputs and complemented enable input. The decoder is enabled when  $E = 0$ . The inputs are active high and outputs active low. As indicated by the truth table, only one output can be equal to 0 at any given time, all other outputs are equal to 1. The output whose value is equal to 0 represents the minterm selected by inputs A and B. The circuit is disabled when  $E = 1$ , regardless of the values of the other two inputs. When the circuit is disabled, none of the outputs are equal to 0 and none of the minterms are selected. In general a decoder may operate with complemented or un complemented outputs. The enable input may be activated with a 0 or a 1 signal. Some decoders have two or more enable inputs that must satisfy a given logic condition in order to enable the circuits.

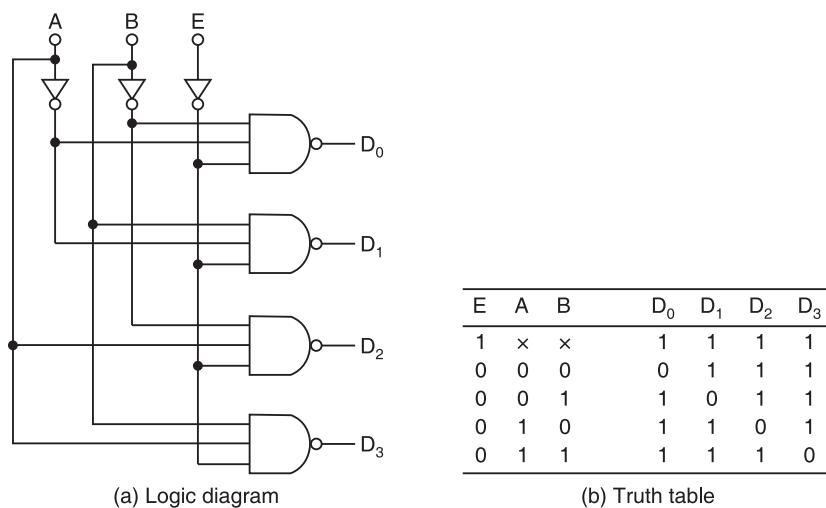


Figure 7.70 2 line-to-4 line decoder with NAND gates.

A decoder with enable input can function as a demultiplexer. A demultiplexer is a circuit that receives information from a single line and directs it to one of the  $2^n$  possible output lines. The selection of a specific output is controlled by the bit combination of  $n$  selection lines.

The decoder of Figure 7.70 can function as a 1-to-4 line demultiplexer when E is taken as a data input line and A and B are taken as the selection inputs. The single input variable E has a path

to all 4 outputs, but the input information is directed to only one of the output lines, as specified by the binary combination of two selection lines A and B. This can be verified from the truth table of the circuit. For example, if the selection lines AB = 10, the output D<sub>2</sub> will be same as the input value, while all other outputs are maintained at a 1. Because decoder and demultiplexer operations are obtained from the same circuit, a decoder with an enable input is referred to as a decoder/demultiplexer.

### 7.23.5 Combinational Logic Implementation

A decoder provides  $2^n$  minterms of  $n$  input variables. Since any Boolean function can be expressed in sum of minterms, one can use a decoder to generate the minterms and an external OR gate to form the logic sum. In this way any combinational circuit with  $n$  inputs and  $m$  outputs can be implemented with an  $n$ -to- $2^n$  decoder and  $m$  OR gates.

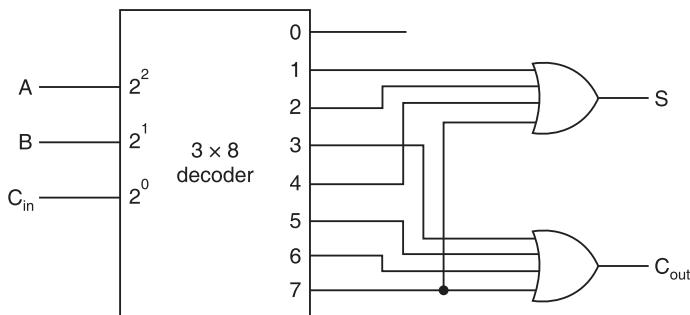
The procedure for implementing a combinational circuit by means of a decoder and OR gates requires that the Boolean function for the circuit is expressed in sum of minterms. A decoder that generates all the minterms of the input variables is then chosen. The inputs to each OR gate are selected from the decoder outputs according to the list of minterms of each function. This procedure will be illustrated by an example that implements a full adder.

From the truth table of the full adder, we obtain the functions for the combinational circuit in sum of minterms.

$$S = \overline{A}\overline{B}C_{in} + \overline{A}\overline{B}\overline{C}_{in} + A\overline{B}\overline{C}_{in} + ABC_{in} = A \oplus B \oplus C_{in} = \Sigma m(1, 2, 4, 7)$$

and  $C_{out} = \overline{ABC}_{in} + \overline{A}\overline{B}C_{in} + A\overline{B}C_{in} + ABC_{in} = AB + (A \oplus B)C_{in} = \Sigma m(3, 5, 6, 7)$

Since there are 3 inputs and a total of 8 minterms, we need a 3-to-8 line decoder. The implementation is shown in Figure 7.71. The decoder generates the 8 minterms for A, B, C<sub>in</sub>. The OR gate for output S forms the logical sum of minterms 1, 2, 4 and 7. The OR gate for C<sub>out</sub> forms the logical sum of the minterms 4, 5, 6 and 7.



**Figure 7.71** Logic diagram of a full adder using a decoder.

A function with a long list of minterms requires an OR gate with a large number of inputs. A function having a list of K minterms can be expressed in its complemented form  $\bar{F}$  with  $2^n - K$  terms. If the number of minterms in a function is greater than  $2^n/2$ , then  $\bar{F}$  can be expressed with fewer minterms. In such a case, it is advantageous to use a NOR gate to sum the minterms of  $\bar{F}$ .

The output of the NOR gate complements this sum and generates the normal output F. If NAND gates are used for the decoder as in Figure 7.68 then the external gates must be NAND gates instead of OR gates. This is because a two-level NAND gate circuit implements a sum of minterms function and is equivalent to a 2-level AND-OR circuit.

### 7.23.6 4-to-16 Decoder from Two 3-to-8 Decoders

Decoders with enable inputs can be connected together to form a larger decoder circuit. Figure 7.72 shows the arrangement for using two 74138s, 3-to-8 decoders, to obtain a 4-to-16 decoder. The most significant input bit  $A_3$  is connected through an inverter to  $\bar{E}$  on the upper decoder (for  $D_0$  through  $D_7$ ) and directly to E on the lower decoder (for  $D_8$  through  $D_{15}$ ). Thus, when  $A_3$  is LOW, the upper decoder is enabled and the lower decoder is disabled. The bottom decoder outputs all 0s, and top 8 outputs generate minterms. When  $A_3$  is HIGH, the lower decoder is enabled and the upper decoder is disabled. The bottom decoder outputs generate minterms 1000 to 1111 while the outputs of the top decoder are all 0s.

This example demonstrates the usefulness of enable inputs in decoders and other combinational logic components. In general, enable inputs are a convenient feature for interconnecting two or more standard components for the purpose of expanding the component into a similar function with more inputs and outputs.

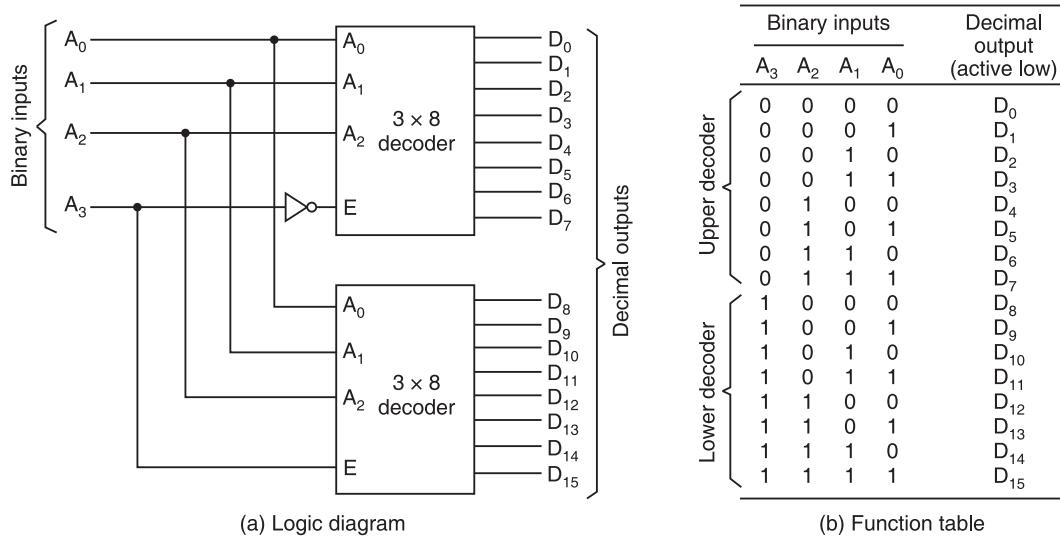


Figure 7.72 Connecting two 74138 3-to-8 decoders to obtain a 4-to-16 decoder.

### 7.23.7 Decoder Applications

Decoders are used whenever an output or a group of outputs is to be activated only on the occurrence of a specific combination of input levels. These input levels are often provided by the outputs of a counter or register. When the decoder inputs come from a counter that is being continually pulsed, the decoder outputs will be activated sequentially, and they can be used as timing or sequencing signals to turn devices on or off at specific times.

Decoders are widely used in memory systems of computers, where they respond to the address code input from the central processor to activate the memory storage location specified by the address code.

### 7.23.8 BCD-to-Seven Segment Decoders

This type of decoder accepts the BCD code and provides outputs to energize seven segment display devices in order to produce a decimal read out. Sometimes, the hex characters A through F may be produced. Each segment is made up of a material that emits light when current is passed through it. The most commonly used materials include LEDs, incandescent filaments and LCDs. The LEDs generally provide greater illumination levels but require more power than that by LCDs.

Figure 7.73a shows a seven-segment display consisting of seven light emitting segments. The segments are designated by letters a–g as shown in the figure. By illuminating various combinations of segments as shown in Figure 7.73b, the numbers 0–9 can be displayed. Figures 7.73c and d show two types of LED display—the common-anode and the common-cathode types.

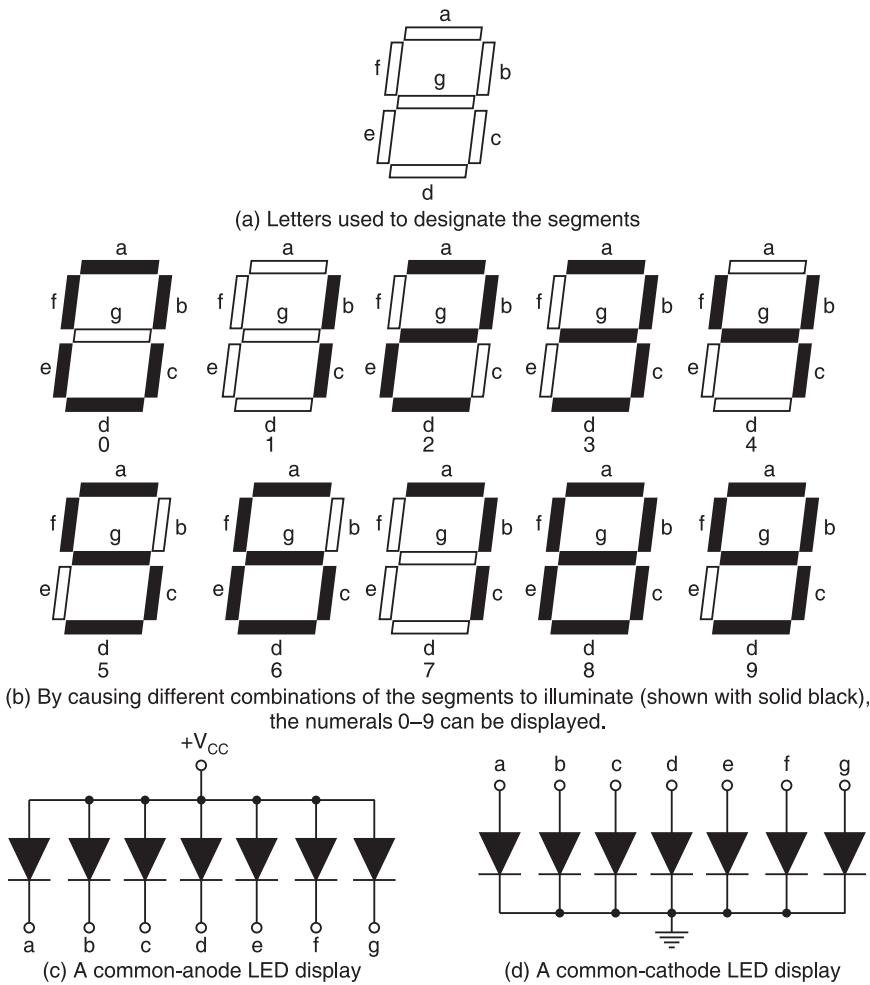


Figure 7.73 The seven segment display.

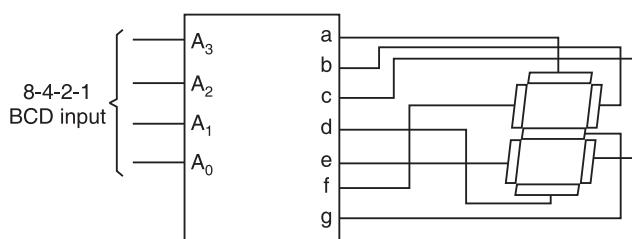
In the common-anode type, a low voltage applied to an LED cathode allows current to flow through the diode, which causes it to emit light. In the common-cathode type, a high voltage applied to an LED anode causes the current to flow and produces the resulting light emission.

An 8-4-2-1 BCD-to-seven segment decoder is a logic circuit as shown in Figure 7.74a. The function table for such a decoder is shown in Figure 7.74b. Since a 1 (HIGH) on any output line activates that line, we assume that the display is of the common-cathode type. The K-map used to simplify the logic expression for driving segment b is shown in Figure 7.74c. Entries 10–15 are don't cares as usual. Since LEDs require considerable power, decoders often contain output drivers capable of supplying sufficient power.

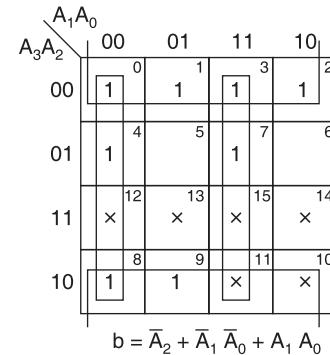
$$\begin{aligned} b &= \overline{A_3} \overline{A_2} \overline{A_1} \overline{A_0} + \overline{A_3} \overline{A_2} \overline{A_1} A_0 + \overline{A_3} \overline{A_2} A_1 \overline{A_0} + \overline{A_3} \overline{A_2} A_1 A_0 + \overline{A_3} A_2 \overline{A_1} \overline{A_0} \\ &\quad + \overline{A_3} A_2 A_1 \overline{A_0} + A_3 \overline{A_2} \overline{A_1} A_0 + A_3 \overline{A_2} \overline{A_1} \overline{A_0} \\ &= \Sigma m(0, 1, 2, 3, 4, 7, 8, 9) \end{aligned}$$

Don't cares,

$$d = \Sigma m(10, 11, 12, 13, 14, 15)$$



(a) Logic circuit



(c) K-map to derive simplified expression for driving segment (b)

Decimal digit	8-4-2-1 BCD				Seven segment code						
	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

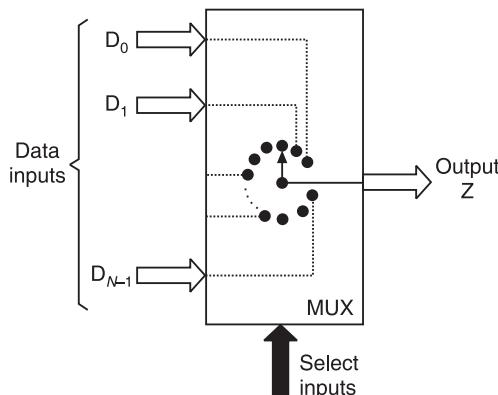
(b) Function table

Figure 7.74 BCD-to-seven segment decoder.

## 7.24 MULTIPLEXERS (DATA SELECTORS)

Multiplexing means sharing. There are two types of multiplexing—time multiplexing and frequency multiplexing. A common example of multiplexing or sharing occurs when several peripheral devices share a single transmission line or bus to communicate with a computer. To accomplish this sharing, each device in succession is allocated a brief time to send or receive data. At any given time, one and only one device is using the line. This is an example of time multiplexing, since each device is given specific time intervals to use the line. In frequency multiplexing, several devices share a common line by transmitting at different frequencies. In a large mainframe computer, numerous users are time-multiplexed to the computer in such a rapid succession that all appear to be using the computer simultaneously.

A multiplexer (MUX) or data selector is a logic circuit that accepts several data inputs and allows only one of them at a time to get through to the output. The routing of the desired data input to the output is controlled by SELECT inputs (sometimes referred to as ADDRESS inputs). Figure 7.75 shows the functional diagram of a general multiplexer. In this diagram, the inputs and outputs are drawn as large arrows to indicate that they may constitute one or more signal lines. Normally there are  $2^n$  input lines and  $n$  select lines whose bit combinations determine which input is selected.



**Figure 7.75** Functional diagram of a digital multiplexer.

The multiplexer acts like a digitally controlled multi-position switch. The digital code applied to the SELECT inputs determines which data inputs will be switched to the output. For example, the output Z will equal the data input  $D_0$  for some particular input code; Z will equal  $D_1$  for another particular code, and so on. In other words, we can say that a multiplexer selects 1-out-of- $N$  input data sources and transmits the selected data to a single output channel. This is called *multiplexing*.

### 7.24.1 Basic 2-Input Multiplexer

Figure 7.76 shows the logic circuitry and function table for a 2-input multiplexer with data inputs  $D_0$  and  $D_1$ , and data select input S. It connects two 1-bit sources to a common destination. It has two input lines, one data select line and one output line. The logic level applied to the S input

determines which AND gate is enabled, so that its data input passes through the OR gate to the output. The output,  $Z = D_0\bar{S} + D_1S$ .

When  $S = 0$ , AND gate 1 is enabled and AND gate 2 is disabled. So,  $Z = D_0$ .

When  $S = 1$ , AND gate 1 is disabled and AND gate 2 is enabled. So,  $Z = D_1$ .

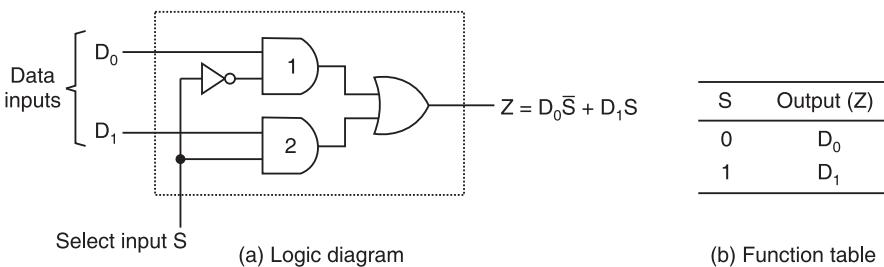


Figure 7.76 2-input multiplexer.

### 7.24.2 The 4-Input Multiplexer

Figure 7.77a shows the logic circuitry for a 4-input multiplexer with data inputs  $D_0, D_1, D_2$ , and  $D_3$ , and data select inputs  $S_0$  and  $S_1$ . The logic levels applied to the  $S_0$  and  $S_1$  inputs determine which AND gate is enabled, so that its data input passes through the OR gate to the output. The function table in Figure 7.77b gives the output for the input select codes as

$$Z = \bar{S}_1\bar{S}_0D_0 + \bar{S}_1S_0D_1 + S_1\bar{S}_0D_2 + S_1S_0D_3$$

The 2-4-8-16-input multiplexers are readily available in the TTL and CMOS families. These basic ICs can be combined for multiplexing a larger number of inputs. Some packages contain more than one multiplexer, for example, the 74157 quad 2-to-1 multiplexer (four 2-to-1 multiplexers having the same data select inputs) and the 74153 dual 4-to-1 multiplexer. Some designs have 3-state outputs and others have open collector outputs. Most have enable inputs to facilitate cascading.

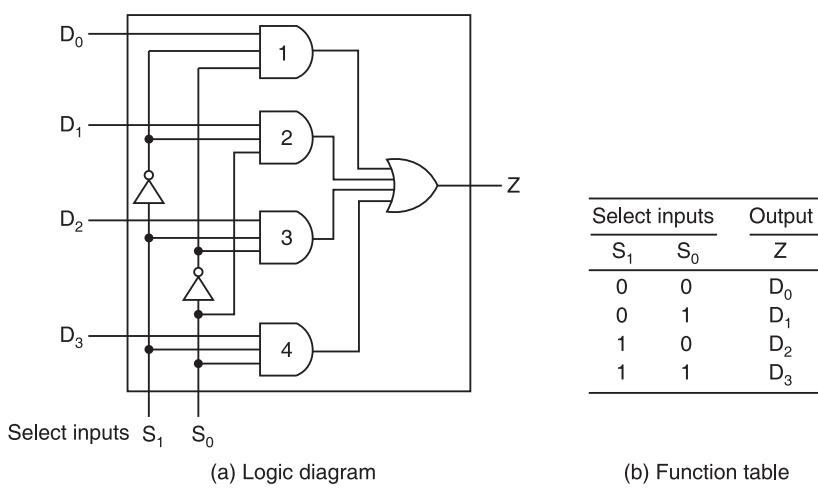
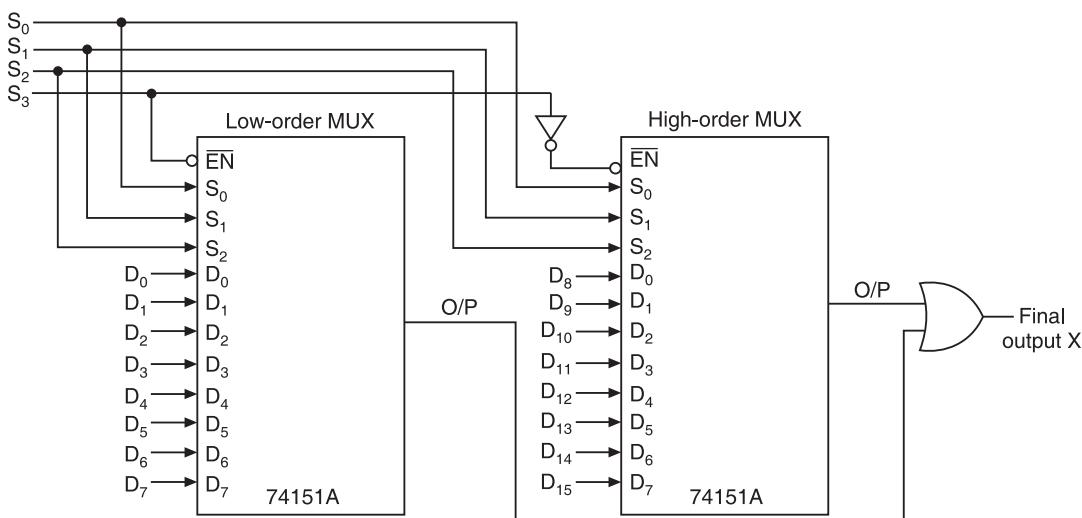


Figure 7.77 4-input multiplexer.

The AND gates and inverters in the multiplexer resemble a decoder circuit and, indeed, they decode the selection input lines. In general, a  $2^n$ -to-1 line multiplexer is constructed from an  $n$ -to- $2^n$  decoder by adding to it  $2^n$  input lines, one to each AND gate. The outputs of the AND gates are applied to a single OR gate. The size of multiplexer is specified by the number  $2^n$  of its data lines and the single output line. The  $n$  selection lines are implied from the  $2^n$  data lines. As in decoders multiplexers may have an enable input to control the operation of the unit. When the enable input is in the inactive state, the outputs are disabled and when it is in the active state, the circuit functions as a normal multiplexer.

### 7.24.3 The 16-Input Multiplexer from Two 8-Input Multiplexers

Figure 7.78 shows an arrangement to use two 8-input multiplexers (74151A) to get a 16-input multiplexer. One OR gate and one inverter are also required. The four select inputs  $S_3$ ,  $S_2$ ,  $S_1$ , and  $S_0$  will select one of the 16 inputs to pass through to X. The  $S_3$  input determines which multiplexer is enabled. When  $S_3 = 0$ , the left multiplexer is enabled and  $S_2$ ,  $S_1$ , and  $S_0$  inputs determine which of its data inputs will appear at its output and pass through the OR gate to X. When  $S_3 = 1$ , the right multiplexer is enabled and  $S_2$ ,  $S_1$ , and  $S_0$  inputs select one of its data inputs for passage to output X. This arrangement is also called multiplexer tree. Figure 7.94 shows an arrangement to obtain a  $32 \times 1$  mux using two  $16 \times 1$  muxes and one  $2 \times 1$  mux.



**Figure 7.78** Logic diagram for cascading of two  $8 \times 1$  muxes to get a 16-bit mux.

### 7.25 APPLICATIONS OF MULTIPLEXERS

Multiplexers find numerous and varied applications in digital systems of all types. These applications include data selection, data routing, operation sequencing, parallel-to-serial conversion, waveform generation, and logic function generation.

### 7.25.1 Logic Function Generator

A multiplexer can be used in place of logic gates to implement a logic expression. It can be so connected that it duplicates the logic of any truth table, i.e. it can generate any Boolean algebraic function of a set of input variables. In such applications, the multiplexer can be viewed as a function generator, because we can easily set or change the logic function it implements. One advantage of using a multiplexer in place of logic gates is that, a single integrated circuit can perform a function that might otherwise require numerous integrated circuits. Moreover, it is very easy to change the logic function implemented, if and when redesign of a system becomes necessary.

The first step in the design of a function generator using a multiplexer is to construct a truth table for the function to be implemented. Then, connect logic 1 to each data input of the multiplexer corresponding to each combination of the input variables, for which the truth table shows the function to be equal to 1. Logic 0 is connected to the remaining data inputs. The variables themselves are connected to the data select inputs of the multiplexer. For example, suppose the truth table specifies that the function  $F$  equals 1 for the input combination 110. So, when  $S_2S_1S_0 = 110$ , the data input 6, which is connected to logic 1, will be selected. This will route a 1 to the output of the multiplexer.

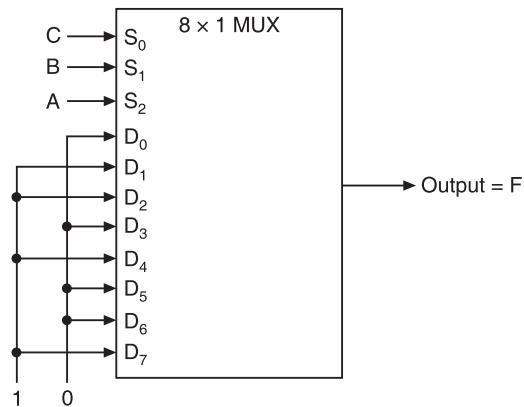
**EXAMPLE 7.11** Use a multiplexer to implement the logic function  $F = A \oplus B \oplus C$ .

#### Solution

The truth table for  $F$  and the logic diagram to implement  $F$  are shown in Figures 7.79a and b respectively. Since there are three input variables, we can use a multiplexer with three data select inputs (an 8-to-1 MUX). The truth table shows the use of data select inputs  $S_2$ ,  $S_1$ , and  $S_0$  for input variables  $A$ ,  $B$ , and  $C$  respectively. Since  $F = 1$  when  $ABC = 001, 010, 100$ , and 111, we connect logic 1 to data inputs  $D_1$ ,  $D_2$ ,  $D_4$ , and  $D_7$ . Logic 0 is connected to other data inputs  $D_0$ ,  $D_3$ ,  $D_5$ , and  $D_6$ . When the data select inputs are any of the combinations for which  $F = 1$ , the output will be a 1, and when the data select inputs are any of the combinations for which  $F = 0$ , the output will be a 0. Thus, the multiplexer behaves in exactly the same way that a set of logic gates implementing the function  $F$  would behave.

$S_2 \quad S_1 \quad S_0$			$F = A \oplus B \oplus C$
$A$	$B$	$C$	
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(a) Truth table



(b) Logic diagram

**Figure 7.79** Example 7.11: Use of 74151A to implement the logic function  $F = A \oplus B \oplus C$ .

**EXAMPLE 7.12** Implement the following function using 8-to-1 MUX

$$F(x, y, z) = \Sigma m(0, 2, 3, 5)$$

**Solution**

The truth table for F and the logic diagram to implement the function F using an 8 : 1 MUX are shown in Figures 7.80a and b respectively. The inputs x, y and z are applied to the data select inputs  $S_2$ ,  $S_1$  and  $S_0$  respectively. Since  $F = 1$  when  $xyz = 000, 010, 011$ , and  $101$ , logic 1 is connected to data inputs  $D_0, D_2, D_3$  and  $D_5$ . Logic 0 is connected to other data inputs  $D_1, D_4, D_6$  and  $D_7$ .

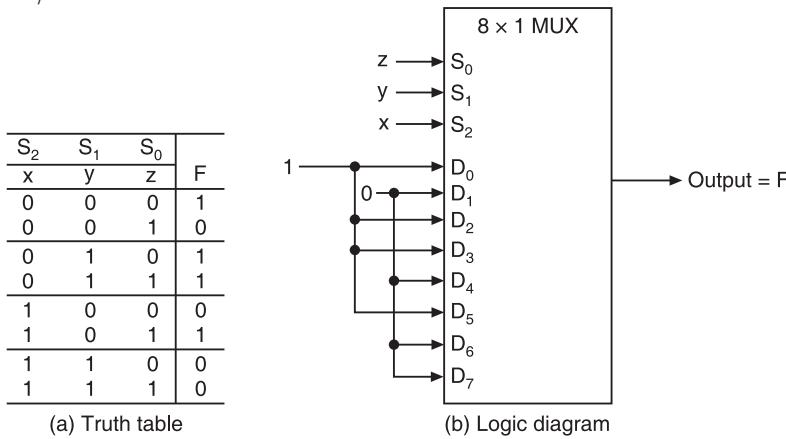


Figure 7.80 Example 7.12.

In general, a multiplexer with  $n$ -data select inputs can implement any function of  $n + 1$  variables. The key to this design is to use the first  $n$  variables of the function as the select inputs and to use the least significant input variable and its complement to drive some of the data inputs. If the single variable is denoted by D, each data output of the multiplexer will be D,  $\bar{D}$ , 1, or 0. Suppose, we wish to implement a 4-variable logic function using a multiplexer with three data select inputs. Let the input variables be A, B, C, and D; D is the LSB. A truth table for the function  $F(A, B, C, D)$  is constructed. In the truth table, we note that ABC has the same value twice once with  $D = 0$  and again with  $D = 1$ . The following rules are used to determine the connections that should be made to the data inputs of the multiplexer.

1. If  $F = 0$  both times when the same combination of ABC occurs, connect logic 0 to the data input selected by that combination.
2. If  $F = 1$  both times when the same combination of ABC occurs, connect logic 1 to the data input selected by that combination.
3. If F is different for the two occurrences of a combination of ABC, and if  $F = D$  in each case, connect D to the data input selected by that combination.
4. If F is different for the two occurrences of a combination of ABC, and if  $F = \bar{D}$  in each case, connect  $\bar{D}$  to the data input selected by that combination.

**EXAMPLE 7.13** Use a multiplexer having three data select inputs to implement the logic for the function given below. Also realize the same using a 16:1 MUX.

$$F = \Sigma m(0, 1, 2, 3, 4, 10, 11, 14, 15).$$

**Solution**

The truth table for the given function is shown in Figure 7.81a. Since the given function is of four variables, we can use a multiplexer with three data select inputs (i.e. 8:1 mux) as shown

in Figure 7.81b. As seen from the truth table, since F is same for each of the two occurrences of ABC = 000, ABC = 001, ABC = 101, and ABC = 111 and since F = 1 in both cases, 1 is connected to D<sub>0</sub>, D<sub>1</sub>, D<sub>5</sub>, D<sub>7</sub>. Since F is the same for each of two occurrences of ABC = 011, ABC = 100 and ABC = 110 and since F = 0 in both cases, 0 is connected to D<sub>3</sub>, D<sub>4</sub> and D<sub>6</sub>. Since F is different for each of the two occurrences of ABC = 010 and since F =  $\bar{D}$  in both cases,  $\bar{D}$  is connected to D<sub>2</sub>.

Realization of the same using a 16:1 MUX is shown in Figure 7.81c.

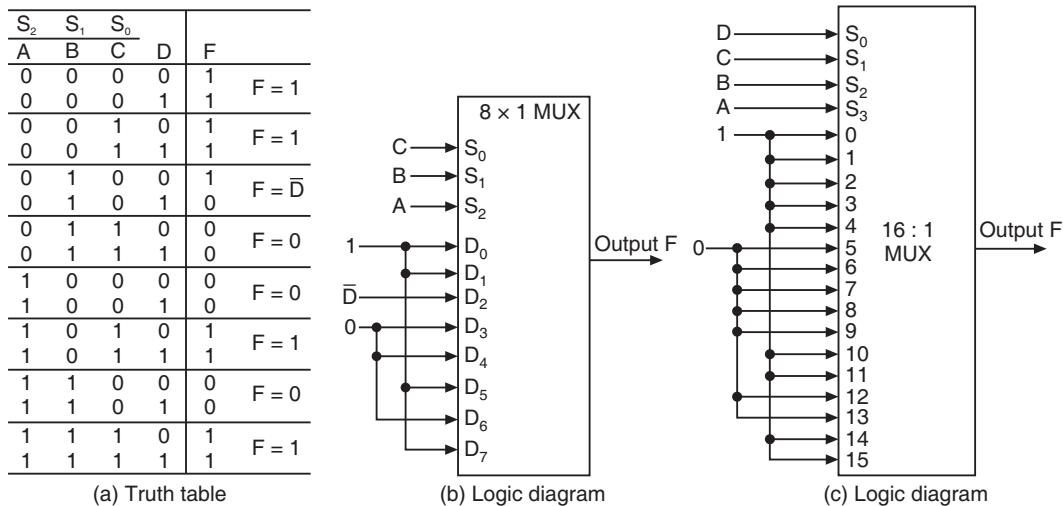


Figure 7.81 Example 7.13.

**EXAMPLE 7.14** Use a 4 × 1 MUX to implement the logic function

$$F(A, B, C) = \sum m(1, 2, 4, 7).$$

### Solution

The logic function  $F(A, B, C) = \sum m(1, 2, 4, 7)$  can be implemented with a 4-to-1 multiplexer as shown in Figure 7.82. The given function is a 3 variable function. The two variables A and B are applied to the selection lines in that order. A is connected to S<sub>1</sub> input and B is connected to S<sub>0</sub> input. The values for the data input lines are determined from the truth table of the function. When

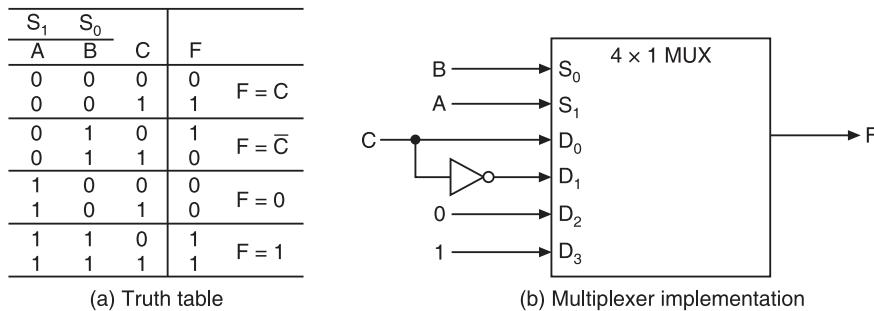


Figure 7.82 Example 7.14.

$AB = 00$ , the output  $F = C$ , because  $F = 0$  when  $C = 0$  and  $F = 1$  when  $C = 1$ . This requires that variable  $C$  be applied to data line 0. The operation of the multiplexer is such that when  $AB = 00$ , data input 0 has a path to the output and that makes  $F = C$ . In a similar fashion, we can determine the required input to data lines 1, 2, and 3 from the values of  $F$  when  $AB = 01, 10$ , and 11 respectively.

**EXAMPLE 7.15** Implement the following function with a MUX:

$$F(a, b, c) = \Sigma m(1, 3, 5, 6)$$

Choose  $a$  and  $b$  as select inputs.

**Solution**

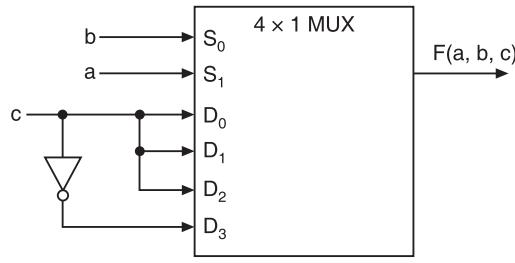
The truth table for  $F$  and the implementation of  $F(a, b, c) = \Sigma m(1, 3, 5, 6)$  using a  $4 : 1$  MUX with select inputs  $a$  and  $b$  are shown in Figures 7.83a and b respectively.

The inputs  $a$  and  $b$  are connected to the data select lines  $S_1$  and  $S_0$ . From the truth table we observe that

1. For both values of  $ab = 00, ab = 01$  and  $ab = 10$ ,  $F = c$ . So  $D_0, D_1$  and  $D_2$  are connected to  $c$ .
2. For both values of  $ab = 11$ ,  $F = \bar{c}$ . So  $D_3$  is connected to  $\bar{c}$ .

$S_1$	$S_0$	$c$	$F$
$a$	$b$		
0	0	0	0 $F = c$
0	0	1	1
0	1	0	0 $F = c$
0	1	1	1
1	0	0	0 $F = c$
1	0	1	1
1	1	0	1 $F = \bar{c}$
1	1	1	0

(a) Truth table



(b) Multiplexer implementation

Figure 7.83 Example 7.15.

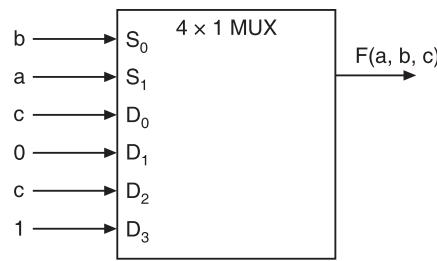
**EXAMPLE 7.16** Implement the function  $F(a, b, c) = ab + \bar{b}c$  using a  $4 : 1$  MUX.

**Solution**

The truth table for  $F$  and the realization of the given function  $F(a, b, c) = ab + \bar{b}c = abc + ab\bar{c} + \bar{a}\bar{b}c + a\bar{b}c = \Sigma m(1, 5, 6, 7)$  using a  $4 : 1$  MUX with two select inputs  $a$ , and  $b$  are shown in Figures 7.84a and b respectively.

$S_1$	$S_0$	$c$	$F$
$a$	$b$		
0	0	0	0 $F = c$
0	0	1	1
0	1	0	0 $F = 0$
0	1	1	0
1	0	0	0
1	0	1	1 $F = c$
1	1	0	1
1	1	1	1 $F = 1$

(a) Truth table



(b) Multiplexer implementation

Figure 7.84 Example 7.16.

The inputs  $a$  and  $b$  are connected to the data select lines  $S_1$  and  $S_0$ . From the truth table we observe that

1. For both values of  $ab = 00$ , and  $ab = 10$ ,  $F = c$ . So  $D_0$  and  $D_2$  are connected to  $c$ .
2. For both values of  $ab = 01$ ,  $F = 0$ . So  $D_1$  is connected to 0.
3. For both values of  $ab = 11$ ,  $F = 1$ . So  $D_3$  is connected to 1.

**EXAMPLE 7.17** Implement the following logic function using an  $8 \times 1$  MUX:

$$F(A, B, C, D) = \sum m(1, 3, 4, 11, 12, 13, 14, 15)$$

**Solution**

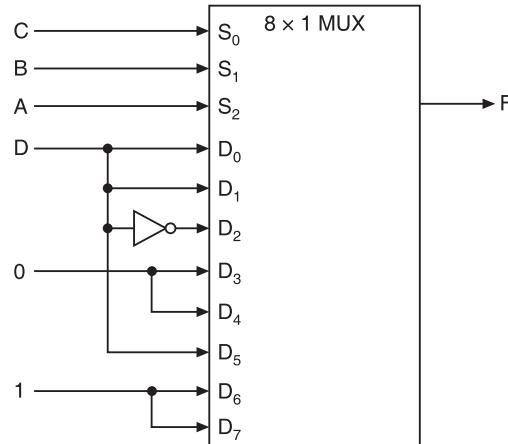
The truth table for the given four variable function  $F$  and its implementation using an 8:1 MUX with three select inputs  $A$ ,  $B$ , and  $C$  are shown in Figures 7.85a and b respectively.

The inputs  $A$ ,  $B$ , and  $C$  are connected to the data select lines  $S_2$ ,  $S_1$ , and  $S_0$ . From the truth table we observe that

1. For both values of  $ABC = 000$ ,  $ABC = 001$  and  $ABC = 101$ ,  $F = D$ . So  $D_0$ ,  $D_1$ , and  $D_5$  are connected to  $D$ .
2. For both values of  $ABC = 010$ ,  $F = \bar{D}$ . So  $D_2$  is connected to  $\bar{D}$ .
3. For both values of  $ABC = 011$ , and  $ABC = 100$ ,  $F = 0$ . So  $D_3$  and  $D_4$  are connected to 0.
4. For both values of  $ABC = 110$ , and  $ABC = 111$ ,  $F = 1$ . So  $D_6$  and  $D_7$  are connected to 1.

$S_2$	$S_1$	$S_0$		
$A$	$B$	$C$	$D$	$F$
0	0	0	0	0 $F = D$
0	0	0	1	1
0	0	1	0	0 $F = D$
0	0	1	1	1
0	1	0	0	1 $F = \bar{D}$
0	1	0	1	0
0	1	1	0	0 $F = 0$
0	1	1	1	0
1	0	0	0	0 $F = 0$
1	0	0	1	0
1	0	1	0	0 $F = D$
1	0	1	1	1
1	1	0	0	1 $F = 1$
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1 $F = 1$

(a) Truth table



(b) Logic diagram

**Figure 7.85** Example 7.17.

**EXAMPLE 7.18** Implement the following function using  $8 : 1$  MUX

$$F(x, y, z) = \sum m(0, 2, 3, 5)$$

**Solution**

The truth table for the given three variable function  $F$  and its implementation using an 8:1 MUX with three select inputs  $x$ ,  $y$ , and  $z$  are shown in Figures 7.86a and b respectively.

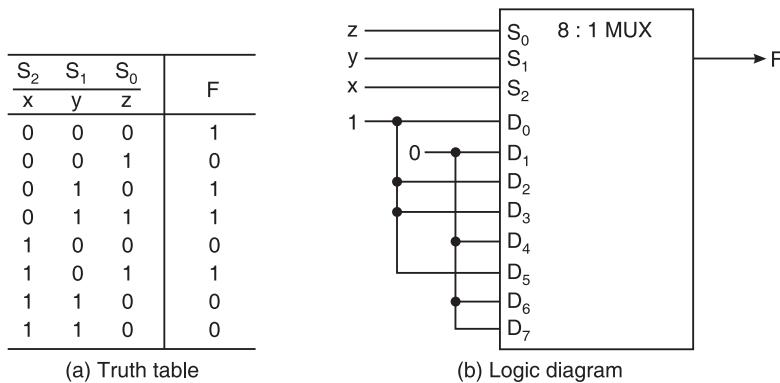


Figure 7.86 Example 7.18.

The inputs  $x$ ,  $y$  and  $z$  are connected to the data select lines  $S_2$ ,  $S_1$  and  $S_0$ . From the truth table we observe that

1.  $F = 1$  for  $xyz = 000, 010, 011$  and  $101$ . So connect  $D_0, D_2, D_3$  and  $D_5$  to 1.
2.  $F = 0$  for  $xyz = 001, 100, 110$  and  $111$ . So connect  $D_1, D_4, D_6$  and  $D_7$  to 0.

**EXAMPLE 7.19** Implement the following Boolean function using an 8:1 multiplexer considering D as the input and A, B, C as the selection lines:

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

**Solution**

$$\begin{aligned} F(A, B, C, D) &= A\bar{B} + BD + \bar{B}C\bar{D} = 10XX + X1X1 + X010 \\ &= 1000 + 1001 + 1010 + 1011 + 0101 + 0111 + 1101 + 1111 + 0010 + 1010 \\ &= \Sigma m(8, 9, 10, 11, 5, 7, 13, 15, 2, 10) = \Sigma m(2, 5, 7, 8, 9, 10, 11, 13, 15) \end{aligned}$$

The truth table for the given four variable function  $F$  and its implementation using an 8:1 MUX with three select inputs A, B, and C are shown in Figures 7.87a and b respectively.

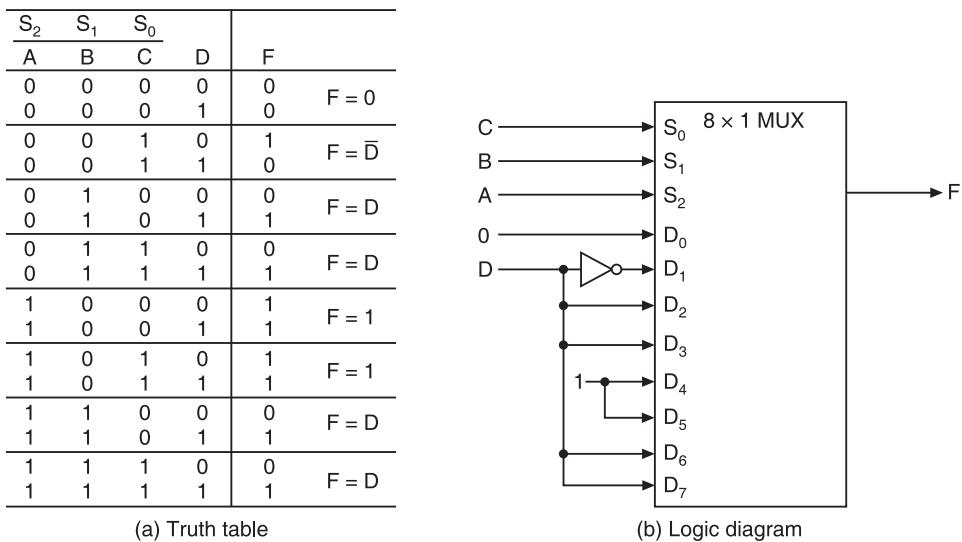


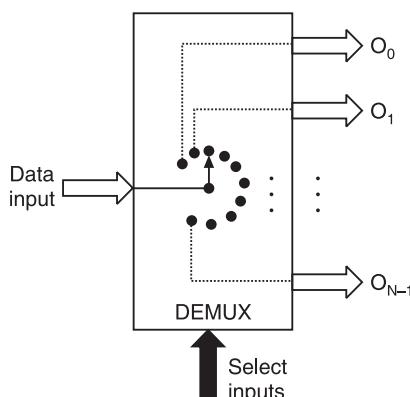
Figure 7.87 Example 7.19.

The inputs A, B, and C are connected to the data select lines  $S_2$ ,  $S_1$ , and  $S_0$ . From the truth table we observe that

1. For both values of  $ABC = 000$ ,  $F = 0$ . So  $D_0$  is connected to 0.
2. For both values of  $ABC = 100$ , and  $ABC = 101$ ,  $F = 1$ . So  $D_4$  and  $D_5$  are connected to 1.
3. For both values of  $ABC = 010$ ,  $ABC = 011$ ,  $ABC = 110$ , and  $ABC = 111$ ,  $F = D$ . So  $D_2$ ,  $D_3$ ,  $D_6$  and  $D_7$  are connected to D.
4. For both values of  $ABC = 001$ ,  $F = \bar{D}$ . So  $D_1$  is connected to  $\bar{D}$ .

## 7.26 DEMULTIPLEXERS (DATA DISTRIBUTORS)

A multiplexer takes several inputs and transmits one of them to the output. A demultiplexer performs the reverse operation; it takes a single input and distributes it over several outputs. So a demultiplexer can be thought of as a ‘distributor’, since it transmits the same data to different destinations. Thus, whereas a multiplexer is an  $N$ -to-1 device, a demultiplexer is a 1-to- $N$  (or  $2^n$ ) device. Figure 7.88 shows the functional diagram for a demultiplexer (DEMUX). The large arrows for inputs and outputs can represent one or more lines. The ‘select’ input code determines the output line to which the input data will be transmitted. In other words, the demultiplexer takes one input data source and selectively distributes it to 1-of- $N$  output channels just like a multi-position switch.



**Figure 7.88** Functional diagram of a general demultiplexer.

### 7.26.1 1-Line to 4-Line Demultiplexer

Figure 7.89 shows a 1-line to 4-line demultiplexer circuit. The input data line goes to all of the AND gates. The two select lines  $S_0$  and  $S_1$  enable only one gate at a time, and the data appearing on the input line will pass through the selected gate to the associated output line.

### 7.26.2 1-Line to 8-Line Demultiplexer

Figure 7.90a shows the logic diagram for a demultiplexer that distributes one input line to eight output lines. The single data input line D is connected to all eight AND gates, but only one of these gates will be enabled by the select input lines. For example, with  $S_2 S_1 S_0 = 000$ , only the AND

gate  $O_0$  will be enabled, and the data input D will appear at output  $O_0$ . Other select codes cause input D to reach the other outputs. The truth table in Figure 7.90b summarizes the operation.

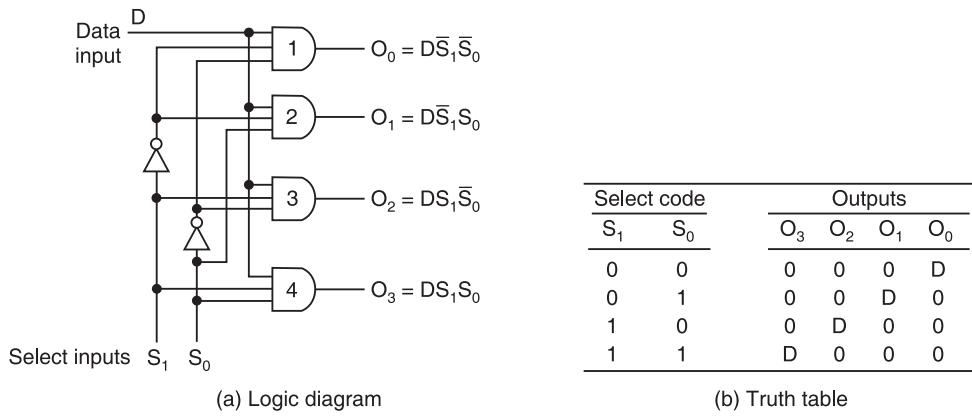


Figure 7.89 1-line to 4-line demultiplexer.

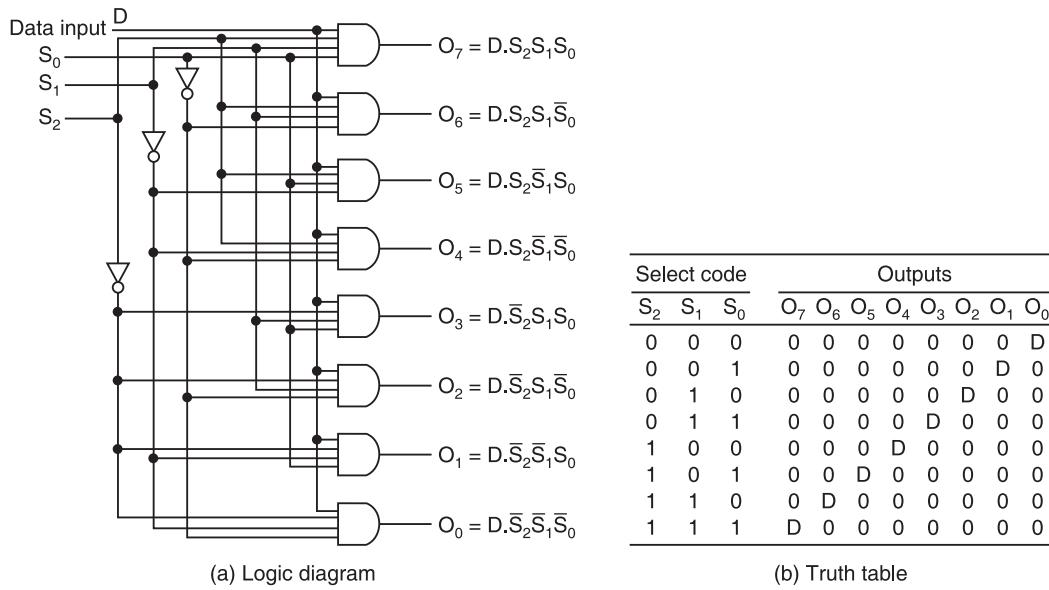


Figure 7.90 1-line to 8-line demultiplexer.

The demultiplexer circuit of Figure 7.90a is very similar to the 3-line to 8-line decoder circuit of Figure 7.69a, except that a fourth input D has been added to each gate. The inputs ABC of Figure 7.69b are here labelled  $S_2$   $S_1$   $S_0$  and become the data select inputs.

In the 3-to-8 IC decoder, there are three input lines and eight output lines. The enable input  $\bar{E}$  is used to enable or disable the decoding process. This 3-to-8 decoder can be used as a 1-to-8 demultiplexer as follows.

The enable input  $\bar{E}$  is used as the data input D, and the binary code inputs are used as the select inputs. Depending on the select inputs, the data input will be routed to a particular output. For this reason, the IC manufacturers often call this type of device a decoder/demultiplexer.

The 74LS138 decoder can be used as a demultiplexer by using  $\bar{E}_1$  as the data input D, holding the other two enable inputs in their active states and using the  $A_2A_1A_0$  inputs as the select code.

**EXAMPLE 7.20** Implement the following multiple output combinational logic circuit using a 4-line to 16-line decoder.

$$F_1 = \Sigma m(1, 2, 4, 7, 8, 11, 12, 13)$$

$$F_2 = \Sigma m(2, 3, 9, 11)$$

$$F_3 = \Sigma m(10, 12, 13, 14)$$

$$F_4 = \Sigma m(2, 4, 8)$$

### Solution

The realization of the given multiple output logic circuit using a 4-line to 16-line decoder is shown in Figure 7.91. The decoder's outputs are active LOW; therefore, a NAND gate is required for every output of the combinational circuit. In combinational logic design using a multiplexer, additional gates are not required, whereas the design using a demultiplexer requires additional gates. However, even with this disadvantage, the decoder is more economical in cases where non-trivial, multiple-output expressions of the same input variables are required. In such cases, one multiplexer is required for each output, whereas it is likely that only one decoder supported with a few gates would be required. Therefore, using a decoder could have advantages over using a multiplexer.

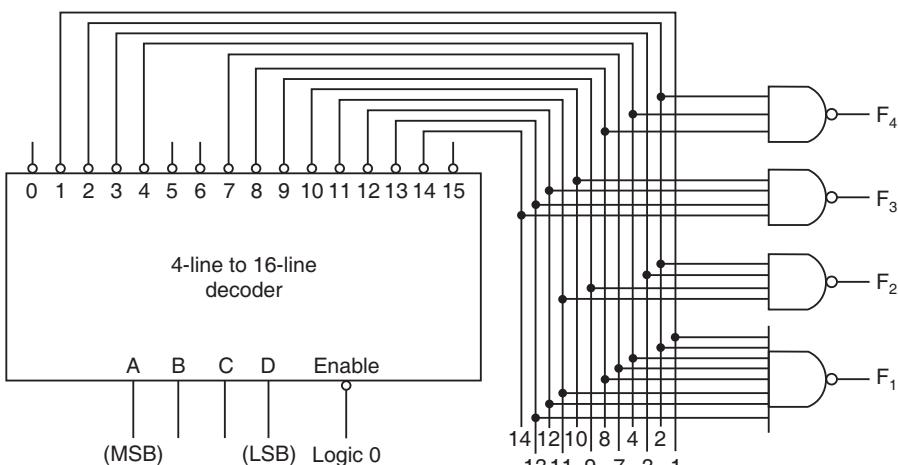
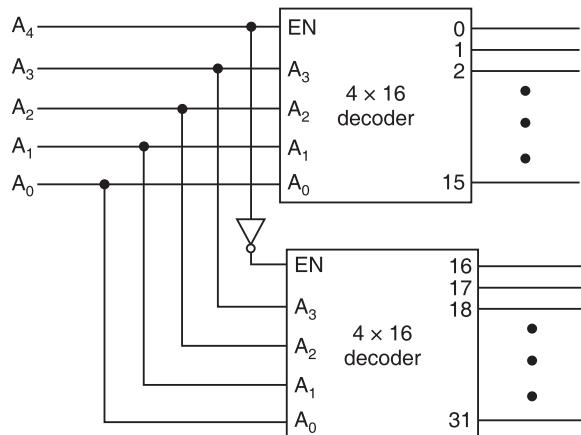


Figure 7.91 Example 7.20.

### 7.26.3 Demultiplexer Tree

Since 4-line to 16-line decoders are the largest available circuits in ICs, to meet the still larger input needs, there should be a provision for expansion. This is made possible by using enable input terminal. Figure 7.92 shows a 5-line to 32-line decoder/demultiplexer using two 4-line to 16-

line decoders. In a similar manner 6-line to 64-line, 7-line to 128-line and 8-line to 256-line and in general  $m$ -line to  $n$ -line decoders can be implemented.



**Figure 7.92** 5:32 decoder from two 4:16 decoders.

## 7.27 MODULAR DESIGN USING IC CHIPS

Earlier, the design of digital circuits was based on discrete components like transistors, diodes, resistors and so on. Now the logic circuit design relies heavily on the availability of modules called *integrated circuits* which are fabricated in solid state form and are referred to as IC chips. The design using ICs is called modular design. There are many levels of integration depending on the number of transistors and other components provided on the chip. Small scale integration (SSI), Medium scale integration (MSI), large scale integration (LSI), very large scale integration (VLSI), are all in use for various applications ranging from realization of simple logic functions to large digital systems like computers. Most of the modules presented in this chapter belong to SSI or MSI category. The designer will have the choice of modular expansion by using the chip enable lead illustrated by the following examples.

### 7.27.1 Design of a 16:1 Mux Using 4:1 Mux Modules

Figure 7.93 shows an arrangement to realize a 16:1 MUX using 4:1 muxes.

Since 16 inputs are there, the first four inputs are applied to the first 4:1 mux, the second four inputs to the second 4:1 mux, the third four inputs to the third 4:1 mux, and the fourth four inputs are applied to the fourth 4:1 mux. Four select inputs are required. Select inputs C and D are applied to  $S_1$  and  $S_0$  terminals of the four muxes. The outputs of these muxes are connected as data inputs to the fifth 4:1 mux and select inputs A and B are applied to  $S_1$  and  $S_0$  of that mux. The output of the last 4:1 mux is  $F(A, B, C, D)$ . When  $CD = 00$ ,  $D_0$  will appear at  $F_1$ ,  $D_4$  at  $F_2$ ,  $D_8$  at  $F_3$  and  $D_{12}$  at  $F_4$ . When  $CD = 01$ ,  $D_1$  appears at  $F_1$ ,  $D_5$  at  $F_2$ ,  $D_9$  at  $F_3$  and  $D_{13}$  at  $F_4$ . Similarly, when  $CD = 10$ ,  $D_2$ ,  $D_6$ ,  $D_{10}$  and  $D_{14}$  appear at  $F_1$ ,  $F_2$ ,  $F_3$  and  $F_4$  respectively. When  $CD = 11$ ,  $D_3$ ,  $D_7$ ,  $D_{11}$ , and  $D_{15}$  appear at  $F_1$ ,  $F_2$ ,  $F_3$  and  $F_4$  respectively. Depending upon the values of AB either  $F_1$  or  $F_2$  or  $F_3$  or  $F_4$  will appear at the output.

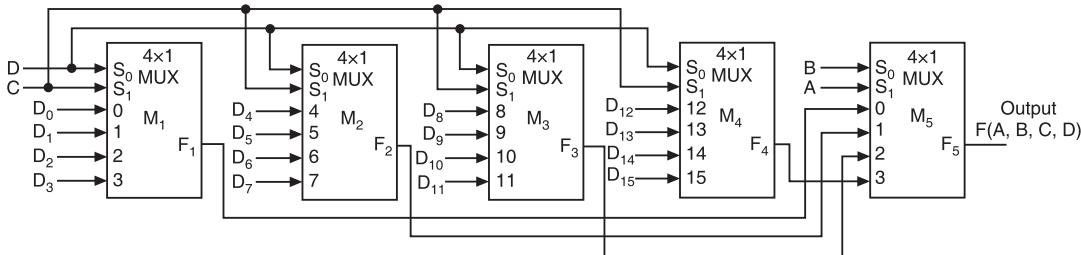


Figure 7.93 16:1 mux using 4:1 muxes.

### 7.27.2 Design of a 32:1 Mux Using Two 16:1 Muxes and One 2:1 Mux Modules

The arrangement to obtain a 32:1 mux using two 16:1 muxes and one 2:1 mux is shown in Figure 7.94. A 32:1 mux has 32 data inputs. So it requires five data select lines. Since a 16:1 mux has only four data select lines, the inputs B,C,D,E are connected to the data select lines of both the 16:1 muxes and the most significant input A is connected to the single data select line of the 2:1 mux. For the values of BCDE = 0000 to 1111, inputs 0 to 15 will appear at the input terminal 0 of the 2:1 mux through the output F<sub>1</sub> of the first 16:1 mux and inputs 16 to 31 will appear at the input terminal 1 of the 2:1 mux through the output F<sub>2</sub> of the second 16:1 mux. For A = 0, output F = F<sub>1</sub>. For A = 1, output F = F<sub>2</sub>.

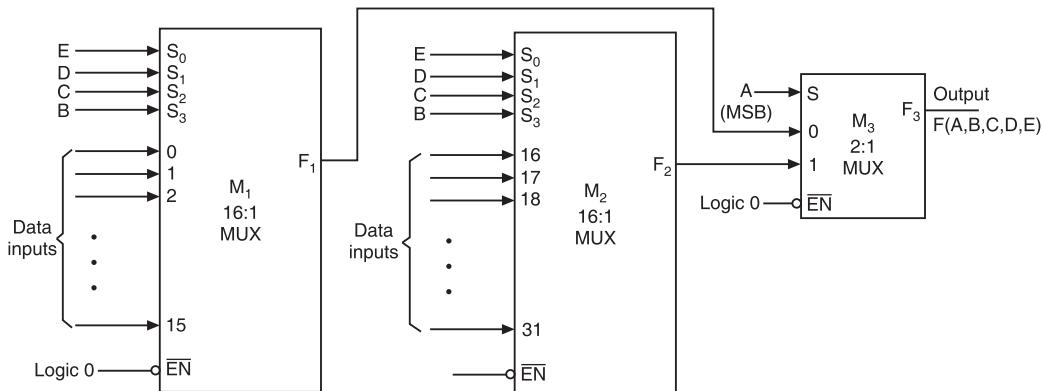


Figure 7.94 32:1 mux using two 16:1 muxes and one 2:1 mux.

### 7.27.3 Design of a $10 \times 1k$ Decoder Using Chips of $8 \times 256$ Decoder and Additional Logic

For each of the given decoder chips, there are 8 address inputs and  $2^8 = 256$  outputs. In addition, there is a chip enable (CE) input, which feeds all the AND gates in the chip. The decoded output becomes 1 only if CE is at 1. The enable input is also called chip select (CS), or strobe input. Four such modules are necessary to decode 10 address inputs and produce  $2^{10} = 1024$  outputs. This requires a simple strategy to connect the least significant 8 bits of the 10-bit address to all the 4 decoders and use the remaining two most significant bits to enable only one of the 4 chips by using discrete AND gates to produce  $\bar{A}_9\bar{A}_8$ ,  $A_9A_8$ ,  $A_9\bar{A}_8$ ,  $A_9A_8$ . It would be more elegant to use a small decoder of size  $2 \times 4$  instead of discrete gates as shown in Figure 7.95.

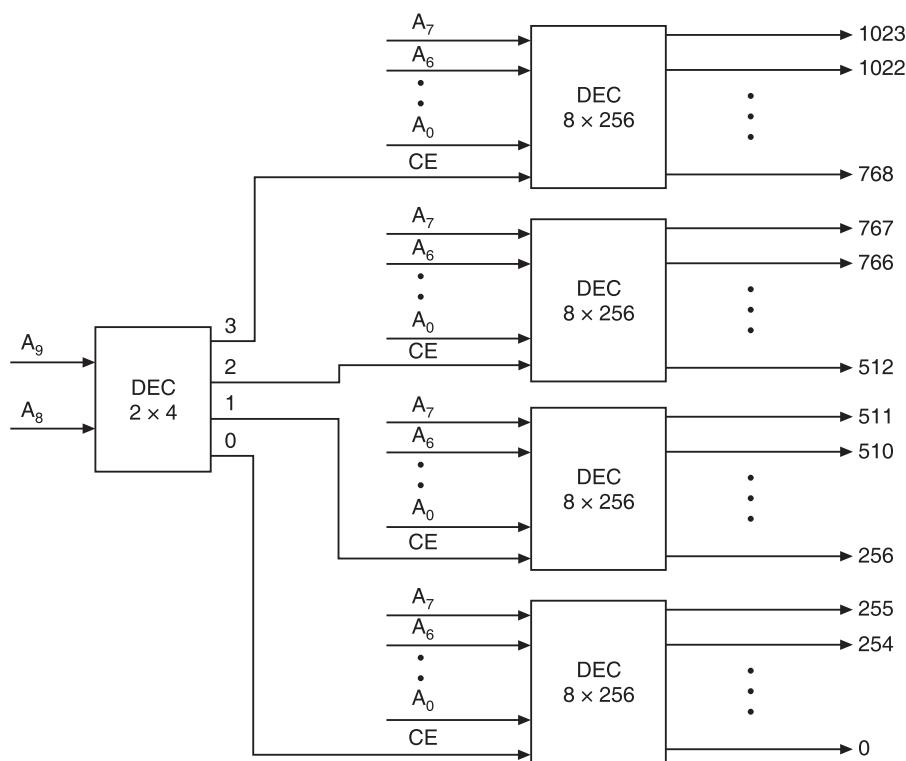


Figure 7.95 Modular expansion of decoding.

#### 7.27.4 Design of a $2k \times 1$ Mux Using 1k Modules

Since a  $2k \times 1$  mux is to be designed using 1k modules to select one out of  $2k$  inputs, we have to use two 1k muxes and we must provide eleven address bits to produce  $2^{11} = 2k$  different addresses. They may be called  $A_{10} \dots A_0$ . The scheme as shown in Figure 7.96 is to connect the 10 bits,  $A_9, \dots, A_0$  to both the 1k chips and use the most significant bit to enable one chip with  $\bar{A}_{10}$  and another

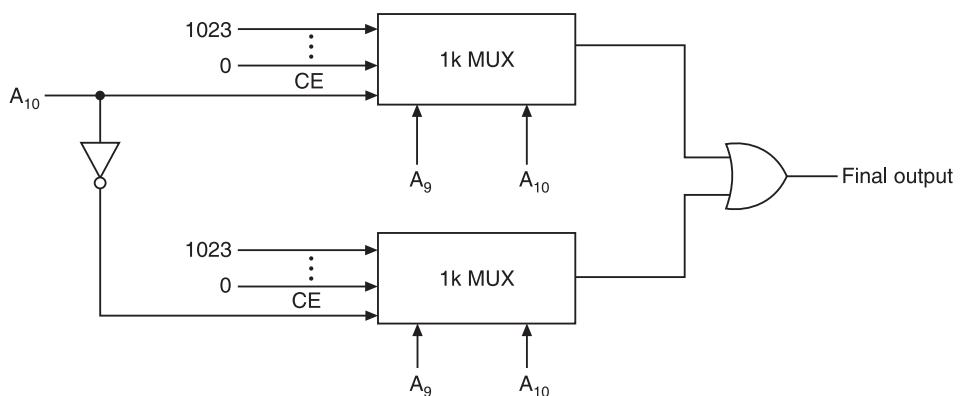


Figure 7.96 Modular expansion of a data selector.

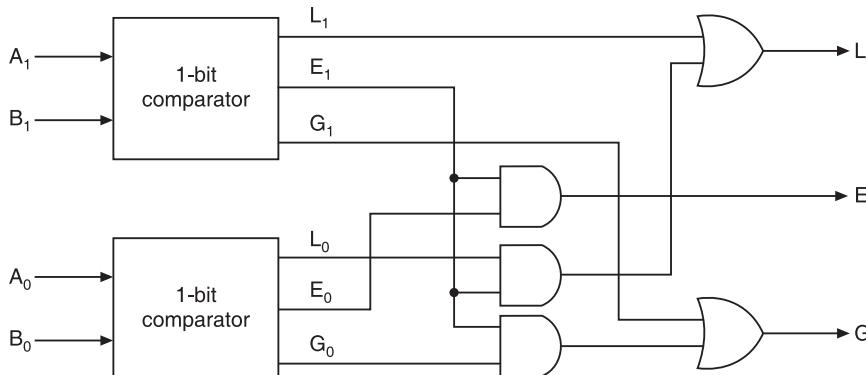
chip with  $A_{10}$ . Finally, the two outputs have to be mixed with an output OR gate. The additional hardware in this case is thus one inverter and one OR gate. A similar scheme is commonly used in expanding the size of memory by connecting additional modules.

### 7.27.5 Design of a 2-bit Comparator Using Two 1-bit Comparator Modules

Let  $A = A_1 A_0$  and  $B = B_1 B_0$  be the 2-bit numbers to be compared.  $A_1$  and  $B_1$  and  $A_0$  and  $B_0$  are compared separately using two 1-bit comparator modules. Then we know

$$\begin{aligned} & A > B \text{ if } A_1 > B_1 \quad \text{or} \quad A_1 = B_1 \text{ and } A_0 > B_0 \\ \therefore & G = G_1 + E_1 \cdot G_0 \\ & A = B \text{ if } A_1 = B_1 \text{ and } A_0 = B_0 \\ \therefore & E = E_1 \cdot E_0 \\ & A < B \text{ if } A_1 < B_1 \quad \text{or} \quad A_1 = B_1 \text{ and } A_0 < B_0 \\ \therefore & L = L_1 + E_1 \cdot L_0 \end{aligned}$$

So the 2-bit comparator using 1-bit modules is as shown in Figure 7.97.



**Figure 7.97** 2-bit comparator using two 1-bit modules.

### 7.27.6 Design of a 4-bit Comparator Using Four 1-bit Comparator Modules

Let  $A = A_3 A_2 A_1 A_0$  and  $B = B_3 B_2 B_1 B_0$  be the two 4-bit numbers to be compared.  $A_3$  and  $B_3$ ,  $A_2$  and  $B_2$ ,  $A_1$  and  $B_1$ , and  $A_0$  and  $B_0$  are compared separately using four 1-bit comparator modules. Then we know

$$\begin{aligned} & A > B \text{ if } A_3 > B_3 \text{ or } A_3 = B_3 \text{ and } A_2 > B_2 \text{ or } A_3 = B_3 \text{ and } A_2 = B_2 \text{ and } \\ & A_1 > B_1 \text{ or } A_3 = B_3 \text{ and } A_2 = B_2 \text{ and } A_1 = B_1 \text{ and } A_0 > B_0 \\ \text{Therefore, } & A > B: G = G_3 + E_3 G_2 + E_3 E_2 G_1 + E_3 E_2 E_1 G_0 \\ & A = B \text{ if } A_3 = B_3 \text{ and } A_2 = B_2 \text{ and } A_1 = B_1 \text{ and } A_0 = B_0 \\ \therefore & A = B: E = E_3 \cdot E_2 \cdot E_1 \cdot E_0 \\ & A < B \text{ if } A_3 < B_3 \text{ or } A_3 = B_3 \text{ and } A_2 < B_2 \text{ or } A_3 = B_3 \text{ and } A_2 = B_2 \text{ and } A_1 < B_1 \\ \text{or } & A_3 = B_3 \text{ and } A_2 = B_2 \text{ and } A_1 = B_1 \text{ and } A_0 < B_0 \\ \therefore & A < B: L = L_3 + E_3 L_2 + E_3 E_2 L_1 + E_3 E_2 E_1 L_0 \end{aligned}$$

So the 4-bit comparator using four 1-bit modules is shown in Figure 7.98.

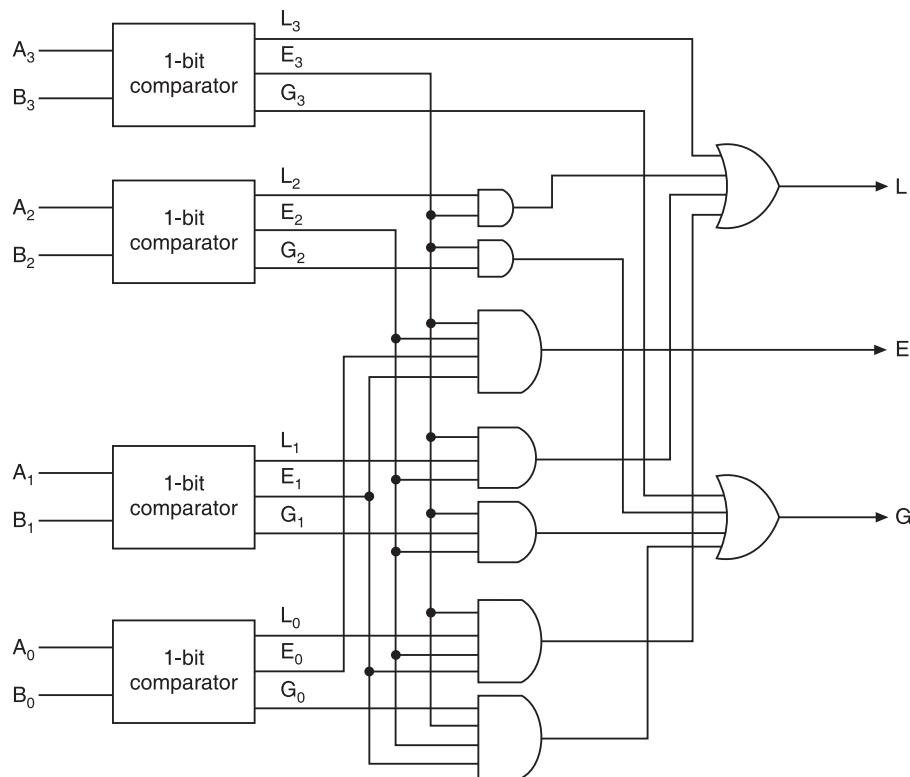


Figure 7.98 4-bit comparator using four 1-bit modules.

## 7.28 HAZARDS AND HAZARD-FREE REALIZATIONS

Hazards are unwanted switching transients that may appear at the output of a circuit because different paths exhibit different propagation delays. Such a transient is also called a *glitch* or a spurious *spike*, which is caused by hazardous behaviour of the logic circuit. Hazards occur in combinational circuits as well as in sequential circuits. A hazard in a combinational circuit is a condition where a single variable change produces a momentary output change when no output change should occur. There are two types of hazards, namely *static hazard* and *dynamic hazard*. "Static hazard is of two types, namely *static 1-hazard* and *static 0-hazard*. Suppose all the inputs to a logic circuit except one remain at their assigned levels and only one input say  $X$  changes from 0 to 1 or 1 to 0. If the output is expected to be at 1 regardless of the changing variable, the spurious 0 level for a short interval is called a static 1-hazard (shown in Figure 7.99d). If the output is expected to be at 0 regardless of the changing variable, the spurious 1 level for a short interval is called a static 0 hazard (shown in Figure 7.99e). Static hazards can be eliminated by using redundant gates. When the output changes three or more times when it should change from 1 to 0 or 0 to 1 only once, it is called dynamic hazard (shown in Figure 7.99f). When a circuit is implemented in

SOP with AND-OR gates or with NAND gates, the removal of static 1 (0) hazard guarantees that no static 0 (1) hazards or dynamic hazards will occur.

Dynamic hazards occur when the output changes for two adjacent input combinations. While changing, the output should change only once, but it may change three or more times in short intervals because of differential delays in several paths. Dynamic hazards occur only in multilevel circuits. A typical dynamic hazard is shown in Figure 7.99f. We consider only single input changes as in other cases it becomes almost impossible to prevent hazards.

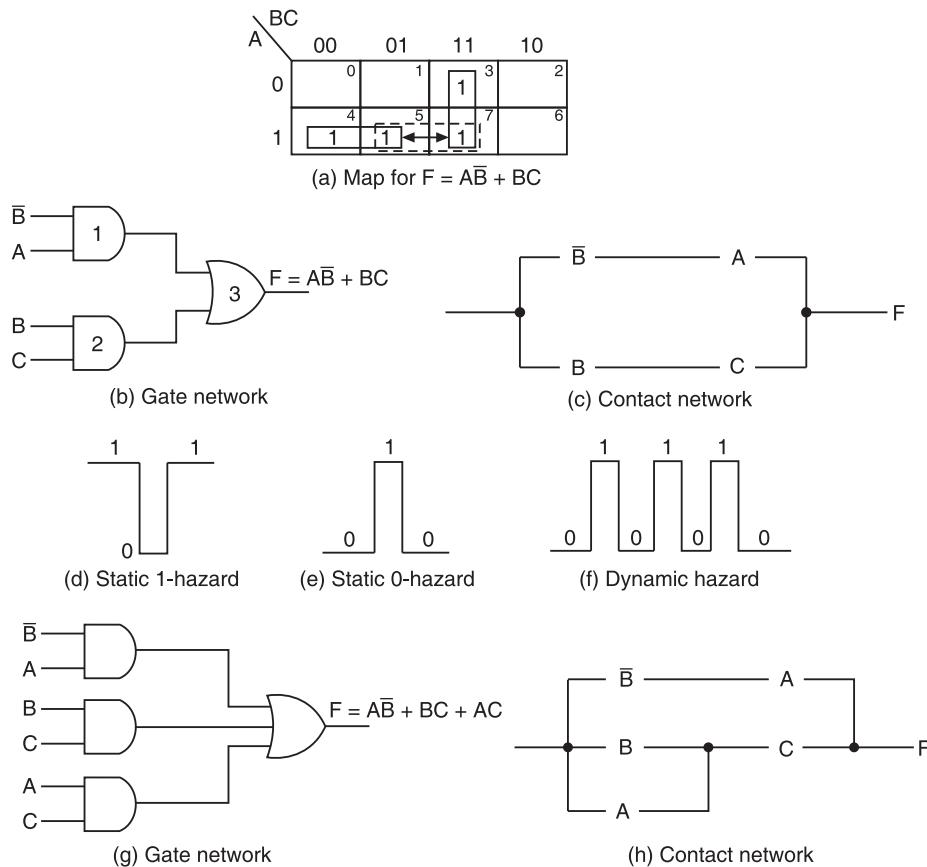


Figure 7.99 Networks containing static hazards.

### 7.28.1 Static Hazards

Consider the function  $F(A, B, C) = \Sigma m(3, 4, 5, 7)$  mapped in Figure 7.99a. The minimal SOP realization ( $F_{\min} = A\bar{B} + BC$ ) with logic gates is shown in Figure 7.99b, while the implementation with contact network is shown in Figure 7.99c.

Consider the situation when  $A = 1, B = 1, C = 1$  and only  $B$  is changing from 1 to 0. Look at the map. The output  $F$  has to remain at 1 by design. Look at the gate circuit. When  $B = 1$ , the output of gate 2 is 1, the output of gate 1 is 0 and the output  $F$  is 1. When  $B$  changes to 0, the output

of gate 2 becomes 0 but the output of gate 1 becomes 1 and the output F remains at 1. For the change in B from 1 to 0, if gate 1 responds faster than gate 2, F will be 1 as expected. If gate 2 is faster than gate 1, its output becomes 0 before the output of gate 1 changes to 1, and for a very short time the outputs of the gates 1 and 2 will be 0 resulting in an output of 0. A little later of course the output goes to 1. This erratic behaviour is shown in Figure 7.99d and is known as static 1-hazard marked by an arrow in the map. With contact networks it is called *tie set hazard*.

If we consider the POS realization of the same function, then  $F = \prod M(0, 1, 2, 6)$ . The minimal POS realization  $[F_{\min} = (A + B)(\bar{B} + C)]$  with logic gates is shown in Figure 7.100b.

Consider the situation when  $A = 0, B = 0, C = 0$  and only B changing from 0 to 1. Look at the map. The output F has to remain at 0 by design. Look at the gate circuit. When  $B = 0$ , the output of gate 1 is 0, the output of gate 2 is 1 and so the output F is 0. When B changes to 1, the output of gate 1 becomes 1 and the output of gate 2 becomes 0 and so the output F remains at 0. For the change in B from 0 to 1, if gate 2 responds faster than gate 1, F will be 0 as expected. If gate 1 is faster than gate 2, its output becomes 1 before the output of gate 2 changes to 0 and for a very short time the outputs of both the gates 1 and 2 will be 1 resulting in an output of 1. This erratic behaviour is shown in Figure 7.99e and is known as static 0-hazard marked by arrows in the map of Figure 7.100a. With contact networks it is called a *cut set hazard*.

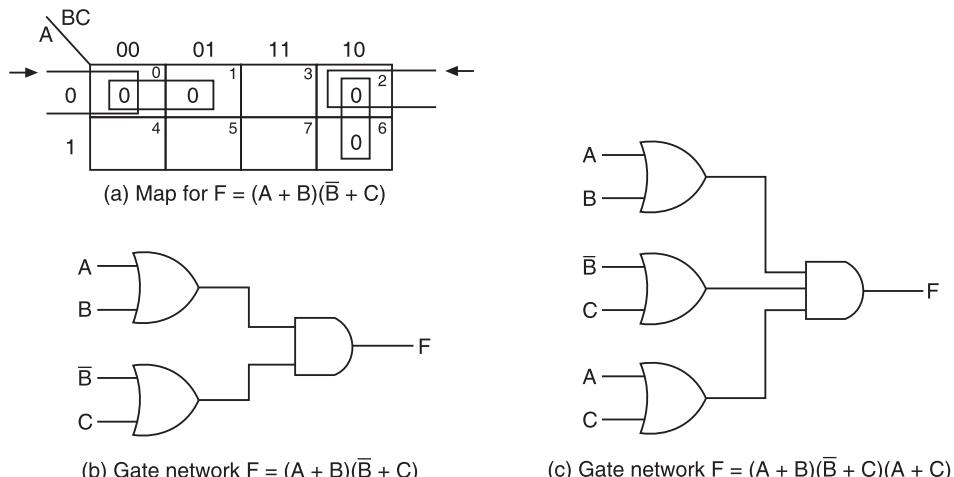
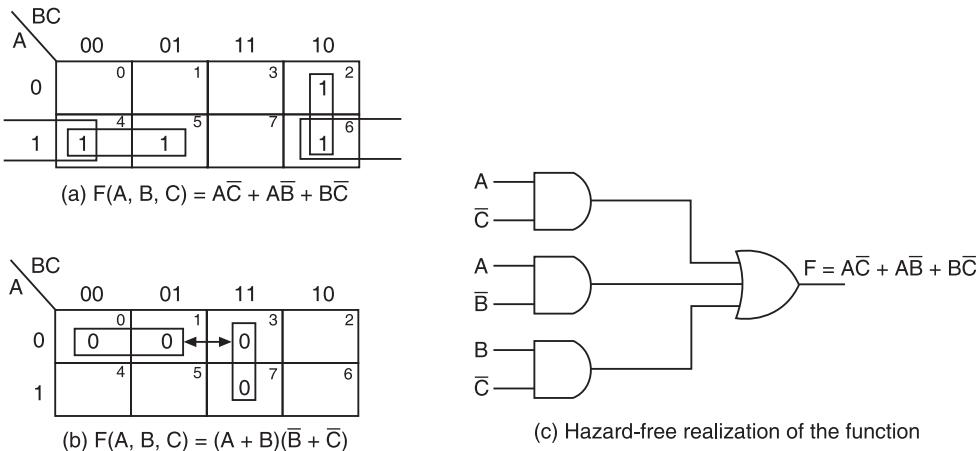


Figure 7.100 Networks containing static 0-hazard.

### 7.28.2 Hazard-free Realization

We should not allow the behaviour of circuits to depend on chance. We should ensure that the functions do not contain hazards. Hence hazard-free realization is necessary. Notice in the map of Figure 7.99a that the static 1-hazard arises because two adjacent 1s are covered by different subcubes (A cluster of  $2^m$  adjacent cells, each adjacent to  $m$  cells). Minimal realizations are more vulnerable to hazards. If we want to ensure that this pair of adjacent 1s is covered by the same subcube shown marked on the map, we need to add one more AND gate shown in Figure 7.99g. The corresponding contact networks will have one more path as shown in Figure 7.99h. This realization will now have no static 1-hazard but the function contains a redundant term  $BC$ .

The removal of static 1-hazards in a function by the addition of subcubes does not guarantee the removal of static 0-hazards in the function. This concept is illustrated in the following example.



**Figure 7.101** A function with no static 1-hazard but having static 0-hazard.

**EXAMPLE 7.21** Realize the switching function  $F(A, B, C) = \Sigma m(2, 4, 5, 6)$  by a hazard-free logic gate network.

### Solution

This function is mapped in Figure 7.101a. If we consider realization in SOP form, the function is expressed as

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

This SOP form is hazard-free because every pair of adjacent 1s is covered by the same subcube, and in order to satisfy this constraint, a redundant subcube  $A\bar{C}$  had to be added. Suppose we wish to realize the same function in POS form by choosing the subcubes marked in Figure 7.101b. There will be a hazard marked by arrows if you realize it as

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

Notice that merely expanding the POS form and ignoring the term  $B\bar{B} = 0$  results in the same SOP form as above, which is hazard free. The hazard in the POS form must be attributed to ignoring the terms like  $B\bar{B} = 0$ .

Naturally, we conclude that the hazard-free function has to be realized in its original form without resorting to simplification or factoring. The realization is shown in Figure 7.101c.

### 7.28.3 Essential Hazards

So far we have considered static and dynamic hazards. There is another type of hazard that may occur in asynchronous sequential circuit, called *essential hazard*. An essential hazard is caused by unequal delays along two or more paths that originate from the same input. An excessive delay through an inverter circuit in comparison to the delay associated with the feedback path may cause such a hazard. Essential hazards cannot be corrected by adding redundant gates as in static hazards.

## 410 FUNDAMENTALS OF DIGITAL CIRCUITS

The problem that they impose can be corrected by adjusting the amount of delay in the effected path. To avoid essential hazards, each feedback loop must be handled with individual care to ensure that the delay in the feedback loop is long enough compared to delays of other signals that originate from the input terminals. This problem tends to be specialized, as it depends on the particular circuit used and the amount of delays that are encountered in its various paths. In synchronous sequential machines, hazards caused by transient behaviour are no consequence as the clock speed is determined to allow all signals to settle in their steady static values before the next change of inputs.

### 7.29 MISCELLANEOUS EXAMPLES

**EXAMPLE 7.22** The inputs to a computer circuit are the 4 bits of the binary number  $A_3A_2A_1A_0$ . The circuit is required to produce a 1 if and only if all of the following conditions hold.

1. The MSB is a 1 or any of the other bits are a 0.
2.  $A_2$  is a 1 or any of the other bits are a 0.
3. Any of the 4 bits are a 0.

Obtain a minimal expression.

**Solution**

From the statement, the Boolean expression must be in the POS form given by

$$f = (A_3 + \bar{A}_2 + \bar{A}_1 + \bar{A}_0)(\bar{A}_3 + A_2 + \bar{A}_1 + \bar{A}_0)(\bar{A}_3 + \bar{A}_2 + \bar{A}_1 + \bar{A}_0)$$

where each non-complemented variable represents a 1 and the complemented variable a 0.

Since  $X \cdot X = X$ , the minimal expression is given by

$$\begin{aligned} f_{\min} &= (A_3 + \bar{A}_2 + \bar{A}_1 + \bar{A}_0)(\bar{A}_3 + A_2 + \bar{A}_1 + \bar{A}_0)(\bar{A}_3 + \bar{A}_2 + \bar{A}_1 + \bar{A}_0) \\ &= (\bar{A}_2 + \bar{A}_1 + \bar{A}_0)(\bar{A}_3 + \bar{A}_1 + \bar{A}_0) \\ &= (\bar{A}_2\bar{A}_3 + \bar{A}_1 + \bar{A}_0) \end{aligned}$$

**EXAMPLE 7.23** A staircase light is controlled by two switches, one is at the top of the stairs and the other at the bottom of the stairs.

- (a) Make a truth table for this system.
- (b) Write the logic equation in the SOP form.
- (c) Realize the circuit using AOI logic.
- (d) Realize the circuit using minimum number of (i) NAND gates and (ii) NOR gates.

**Solution**

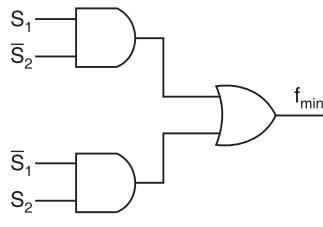
Let the switches be  $S_1$  and  $S_2$ . A staircase light is ON only when one of the switches is ON. It is OFF when both the switches are ON, or when both the switches are OFF. The truth table and the AOI logic diagram are shown in Figure 7.102.

The logic diagrams using NAND gates and NOR gates, respectively, are shown in Figure 7.103.

The given operation is XOR of  $S_1$  and  $S_2$ . Therefore, the logic equation is  $f = \bar{S}_1S_2 + S_1\bar{S}_2$ .

Truth table		
Inputs		Output
$S_1$	$S_2$	$f$
0	0	0
0	1	1
1	0	1
1	1	0

(a) Truth table

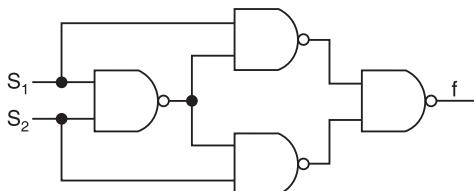


(b) AOI logic diagram

Figure 7.102 Example 7.23.

NAND logic

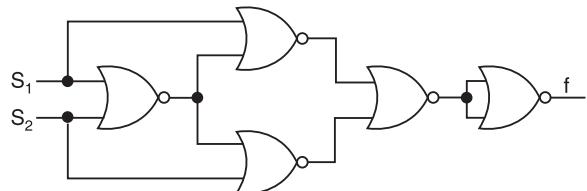
$$\begin{aligned}
 f &= S_1 \overline{S_2} + \overline{S_1} S_2 \\
 &= S_1 \overline{S_2} + \overline{S_1} S_2 + S_1 \overline{S_1} + S_2 \overline{S_2} \\
 &= S_1 (\overline{S_1} + \overline{S_2}) + S_2 (\overline{S_1} + \overline{S_2}) \\
 &= S_1 (\overline{S_1} \overline{S_2}) + S_2 (\overline{S_1} \overline{S_2}) \\
 &= \overline{S_1 \cdot (S_1 \cdot S_2)} \cdot \overline{S_2 \cdot (S_1 \cdot S_2)}
 \end{aligned}$$



(a) NAND logic

NOR logic

$$\begin{aligned}
 f &= S_1 \overline{S_2} + \overline{S_1} S_2 \\
 &= \overline{\overline{S_1} \overline{S_2}} + \overline{\overline{S_1} S_2} \\
 &= \overline{\overline{S_1} + S_2} + \overline{\overline{S_1} + \overline{S_2}} \\
 &= \overline{(S_1 + S_2)(S_2 + \overline{S_2})} + \overline{(S_1 + \overline{S_2})(S_1 + \overline{S_1})} \\
 &= \overline{S_2 + \overline{S_1} \cdot S_2} + \overline{S_1 + \overline{S_1} \cdot S_2} \\
 &= \overline{S_1 + S_2} + \overline{S_2 + S_1 + S_2}
 \end{aligned}$$



(b) NOR logic

Figure 7.103 Example 7.23.

**EXAMPLE 7.24** A safe has 5 locks v, w, x, y, and z; all of which must be unlocked for the safe to open. The keys to the locks are distributed among five executives in the following manner.

- Mr. A has keys for locks v and x.
- Mr. B has keys for locks v and y.
- Mr. C has keys for locks w and y.
- Mr. D has keys for locks x and z.
- Mr. E has keys for locks v and z.

- (a) Determine the minimal number of executives required to open the safe.
- (b) Find all the combinations of executives that can open the safe; write an expression  $f(A, B, C, D, E)$  which specifies when the safe can be opened as a function of what executives are present.
- (c) Who is the essential executive?

**Solution**

Table 7.3 indicates the executives and the locks they can open.

**Table 7.3** Example 7.24

Executive	Keys for locks				
	v	w	x	y	z
Mr. A	✓		✓		
Mr. B		✓			✓
Mr. C			✓	✓	
Mr. D				✓	✓
Mr. E	✓				✓

We see that the key for lock w is only with Mr. C. So Mr. C is the essential executive, without whom the safe cannot be opened. Once C is present, he can open lock y too. As seen from the table, the remaining locks v, x, and z can be opened by A and D or A and E or B and D or D and E. So the combinations of executives who can open the locks are CAD or CAE or CBD or CDE.

The Boolean expression corresponding to the above statement is

$$f(A, B, C, D, E) = CAD + CAE + CBD + CDE$$

The minimal number of executives required is 3.

**EXAMPLE 7.25** You are presented with a set of requirements under which an insurance policy can be issued. The applicant must be

1. a married female 25 years old or over, or
2. a female under 25, or
3. a married male under 25 who has not been involved in a car accident, or
4. a married male who has been involved in a car accident, or
5. a married male 25 years or over who has not been involved in a car accident.

Find an algebraic expression which assumes a value 1 whenever the policy is issued. Simplify the expression obtained.

**Solution**

Let the variables w, x, y, and z assume the truth value in the following cases.

w = 1, if the applicant has been involved in a car accident.

x = 1, if the applicant is married.

y = 1, if the applicant is a male.

z = 1, if the applicant is under 25.

The policy can be issued when any one of the conditions 1, 2, 3, 4, or 5 is met. The conditions 1, 2, 3, 4 and 5 are represented algebraically by  $x\bar{y}\bar{z}$ ,  $\bar{y}z$ ,  $xyz\bar{w}$ ,  $xyw$ ,  $xy\bar{z}\bar{w}$ . Therefore,

$$\begin{aligned} f(w, x, y, z) &= x\bar{y}\bar{z} + \bar{y}z + xyz\bar{w} + xyw + xy\bar{z}\bar{w} \\ &= xy\bar{w}(z + \bar{z}) + xyw + \bar{y}(z + x\bar{z}) \\ &= xy\bar{w} + xyw + \bar{y}(z + \bar{z})(z + x) \end{aligned}$$

$$\begin{aligned}
 &= xy(w + \bar{w}) + (z + x)\bar{y} \\
 &= xy + x\bar{y} + \bar{y}z \\
 &= x(y + \bar{y}) + \bar{y}z \\
 &= x + \bar{y}z
 \end{aligned}$$

So the policy can be issued if the applicant is either married or is a female under 25.

**EXAMPLE 7.26** An air-conditioning unit is controlled by four variables: temperature T, humidity H, the time of the day D, and the day of the week W. The unit is turned on under any of the following circumstances.

1. The temperature exceeds 78° F, and the time of the day is between 8 a.m. and 5 p.m.
2. The humidity exceeds 85%, the temperature exceeds 78°F, and the time of day is between 8 a.m. and 5 p.m.
3. The humidity exceeds 85%, the temperature exceeds 78°F, and it is a weekend.
4. It is Saturday or Sunday and humidity exceeds 85%.

Write a logic expression for controlling the air-conditioning unit. Simplify the expression obtained as far as possible.

### **Solution**

Define the variables:

- (a) T = 1, if the temperature exceeds 78°F.
- (b) H = 1, if the humidity exceeds 85%.
- (c) D = 1, if the time of the day is between 8 a.m. and 5 p.m.
- (d) W = 1, if it is weekend, i.e. Saturday or Sunday.

The circumstances 1, 2, 3 and 4, respectively, are then given algebraically as TD, HTD, HTW and WH. Therefore, the Boolean expression for turning on the machine is

$$\begin{aligned}
 f &= TD + HTD + HTW + WH \\
 &= TD(1 + H) + HW(1 + T) = TD + HW
 \end{aligned}$$

So the air-conditioning unit is turned on, if the temperature exceeds 78°F and the time of the day is between 8 a.m. and 5 p.m., or if it is a weekend and humidity exceeds 85%.

**EXAMPLE 7.27** Five soldiers A, B, C, D and E volunteer to perform an important military task if their following conditions are satisfied.

1. Either A or B or both must go.
2. Either C or E but not both must go.
3. Either both A and C go or neither goes.
4. If D goes, then E must also go.
5. If B goes, then A and C must also go.

Define the variables A, B, C, D and E, so that an unprimed variable will mean that the corresponding soldier has been selected to go. Determine the expression which specifies the combinations of volunteers who can get the assignment.

### **Solution**

Analyzing the problem to perform the task, the first condition is, either A or B or both must go.

**Case 1.** Suppose A goes, then according to condition 3, C must also go. If C goes, then according to condition 2, E cannot go. Then according to condition 4 when E is not going, D also cannot go. So D does not go. So A and C can go to perform the task.

**Case 2.** When B goes, according to condition 5, A and C must go. When C goes, E cannot go, and when E cannot go, D also cannot go. So the second combination of soldiers who can perform the task is ABC.

**Case 3.** When both A and B go, C has to go. When C goes, E and therefore D cannot go. This is the same as the second combination ABC.

So the conclusion is either A and C, or A, B and C can go and perform the military task. Therefore,

$$\begin{aligned} f &= AC + ABC \\ &= AC(1 + B) = AC \end{aligned}$$

So the minimal combination of soldiers who can get the assignment is A and C.

**EXAMPLE 7.28** A lawn-sprinkling system is controlled automatically by certain combinations of the following variables.

Season (S = 1, if summer; 0, otherwise)

Moisture content of soil (M = 1, if high; 0, if low)

Outside temperature (T = 1, if high; 0, if low)

Outside humidity (H = 1, if high; 0, if low)

The sprinkler is turned on under any of the following circumstances.

1. The moisture content is low in winter.
2. The temperature is high and the moisture content is low in summer.
3. The temperature is high and the humidity is high in summer.
4. The temperature is low and the moisture content is low in summer.
5. The temperature is high and the humidity is low.

Use a K-map to find the simplest possible logic expression involving the variables S, M, T and H for turning on the sprinkler system.

### Solution

The given circumstances 1, 2, 3, 4 and 5 are expressed in terms of the defined variables S, M, T, and H as  $\bar{M}\bar{S}$ ,  $T\bar{M}S$ ,  $THS$ ,  $\bar{T}\bar{M}S$ , and  $T\bar{H}$ , respectively.

The Boolean expression is

$$\begin{aligned} f &= \bar{S}\bar{M} + S\bar{M}T + STH + S\bar{M}\bar{T} + T\bar{H} \\ &= 0\ 0\ X\ X + 1\ 0\ 1\ X + 1\ X\ 1\ 1 + 1\ 0\ 0\ X + X\ X\ 1\ 0 \end{aligned}$$

The expressions in terms of minterms and maxterms are

$$\begin{aligned} f &= \sum m(0, 1, 2, 3, 6, 8, 9, 10, 11, 14, 15) \\ &= \prod M(4, 5, 7, 12, 13) \end{aligned}$$

The K-maps in SOP and POS forms, their minimization, the minimal expressions obtained from each, and the logic diagram in the SOP form are all shown in Figure 7.104. Both SOP and POS forms give the same minimum.

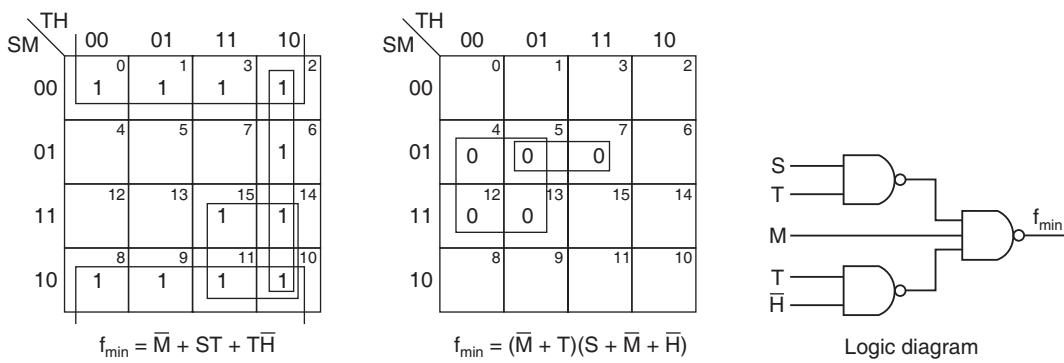


Figure 7.104 Example 7.28.

### SHORT QUESTIONS AND ANSWERS

- 1. What are arithmetic circuits?**
  - A.** Arithmetic circuits are the circuits that perform arithmetic operations.
- 2. What is a half-adder?**
  - A.** A half-adder is an arithmetic circuit that adds two binary digits. It has two inputs and two outputs (sum and carry).
- 3. What is a full-adder?**
  - A.** A full-adder is an arithmetic circuit that adds two binary digits and a carry, i.e. three bits. It has three inputs and two outputs (sum and carry).
- 4. What is the disadvantage of realizing a full-adder using two half-adders?**
  - A.** The disadvantage of realizing a full-adder using two half-adders is that, in this, the bits must propagate through several gates in succession, which makes the propagation delay greater than that of the full-adder circuit using AOI logic.
- 5. What is a half-subtractor?**
  - A.** A half-subtractor is an arithmetic circuit that subtracts one binary digit from another. It has two inputs and two outputs (difference and borrow).
- 6. What is a full-subtractor?**
  - A.** A full-subtractor is an arithmetic circuit that subtracts one binary digit from another considering a borrow. It has three inputs and two outputs (difference and borrow).
- 7. Why are subtractor ICs not available?**
  - A.** Since subtraction is performed using adders by making use of 1's and 2's complement methods, separate subtraction ICs are not available.
- 8. What is a parallel adder?**
  - A.** A parallel adder is an arithmetic circuit that adds two numbers in parallel form.
- 9. What is carry propagation delay of a full-adder?**
  - A.** The carry propagation delay of a full-adder in a parallel adder is the time between the application of the carry-in and the occurrence of the carry-out.

## 416 FUNDAMENTALS OF DIGITAL CIRCUITS

10. What do you mean by cascading of parallel adders? Why is it required?
  - A. Connecting the parallel adders in series, i.e. connecting the carry out of one parallel adder to the carry-in of another parallel adder is called cascading them. It is required when a large number of bits are to be added.
11. How is addition of large binary numbers accomplished?
  - A. The addition of large binary numbers can be accomplished by cascading two or more parallel adder chips.
12. What is a ripple-carry-adder?
  - A. A ripple-carry-adder is a parallel adder in which the carry-out of each full adder is the carry-in to the next most significant adder.
13. How does the look-ahead-carry adder speed up the addition process?
  - A. The look-ahead-carry adder speeds up the addition process by eliminating the ripple carry delay. It examines all the input bits simultaneously and also generates the carry-in bits for all the stages simultaneously.
14. When is carry generated and when is carry propagated in a look-ahead-carry adder?
  - A. In a look-ahead-carry adder, a carry-out is generated when both the input bits are 1. A carry-in may be propagated by the full-adder when either or both of the input bits are 1.
15. What is a serial adder?
  - A. A serial adder is a sequential circuit used to add serial binary numbers.
16. Why does a serial adder require only one full-adder?
  - A. A serial adder requires only one full-adder because in this the bits are added serially, i.e. one pair of bits at a time.
17. What is the drawback of serial adders? For which applications are they preferred?
  - A. The drawback of the serial adders is that serial adders are slower than parallel adders. They are preferred for applications where circuit minimization is more important than speed as in pocket calculators.
18. Why serial adders are slower than parallel adders?
  - A. Serial adders are slower than parallel adders because they require one clock pulse for each pair of bits added.
19. What are the differences between serial and parallel adders?
  - A. The parallel adder uses registers with parallel load, whereas the serial adder uses shift registers. The number of full-adder circuits in the parallel adder is equal to the number of bits in the binary numbers, whereas the serial adder requires only one full-adder circuit and a carry flip-flop. Excluding the registers, the parallel adder is a combinational circuit, whereas the serial adder is a sequential circuit. The sequential circuit in the serial adder consists of a full-adder and a flip-flop that stores the output carry.
20. In what way is a BCD adder different from a binary adder?
  - A. While adding BCD numbers, the output is required to be corrected which is not required in the case of binary adders.
21. Compare the hardware requirements of a BCD arithmetic unit and a straight binary arithmetic unit.
  - A. Because of the need for correction circuit, the BCD arithmetic unit requires more hardware than the straight binary arithmetic unit.
22. Why is decimal 6 required to be added in a BCD adder if the sum is not a valid BCD number?
  - A. 16 possible combinations are there with 4 bit numbers. In BCD only 10 of these are used and the other 6 are skipped. That is why 6 is required to be added.

**23.** What are code converters?

- A. Code converters are logic circuits whose inputs are bit patterns representing numbers or characters in one code and whose outputs are the corresponding representations in a different code.

**24.** What is a parity bit generator?

- A. A parity bit generator is a digital circuit that generates a bit called the parity bit to be added to the data bits.

**25.** What is the basic principle used in order to check or generate the proper parity bit in a given code word?

- A. The basic principle used in order to check or generate the proper parity bit in a given code word is ‘the modulo sum of an even number of 1s is always a 0 and the modulo sum of an odd number of 1s is always a 1’.

**26.** How is even parity bit generated for four data bits?

- A. To generate an even parity bit for four data bits, the four data bits are added using three X-OR gates. The sum bit will be the parity bit.

**27.** How is odd parity bit generated for four data bits?

- A. To generate an odd parity bit for four data bits, the four data bits are added using three X-OR gates and the sum bit is inverted.

**28.** What is a comparator?

- A. A comparator is a logic circuit that compares the magnitudes of two binary numbers.

**29.** Which logic gate is a basic comparator?

- A. The X-NOR gate (coincidence gate) is a basic comparator.

**30.** When are two binary numbers equal?

- A. Two binary numbers are equal, if and only if all their corresponding bits coincide.

**31.** What is an encoder?

- A. An encoder is a device whose inputs are decimal digits and/or alphabetic characters and whose outputs are the coded representations of those inputs.

**32.** What is encoding?

- A. Encoding is a process of converting familiar numbers or symbols into a coded format.

**33.** What is arbitration?

- A. In some practical applications, priority encoders may have several inputs that are routinely high at the same time, and the principle function of the encoder in those cases is to select the input with the highest priority. This function is called arbitration.

**34.** What is a priority encoder?

- A. A priority encoder is a logic circuit that responds to just one input, in accordance with some priority system, among those that may be simultaneously high.

**35.** The most common priority system is based on what?

- A. The most common priority system is based on the relative magnitudes of the inputs: whichever decimal digit is the largest is the one encoded.

**36.** What is a decoder?

- A. A decoder is a logic circuit that converts an  $n$ -input binary code into a corresponding single numeric output code. In other words, a decoder is a device that identifies or recognizes or detects a particular code.

## 418 FUNDAMENTALS OF DIGITAL CIRCUITS

37. Why a binary-to-octal decoder is called a 1-of-8 decoder?
- A. A binary-to-octal decoder is called a 1-of-8 decoder because only one of the eight outputs is activated at one time.
38. What for are enable inputs used in a decoder?
- A. Enable inputs are used to control the operation of the decoder.
39. What is a multiplexer (MUX)?
- A. A multiplexer or data selector is a logic circuit that accepts several data inputs and allows only one of them at a time to get through to the outputs. It is a N:1 device.
40. What is multiplexing?
- A. Multiplexing means sharing. Selecting 1-out of-N input data sources and transmitting the selected data to a single output channel is called multiplexing.
41. How many types of multiplexing are there? Name them.
- A. There are two types of multiplexing. They are time division multiplexing and frequency division multiplexing.
42. Why is a multiplexer called a data selector?
- A. A multiplexer is called a data selector because it accepts several data inputs and allows only one of them at a time to get through to the output.
43. What are the applications of multiplexers?
- A. Multiplexers are used for data selection, data routing, operation sequencing, parallel-to-serial conversion, waveform generation, logic function generation, etc.
44. Can a multiplexer be used to realize a logic function? If yes, in what ways this realization is better than realization using gates.
- A. Yes. A multiplexer can be used to realize a logic function.  
This realization is better in the following ways.  
(a) Simplification of logic function is not required.  
(b) IC package count is reduced.  
(c) Reliability of the system is improved because of external wired connections.  
(d) It is very easy to change the logic function implemented if and when redesign of a system becomes necessary.
45. What is the minimum number of selection lines required for selecting one out of  $n$  input lines?
- A. The minimum number of selection lines  $m$  required for selecting one out of  $n$  input lines is  $m = \log n/\log 2$ .
46. Can a ‘strobe’ or ‘enable’ input of a multiplexer be used to increase its size.
- A. Yes. A ‘strobe’ or ‘enable’ input of a multiplexer can be used to increase its size.
47. Why is time multiplexing used in display system?
- A. Time multiplexing is used in display systems mainly to conserve power.
48. What is a demultiplexer (DMUX)?
- A. A demultiplexer is a logic circuit that depending on the status of its select inputs, channels its data input to one of the several data outputs.
49. Why is a demultiplexer called a distributor?
- A. A demultiplexer can be thought of as a distributor since it takes a single input and distributes it over several outputs.

**50.** Compare a decoder with a demultiplexer.

- A. A demultiplexer has one data input,  $m$  select lines, and  $n$  output lines. A decoder, on the other hand, does not have the data input, but the select lines are used as input lines.

**51.** Can a demultiplexer be used as a logic element? If yes, what are its advantages over realization using gates?

- A. Yes. A demultiplexer can be used as a logic element. Its advantages over realization using gates are as follows.

- (a) Simplification of logic function is not required.
- (b) IC package count is reduced especially in multi-output circuits.
- (c) Reliability of the system is improved.

**52.** What is the type of display used in calculators?

- A. The type of display used in calculators is 7-segment LED/LCD.

**53.** Is the display used in a digital wristwatch LED or LCD? Why?

- A. The display used in digital wristwatches is usually LCD. It is because LCD requires significantly less power than LED.

**54.** What is a driver IC?

- A. A driver IC is an IC whose outputs can operate with higher current and/or voltage limits than those of a normal standard IC.

**55.** What do you mean by a module?

- A. Modules are integrated circuits which are fabricated in solid state form and are referred to as IC chips.

**56.** What is modular design?

- A. The design using ICs is called modular design.

**57.** What are hazards? Why do they occur?

- A. Hazards are unwanted switching transients. They may appear at the output of a circuit because different paths exhibit different propagation delays.

**58.** What is a glitch?

- A. A glitch or a spurious spike is a transient caused by hazardous behaviour of the logic circuit.

**59.** What is hazard in a combinational circuit?

- A. A hazard in a combinational circuit is a condition where a single variable change produces a momentary output change when no output change should occur.

**60.** Hazards are of how many types? Name them.

- A. Hazards are of two types. They are:

- (a) Static hazards, and
- (b) Dynamic hazards.

**61.** Static hazards are of how many types? Name them.

- A. Static hazards are of two types. They are:

- (a) Static 1-hazard, and
- (b) Static 0-hazard.

**62.** What is static-1 hazard? What is static-0 hazard?

- A. When only one of the input variable of a circuit changes from 0 to 1 or 1 to 0—if the output is expected to be at 1 regardless of the changing variable, the spurious 0 level for a short interval is called a static 1 hazard. If the output is expected to be at 0 regardless of the changing variable the spurious 1 level for a short interval is called a static 0 hazard.

- 63.** How are static hazards eliminated?  
A. Static hazards can be eliminated using redundant gates.
- 64.** What is dynamic hazard? When do they occur?  
A. The change in output three or more times when it should change from 1 to 0 or 0 to 1 only once is called dynamic hazard. Dynamic hazards occur only in multi level circuits. They occur when the output changes for two adjacent input combinations.
- 65.** What is a tie set hazard?  
A. With contact networks static-1 hazard is called a tie set hazard.
- 66.** What is cut set hazard?  
A. With contact networks static-0 hazard is called a cut set hazard.

### REVIEW QUESTIONS

- 1.** Distinguish between a half-adder and a full-adder.
- 2.** Distinguish between a half-subtractor and a full-subtractor.
- 3.** Realize a half-adder using (a) only NAND gates and (b) only NOR gates.
- 4.** Realize a half-subtractor using (a) only NAND gates and (b) only NOR gates.
- 5.** Realize a full-adder using (a) only NAND gates and (b) only NOR gates.
- 6.** Realize a full-subtractor using (a) only NAND gates and (b) only NOR gates.
- 7.** Distinguish between a serial adder and a parallel adder.
- 8.** With the help of a block diagram explain the working of a serial adder.
- 9.** Give the implementation of a 4-bit ripple adder using half-adder(s)/full-adder(s).
- 10.** Realize a look-ahead-carry adder.
- 11.** With the help of a logic diagram explain a parallel adder/subtractor using 2's complement system.
- 12.** Explain the working of a BCD adder.
- 13.** Realize a single bit comparator.
- 14.** Realize a 2-bit comparator.
- 15.** Realize a 4-bit comparator.
- 16.** Write notes on code converters.
- 17.** Write notes on parity bit generators.
- 18.** Distinguish between an encoder and a decoder.
- 19.** With the help of a logic diagram and a truth table, explain an octal-to-binary encoder.
- 20.** With the help of a gate level logic diagram and a truth table, explain a decimal-to-BCD encoder.
- 21.** Explain a keyboard encoder using diode matrix.
- 22.** With the help of a logic diagram and a truth table, explain a 3-line to 8-line decoder.
- 23.** With the help of a logic diagram and a truth table, explain a BCD-to-decimal decoder.
- 24.** Write notes on BCD-to-7 segment decoders.
- 25.** Distinguish between a multiplexer and a demultiplexer.
- 26.** Discuss a few applications of multiplexers.
- 27.** With the help of logic diagram and function table explain (a) a 4-input multiplexer and (b) an 8-input multiplexer.

28. Explain how a 4-variable function can be realized using an 8:1 mux.
29. Show an arrangement to obtain a 16-input multiplexer from two 8-input multiplexers.
30. With the help of a logic diagram and truth table explain (a) a 1-line to 4-line demultiplexer and (b) a 1-line to 8-line demultiplexer.

**FILL IN THE BLANKS**

1. An arithmetic circuit that adds only two binary digits is called a \_\_\_\_\_.
2. An arithmetic circuit that adds two binary digits and a carry is called a \_\_\_\_\_.
3. An arithmetic circuit that subtracts one binary digit from another without considering a borrow is called a \_\_\_\_\_.
4. An arithmetic circuit that subtracts one binary digit from another considering a borrow is called a \_\_\_\_\_.
5. An adder that adds two numbers in parallel form and produces the sum bits in parallel form is called a \_\_\_\_\_.
6. A parallel adder in which the carry-out of each full-adder is the carry-in to the next most significant adder is called a \_\_\_\_\_.
7. The \_\_\_\_\_ adder speeds up the process by eliminating the ripple carry.
8. \_\_\_\_\_ adders are used where circuit minimization is more important than speed as in pocket calculators.
9. A \_\_\_\_\_ is a logic circuit that compares the magnitudes of two binary numbers.
10. A \_\_\_\_\_ is a logic circuit that converts an  $n$ -input binary code into a corresponding single numeric output code.
11. \_\_\_\_\_ inputs are used to control the operation of the decoder.
12. A device whose inputs are decimal digits and /or alphabetic characters and whose outputs are the coded representations of those inputs is called \_\_\_\_\_.
13. A \_\_\_\_\_ is a logic circuit that responds to just one input, in accordance with some priority system, among those that may be simultaneously high.
14. A \_\_\_\_\_ is a logic circuit that accepts several data inputs and allows only one of them at a time to get through to the output.
15. A 4-variable logic expression can be realized using a single \_\_\_\_\_ multiplexer.
16. A \_\_\_\_\_ is a logic circuit that depending on the status of the select inputs, channels its data input to one of several data outputs.
17. A demultiplexer can be thought of as a \_\_\_\_\_.
18. The \_\_\_\_\_ gate is a basic comparator.
19. \_\_\_\_\_ is a process of converting familiar numbers or symbols into a coded format.
20. \_\_\_\_\_ means sharing.
21. \_\_\_\_\_ and \_\_\_\_\_ are the two types of multiplexing.
22. A \_\_\_\_\_ identifies or recognizes or detects a particular code.
23. A decoder with 64 output lines has \_\_\_\_\_ select lines.
24. A binary-to-octal decoder is a \_\_\_\_\_ line to \_\_\_\_\_ line decoder.
25. A 3-line to 8-line decoder is referred to as \_\_\_\_\_ decoder or \_\_\_\_\_ decoder.

26. A BCD-to-decimal decoder is a \_\_\_\_\_ line to \_\_\_\_\_ line decoder.
27. A BCD-to-decimal decoder is referred to as a \_\_\_\_\_ to \_\_\_\_\_ decoder, or a \_\_\_\_\_ of \_\_\_\_\_ decoder.
28. An octal-to-binary encoder is a \_\_\_\_\_ line to \_\_\_\_\_ encoder.
29. A decimal-to-BCD encoder is a \_\_\_\_\_ line to \_\_\_\_\_ line encoder.
30. A half-adder can be realized using at least \_\_\_\_\_ NAND gates or NOR gates.
31. A half-subtractor can be realized by using at least \_\_\_\_\_ NAND gates or NOR gates.
32. A full-adder and a full-subtractor can be realized by using at least \_\_\_\_\_ NAND gates.
33. A full-adder and full-subtractor can be realized by using at least \_\_\_\_\_ NOR gates.

### OBJECTIVE TYPE QUESTIONS

1. The difference output in a full-subtractor is the same as the  
 (a) difference output of a half-subtractor      (b) sum output of a half-adder  
 (c) sum output of a full-adder      (d) carry output of a full-adder
2. Which of the following logic circuits accepts two binary digits on inputs, and produces two binary digits, a sum bit and a carry bit on its outputs?  
 (a) full-adder      (b) half-adder      (c) serial adder      (d) parallel adder
3. How many inputs and outputs does a full-adder have?  
 (a) two inputs, two outputs      (b) two inputs, one output  
 (c) three inputs, two outputs      (d) two inputs, three outputs
4. How many inputs and outputs does a full-subtractor circuit have?  
 (a) two inputs, one output      (b) two inputs, two outputs  
 (c) two inputs, three outputs      (d) three inputs, two outputs
5. A full-adder can be realized using  
 (a) one half-adder, two OR gates      (b) two half-adders, one OR gate  
 (c) two half-adders, two OR gates      (d) two half-adders, one AND gate
6. The minimum number of 2-input NAND/NOR gates required to realize a half-adder is  
 (a) 3      (b) 4      (c) 5      (d) 6
7. The minimum number of 2-input NAND/NOR gates required to realize a half-subtractor is  
 (a) 3      (b) 4      (c) 5      (d) 6
8. The minimum number of 2-input NAND gates required to realize a full-adder/full-subtractor is  
 (a) 8      (b) 9      (c) 10      (d) 12
9. The minimum number of 2-input NOR gates required to realize a full-subtractor is  
 (a) 8      (b) 9      (c) 10      (d) 12
10. How many full-adders are required to construct an  $m$ -bit parallel adder?  
 (a)  $m/2$       (b)  $m - 1$       (c)  $m$       (d)  $m + 1$
11. Parallel adders are  
 (a) combinational logic circuits      (b) sequential logic circuits  
 (c) both of the above      (d) none of the above

- 12.** In which of the following adder circuits is the carry ripple delay eliminated?  
 (a) half-adder      (b) full-adder      (c) parallel adder      (d) carry-look-ahead adder
- 13.** To secure a higher speed of addition, which of the following is the preferred solution?  
 (a) serial adder      (b) parallel adder  
 (c) adder with a look-ahead-carry      (d) full-adder
- 14.** A parallel adder in which the carry-out of each full-adder is the carry-in to the next significant digit adder is called a  
 (a) ripple carry adder      (b) look-ahead-carry adder  
 (c) serial carry adder      (d) parallel carry adder
- 15.** The adder preferred for applications where circuit minimization is more important than speed is  
 (a) parallel adder      (b) serial adder      (c) full-adder      (d) half-adder
- 16.** A serial adder requires only one  
 (a) half-adder      (b) full-adder      (c) counter      (d) multiplexer
- 17.** In digital systems subtraction is performed  
 (a) using half-adders      (b) using half-subtractors  
 (c) using adders with 1's complement representation of negative numbers  
 (d) by none of the above.
- 18.** In a digital system BCD arithmetic is preferred to normal binary arithmetic because  
 (a) BCD arithmetic circuits are simpler than binary arithmetic circuits  
 (b) BCD arithmetic circuits are faster than binary arithmetic circuits  
 (c) BCD arithmetic circuits are less expensive than binary arithmetic circuits  
 (d) of ease of operation when input is in BCD format and the output display is decimal
- 19.** In BCD addition, 0110 is required to be added to the sum for getting the correct result, if  
 (a) the sum of two BCD numbers is not a valid BCD number  
 (b) the sum of two BCD numbers is not a valid BCD number or a carry is produced  
 (c) a carry is produced  
 (d) none of the above is true
- 20.** BCD subtraction is performed by using  
 (a) 1's complement representation      (b) 2's complement representation  
 (c) 5's complement representation      (d) 9's complement representation
- 21.** Which logic gate is a basic comparator?  
 (a) NOR gate      (b) NAND gate      (c) X-OR gate      (d) X-NOR gate
- 22.** The logic gate used in parity checkers is  
 (a) NAND gate      (b) NOR gate      (c) X-OR gate      (d) X-NOR gate
- 23.** A device whose inputs are decimal digits and/or alphabetic characters and whose outputs are the coded representations of those inputs is called  
 (a) an encoder      (b) a decoder      (c) a code converter      (d) a decimal converter
- 24.** A logic circuit that responds to just one input, in accordance with some priority system, among those that may be simultaneously high is called  
 (a) an encoder      (b) a priority encoder      (c) a priority decoder      (d) a decoder



## PROBLEMS

- 7.1 Design an 8421-to-2421 BCD code converter and draw its logic diagram.
  - 7.2 Find the simplest possible logic expressions for a 2421-to-51111 BCD code converter.
  - 7.3 Draw a logic diagram for an excess-3-to-decimal decoder. Inputs and outputs should be active HIGH.



## VHDL PROGRAMS

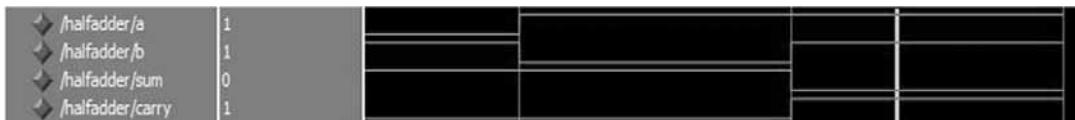
### 1. VHDL PROGRAM FOR HALF-ADDER USING X-OR GATE AND AND GATE

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity HALFADDER is
    Port ( A,B: in STD_LOGIC;
           SUM,CARRY: out STD_LOGIC);
end HALFADDER;
architecture Structural of HALFADDER is
component XORGATE is
    Port ( A,B: in STD_LOGIC;
           Y: out STD_LOGIC);
end component;
component ANDGATE is
    Port ( A,B: in STD_LOGIC;
           Y: out STD_LOGIC);
end component;
begin
    x1: XORGATE port map(A,B,SUM);
    x2: ANDGATE port map(A,B,CARRY);
end Structural;

```

#### SIMULATION OUTPUT:



### 2. VHDL PROGRAM FOR HALF-SUBTRACTOR USING X-OR GATE AND AND GATE

```

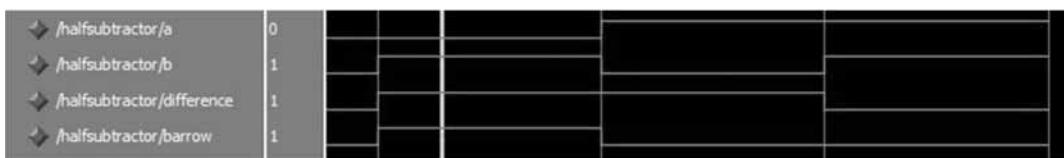
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity HALFSUBTRACTOR is
    Port ( A,B: in STD_LOGIC;
           DIFFERENCE,BARROW: out STD_LOGIC);
end HALFSUBTRACTOR;
architecture Structural of HALFSUBTRACTOR is
component XORGATE is
    Port ( A,B: in STD_LOGIC;
           Y: out STD_LOGIC);
end component;

```

```

component ANDGATE is
    Port ( A,B: in STD_LOGIC;
            Y: out STD_LOGIC);
end component;
component NOTGATE is
    Port ( A : in STD_LOGIC;
            Y : out STD_LOGIC);
end component;
signal abar:STD_LOGIC;
begin
x1: XORGATE port map(A,B,DIFFERENCE);
x2: NOTGATE port map(A,abar);
x3: ANDGATE port map(abar,B,BARROW);
end Structural;

```

**SIMULATION OUTPUT:****3. VHDL PROGRAM FOR FULL-ADDER USING TWO HALF-ADDER AND OR GATE**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity FULLADDER is
    Port ( A,B,C: in STD_LOGIC;
            SUM,CARRY: out STD_LOGIC);
end FULLADDER;
architecture Structural of FULLADDER is
component HALFADDER is
    Port ( A,B: in STD_LOGIC;
            SUM,CARRY: out STD_LOGIC);
end component;
component XORGATE is
    Port ( A,B: in STD_LOGIC;
            Y: out STD_LOGIC);
end component;
signal sum1, carry2, carry1:STD_LOGIC;
begin
x1: HALFADDER port map(A,B,sum1, carry1);

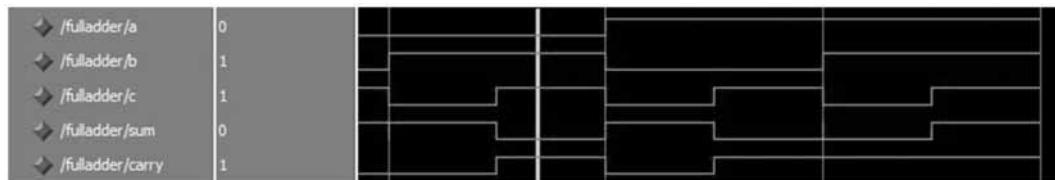
```

```

x2: HALFADDER port map(sum1,C,SUM,carry2);
x3: XORGATE port map(carry1,carry2,CARRY);
end Structural;

```

#### SIMULATION OUTPUT:



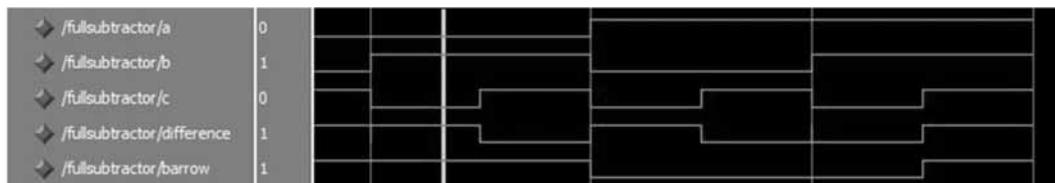
#### 4. VHDL PROGRAM FOR FULL-SUBTRACTOR USING TWO HALF-SUBTRACTORS AND OR GATE

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity FULLSUBTRACTOR is
    Port ( A,B,C: in STD_LOGIC;
           DIFFERENCE,BARROW: out STD_LOGIC) ;
end FULLSUBTRACTOR;
architecture Structural of FULLSUBTRACTOR is
component HALFSUBTRACTOR is
    Port ( A,B: in STD_LOGIC;
           DIFFERENCE,BARROW: out STD_LOGIC) ;
end component;
component XORGATE is
    Port ( A,B: in STD_LOGIC;
           Y: out STD_LOGIC) ;
end component;
signal difference1,barrow1,barrow2:STD_LOGIC;
begin
x1: HALFSUBTRACTOR port map(A,B,difference1,barrow1);
x2: HALFSUBTRACTOR port map(difference1,C,DIFFERENCE,barrow2);
x3: XORGATE port map (barrow1,barrow2,BARROW);
end Structural;

```

#### SIMULATION OUTPUT:



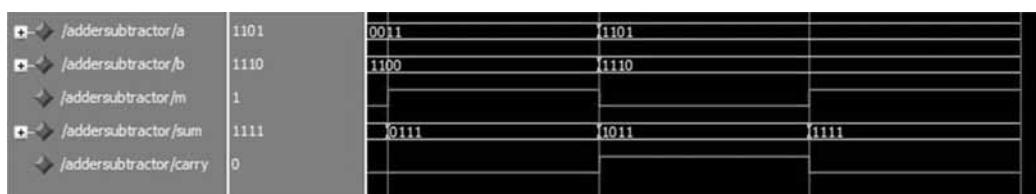
## 5. VHDL PROGRAM FOR 4-BIT BINARY ADDER / SUBTRACTOR

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ADDERSUBTRACTOR is
    Port ( A,B: in STD_LOGIC_VECTOR (3 downto 0);
           M: in STD_LOGIC;
           SUM: out STD_LOGIC_VECTOR (3 downto 0);
           CARRY: out STD_LOGIC);
end ADDERSUBTRACTOR;
architecture Structural of ADDERSUBTRACTOR is
component FULLADDER is
    Port ( A,B,C: in STD_LOGIC;
           SUM,CARRY: out STD_LOGIC);
end component;
component XORGATE is
    Port ( A,B: in STD_LOGIC;
           Y: out STD_LOGIC);
end component;
signal r0,r1,r2,r3,c0,c1,c2:STD_LOGIC;
begin
x1: XORGATE port map(M,B(0),r0);
x2: XORGATE port map(M,B(1),r1);
x3: XORGATE port map(M,B(2),r2);
x4: XORGATE port map(m,B(3),r3);
x5: FULLADDER port map(A(0),r0,M,SUM(0),c0);
x6: FULLADDER port map(A(1),r1,c0,SUM(1),c1);
x7: FULLADDER port map(A(2),r2,c1,SUM(2),c2);
x8: FULLADDER port map(A(3),r3,c2,SUM(3),CARRY);
end Structural;

```

### SIMULATION OUTPUT:



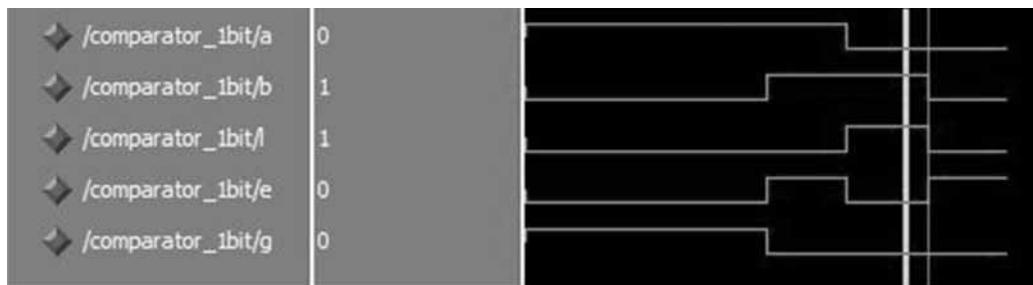
## 6. VHDL PROGRAM FOR 1-BIT COMPARATOR USING DATA FLOW MODELING

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity COMPARATOR_1BIT is
    Port ( A,B : in STD_LOGIC;
           L,E,G : out STD_LOGIC);
end COMPARATOR_1BIT;
architecture Dataflow of COMPARATOR_1BIT is
begin
L<= ((not A) and B);
E<= A xnor B;
G<= (A and (not B));
end Dataflow;

```

### SIMULATION OUTPUT:

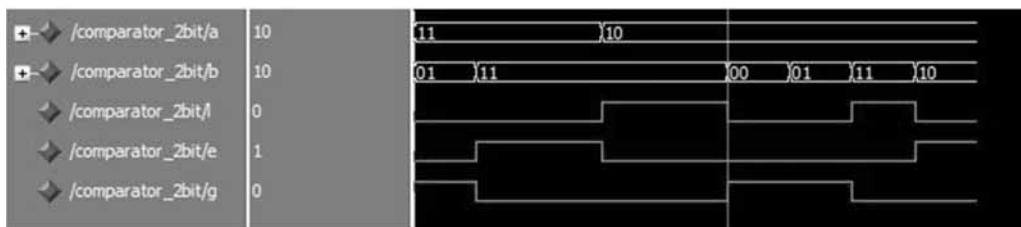


## 7. VHDL PROGRAM FOR 2-BIT COMPARATOR USING DATA FLOW MODELING

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity COMPARATOR_2BIT is
    Port ( A,B : in STD_LOGIC_VECTOR (1 downto 0);
           L,E,G : out STD_LOGIC);
end COMPARATOR_2BIT;
architecture Dataflow of COMPARATOR_2BIT is
begin
L<= ((not A(1)) and B(1)) or ((A(1) xnor B(1)) and ((not A(0)) and B(0)));
E<= (A(1) xnor B(1)) and (A(0) xnor B(0));
G<= (A(1) and (not B(1))) or ((A(1) xnor B(1)) and (A(0) and (not B(0))));
end Dataflow;

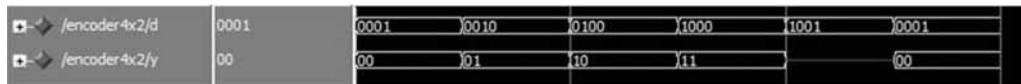
```

**SIMULATION OUTPUT:****8. VHDL PROGRAM FOR 4:2 ENCODER USING BEHAVIORAL MODELING**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ENCODER4X2 is
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
           Y : out STD_LOGIC_VECTOR (1 downto 0));
end ENCODER4X2;
architecture Behavioral of ENCODER4X2 is
begin
process(D)
begin
if(D="1000") then Y<="11";
elsif (D="0100") then Y<="10";
elsif (D="0010") then Y<="01";
elsif (D="0001") then Y<="00";
else Y<="XX";
end if;
end process;
end Behavioral;

```

**SIMULATION OUTPUT:****9. VHDL PROGRAM FOR 8:3 ENCODER USING BEHAVIORAL MODELING**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ENCODER8X3 is
    Port ( D : in STD_LOGIC_VECTOR (7 downto 0);
           Y : out STD_LOGIC_VECTOR (2 downto 0));
end ENCODER8X3;

```

```

end ENCODER8X3;
architecture Behavioral of ENCODER8X3 is
begin
process(D)
begin
if(D="10000000") then Y<="111";
elsif (D="01000000") then Y<="110";
elsif (D="00100000") then Y<="101";
elsif (D="00010000") then Y<="100";
elsif (D="00001000") then Y<="011";
elsif (D="00000100") then Y<="010";
elsif (D="00000010") then Y<="001";
elsif (D="00000001") then Y<="000";
else Y<="XXX";
end if;
end process;
end Behavioral;

```

### SIMULATION OUTPUT:

/encoder8x3/d	10000000	0...00000100	00001000	00001001	00010000	10010000	10000000
/encoder8x3/y	111	001010	011	(100		(111	

### 10. VHDL PROGRAM FOR 2:4 DECODER USING BEHAVIORAL MODELING

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity DECODER2X4 is
    Port ( E : in STD_LOGIC;
           I : in STD_LOGIC_VECTOR (1 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end DECODER2X4;
architecture Behavioral of DECODER2X4 is
begin
process(E,I)
begin
if (E='1') then
    if (I="00") then Y<="0001";
    elsif (I="01") then Y<="0010";
    elsif (I="10") then Y<="0100";
    elsif (I="11") then Y<="1000";
else Y<="0000";
end if;
end process;

```

```

end if;
end if;
end process;
end Behavioral;

```

**SIMULATION OUTPUT:****11. VHDL PROGRAM FOR 3:8 DECODER USING BEHAVIORAL MODELING**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity DECODER3X8 is
    Port ( E : in STD_LOGIC;
           I : in STD_LOGIC_VECTOR (2 downto 0);
           Y : out STD_LOGIC_VECTOR (7 downto 0));
end DECODER3X8;
architecture Behavioral of DECODER3X8 is
begin
process(E,I)
begin
if (E='0') then Y<="00000000";
elsif (E='1') then
    if (I="000") then Y<="00000001";
    elsif (I="001") then Y<="00000010";
    elsif (I="010") then Y<="00000100";
    elsif (I="011") then Y<="00001000";
    elsif (I="100") then Y<="00010000";
    elsif (I="101") then Y<="00100000";
    elsif (I="110") then Y<="01000000";
    elsif (I="111") then Y<="10000000";
end if;
end if;
end process;
end Behavioral;

```

**SIMULATION OUTPUT:**

/decoder3x8/I	111	011	100	101	110	111
/decoder3x8/y	10000000	00001000	00010000	00100000	01000000	10000000

**12. VHDL PROGRAM IN STRUCTURAL MODELING FOR 4:16 DECODER USING 3:8 DECODERS**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity DECODER4X16 is
    Port ( I : in STD_LOGIC_VECTOR (3 downto 0);
           Y : out STD_LOGIC_VECTOR (15 downto 0));
end DECODER4X16;
architecture Structural of DECODER4X16 is
component DECODER3X8 is
    Port ( E : in STD_LOGIC;
           I : in STD_LOGIC_VECTOR (2 downto 0);
           Y : out STD_LOGIC_VECTOR (7 downto 0));
end component;
component NOTGATE is
    Port ( A : in STD_LOGIC;
           Y : out STD_LOGIC);
end component;
signal k:STD_LOGIC;
begin
x1: NOTGATE port map(I(3),k);
x2: DECODER3X8 port map(k,I(2 downto 0),Y(7 downto 0));
x3: DECODER3X8 port map(I(3),I(2 downto 0),Y(15 downto 8));
end Structural;

```

**SIMULATION OUTPUT:**

/decoder4x16/I	0111	0100	0101	0110	0111	1000	1001	1010
/decoder4x16/y	0000000010000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000	0000000000000000

**13. VHDL PROGRAM FOR 8-BIT PRIORITY ENCODER USING BEHAVIORAL MODELING**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

```

## 436 FUNDAMENTALS OF DIGITAL CIRCUITS

```
entity PRIORITYENCODER8X3 is
    Port ( D : in STD_LOGIC_VECTOR (7 downto 0);
           Y : out STD_LOGIC_VECTOR (2 downto 0));
end PRIORITYENCODER8X3;
architecture Behavioral of PRIORITYENCODER8X3 is
begin
process(D)
begin
if (D(7)='1') then Y<="111";
elsif (D(6)='1') then Y<="110";
elsif (D(5)='1') then Y<="101";
elsif (D(4)='1') then Y<="100";
elsif (D(3)='1') then Y<="011";
elsif (D(2)='1') then Y<="010";
elsif (D(1)='1') then Y<="001";
elsif (D(0)='1') then Y<="000";
else Y<="XXX";
end if;
end process;
end Behavioral;
```

### SIMULATION OUTPUT:

/priorityencoder8x3/d	00011000	00000010	00000011	00001000	00001100	00001101	00000000	11000000	11100000	10100000
/priorityencoder8x3/y	100	000	1001	1011	1111	111	11100000	11100000	1110	110

## 14. VHDL PROGRAM FOR 2:1 MULTIPLEXER USING BEHAVIORAL MODELING

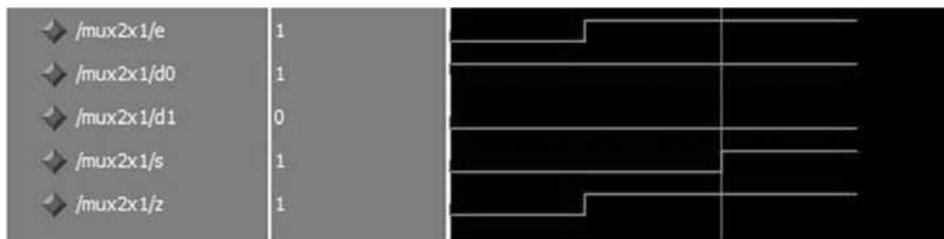
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity MUX2X1 is
    Port ( E,D0,D1 : in STD_LOGIC;
           S : in STD_LOGIC;
           Z : out STD_LOGIC);
end MUX2X1;
architecture Behavioral of MUX2X1 is
begin
process(E,D0,D1)
begin
if (E='0') then Z<='0';
elsif (S='0') then Z<=D0;
elsif (S='1') then Z<=D1;
```

```

end if;
end process;
end Behavioral;

```

#### SIMULATION OUTPUT:



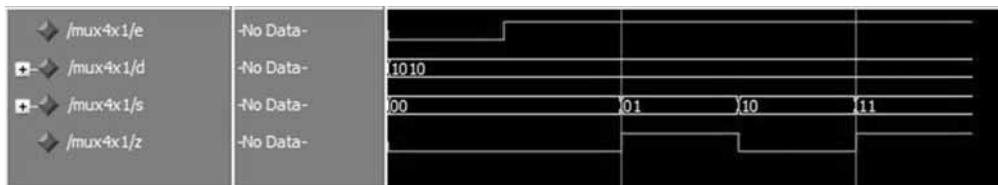
#### 15. VHDL PROGRAM FOR 4:1 MULTIPLEXER USING BEHAVIORAL MODELING

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity MUX4X1 is
    Port ( E : in STD_LOGIC;
           D : in STD_LOGIC_VECTOR (3 downto 0);
           S : in STD_LOGIC_VECTOR (1 downto 0);
           Z : out STD_LOGIC);
end MUX4X1;
architecture Behavioral of MUX4X1 is
begin
process(E,D,S)
begin
if (E='0') then Z<='0';
elsif (S="00") then Z<=D(0);
elsif (S="01") then Z<=D(1);
elsif (S="10") then Z<=D(2);
elsif (S="11") then Z<=D(3);
end if ;
end process;
end Behavioral;

```

#### SIMULATION OUTPUT:



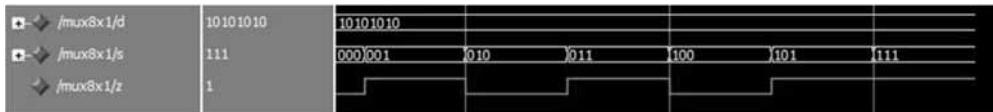
## 16. VHDL PROGRAM IN STRUCTURAL MODELING FOR 8:1 MULTIPLEXER USING TWO 4:1 MULTIPLEXERS

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity MUX8X1 is
    Port ( D : in STD_LOGIC_VECTOR (7 downto 0);
           S : in STD_LOGIC_VECTOR (2 downto 0);
           Z : out STD_LOGIC);
end MUX8X1;
architecture Structural of MUX8X1 is
component MUX4X1 is
    Port ( E : in STD_LOGIC;
           D : in STD_LOGIC_VECTOR (3 downto 0);
           S : in STD_LOGIC_VECTOR (1 downto 0);
           Z : out STD_LOGIC);
end component;
component ORGATE is
    Port ( A,B : in STD_LOGIC;
           Y : out STD_LOGIC);
end component;
component NOTGATE is
    Port ( A : in STD_LOGIC;
           Y : out STD_LOGIC);
end component;
signal S2bar,t,u:std_logic;
begin
x1: NOTGATE port map(S(2), S2bar);
x2: MUX4X1 port map(S2bar,D(3 downto 0),S(1 downto 0),t);
x3: MUX4X1 port map(S(2),D(7 downto 4),S(1 downto 0),u);
x4: ORGATE port map(t,u,Z);
end Structural;

```

### SIMULATION OUTPUT:



## 17. VHDL PROGRAM FOR 1:4 DEMULTIPLEXER USING BEHAVIORAL MODELING

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

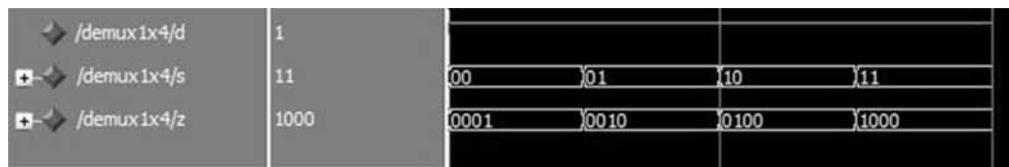
```

```

use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity DEMUX1X4 is
    Port ( D : in STD_LOGIC;
           S : in STD_LOGIC_VECTOR (1 downto 0);
           Z : out STD_LOGIC_VECTOR (3 downto 0));
end DEMUX1X4;
architecture Behavioral of DEMUX1X4 is
begin
process(D,S)
begin
if (S="00") then Z<="000" & D;
elsif (S="01") then Z<="00" & D & '0';
elsif (S="10") then Z<='0' & D & "00";
elsif (S="11") then Z<=D & "000";
end if;
end process;
end Behavioral;

```

#### SIMULATION OUTPUT:



#### 18. VHDL PROGRAM FOR 1:8 DEMULTIPLEXER USING BEHAVIORAL MODELING

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity DEMUX1X8 is
    Port ( D : in STD_LOGIC;
           S : in STD_LOGIC_VECTOR (2 downto 0);
           Z : out STD_LOGIC_VECTOR (7 downto 0));
end DEMUX1X8;
architecture Behavioral of DEMUX1X8 is
begin
process(D,S)
begin
if (S="000") then Z<="0000000" & D;
elsif (S="001") then Z<="0000000" & D & '0';
elsif (S="010") then Z<="0000000" & D & "00";
elsif (S="011") then Z<="0000000" & D & "000";
elsif (S="100") then Z<="0000000" & D & "0000";

```

## 440 FUNDAMENTALS OF DIGITAL CIRCUITS

```
elsif (S="101") then Z<="00" & D & "00000";
elsif (S="110") then Z<='0' & D & "000000";
elsif (S="111") then Z<= D & "0000000";
end if;
end process;
end Behavioral;
```

### SIMULATION OUTPUT:

/demux1x8/d	1					
+---/demux1x8/s	111	000	001	100	101	111
+---/demux1x8/z	10000000	00000001	00000010	00010000	00100000	10000000

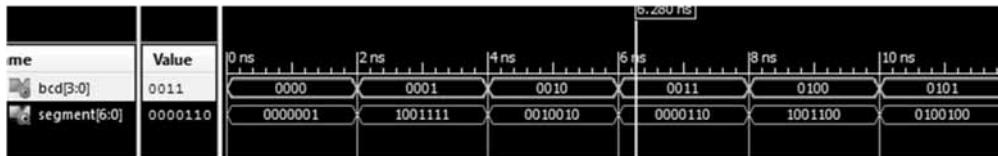
## 19. VHDL PROGRAM FOR BCD-TO-SEVEN SEGMENT DISPLAY USING BEHAVIORAL MODELING

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Seven_Segment_Display is
port (
    clk : in std_logic;
    bcd : in std_logic_vector(3 downto 0);
    segment : out std_logic_vector(6 downto 0) );
end Seven_Segment_Display;
architecture Behavioral of Seven_Segment_Display is
begin
process (clk,bcd)
begin
if (clk'event and clk='1') then
case bcd is
when "0000"=> segment <="0000001"; - '0'
when "0001"=> segment <="1001111"; - '1'
when "0010"=> segment <="0010010"; - '2'
when "0011"=> segment <="0000110"; - '3'
when "0100"=> segment <="1001100"; - '4'
when "0101"=> segment <="0100100"; - '5'
when "0110"=> segment <="0100000"; - '6'
when "0111"=> segment <="0001111"; - '7'
when "1000"=> segment <="0000000"; - '8'
when "1001"=> segment <="0000100"; - '9'
end case;
end if;
end process;
end Behavioral;
```

```

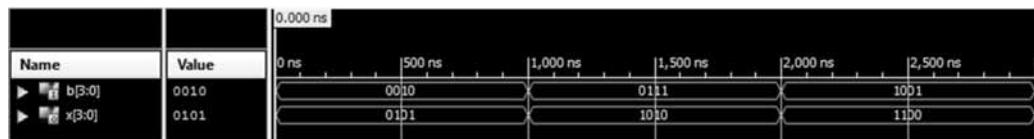
when others=> segment <="1111111";
end case;
end if;
end process;
end Behavioral;
```

**SIMULATION OUTPUT:****20. VHDL PROGRAM FOR BCD-TO-XS-3 CODE CONVERTER**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity BCD_XS3 is
    Port ( B : in STD_LOGIC_VECTOR (3 downto 0);
           X : out STD_LOGIC_VECTOR (3 downto 0));
end BCD_XS3;

architecture Behavioral of BCD_XS3 is
begin
    X(0)<= (NOT B(0));
    X(1)<= ((NOT B(1)) AND (NOT B(0))) OR (B(1) AND B(0));
    X(2)<= (B(2) AND (NOT B(1)) AND (NOT B(0))) OR ((NOT B(2)) AND B(0)) OR ((NOT B(2)) AND B(1));
    X(3)<= (B(3)) OR (B(2) AND B(0)) OR (B(2) AND B(1));
end Behavioral;
```

**SIMULATION OUTPUT:****21. VHDL PROGRAM FOR BINARY-TO-BCD CODE CONVERTER**

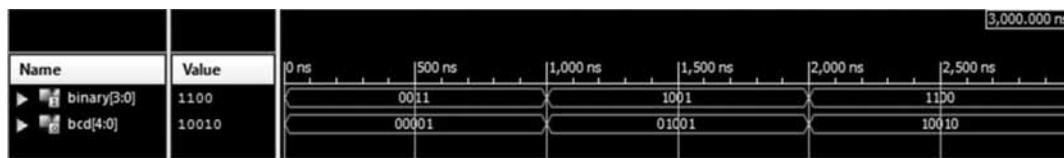
```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity BINRY_BCD is
    Port ( BINARY : in STD_LOGIC_VECTOR(3 DOWNTO 0);
           BCD : out STD_LOGIC_VECTOR(4 DOWNTO 0));
end BINRY_BCD;
```

## 442 FUNDAMENTALS OF DIGITAL CIRCUITS

```
architecture Behavioral of BINRY_BCD is
begin
BCD(0)<=  BINARY(0);
BCD(1)<=  (BINARY(3) AND BINARY(2) AND (NOT BINARY(1))) OR ((NOT
BINARY(3)) AND BINARY(2));
BCD(2)<=  ((NOT BINARY(3)) AND BINARY(2)) OR (BINARY(2) AND
BINARY(1));
BCD(3)<=  (BINARY(3) AND (NOT BINARY(2)) AND (NOT BINARY(1)));
BCD(4)<=  (BINARY(3) AND BINARY(2)) OR (BINARY(3) AND BINARY(1));
end Behavioral;
```

### SIMULATION OUTPUT:

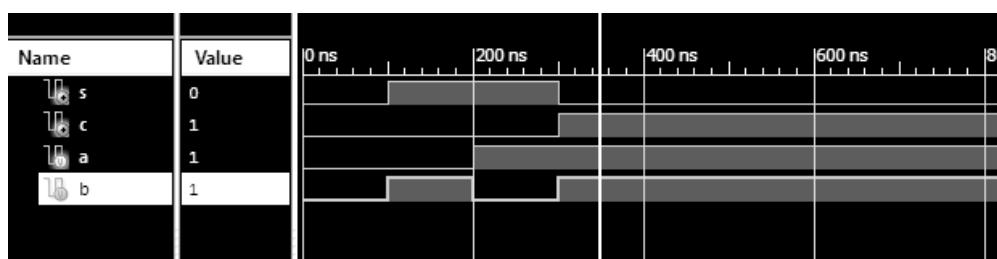


## VERILOG PROGRAMS

### 1. VERILOG PROGRAM FOR HALF-ADDER USING DATA FLOW MODELING

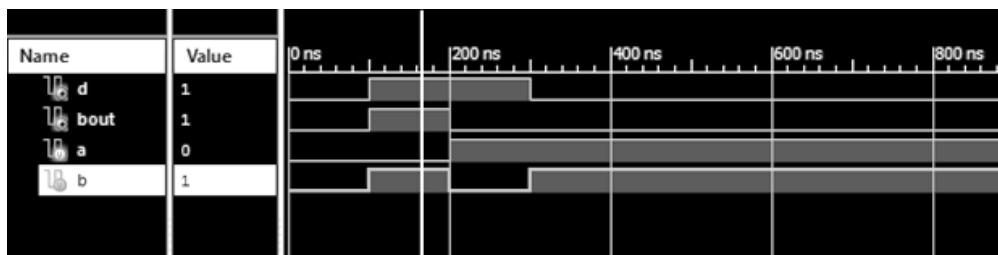
```
module Half_Adder(input a,b,output s,c );
assign s = a ^ b;
assign c = a & b;
endmodule
```

### SIMULATION OUTPUT:

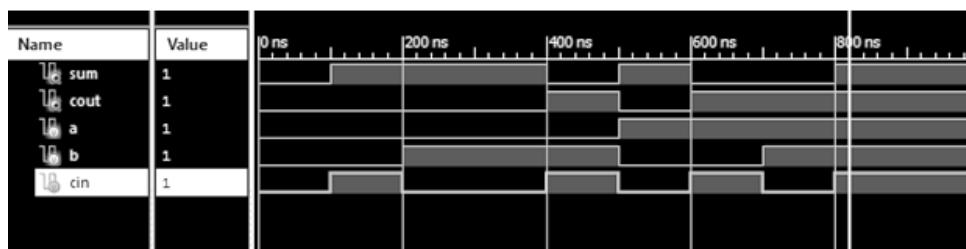


### 2. VERILOG PROGRAM FOR HALF-SUBTRACTOR USING DATA FLOW MODELING

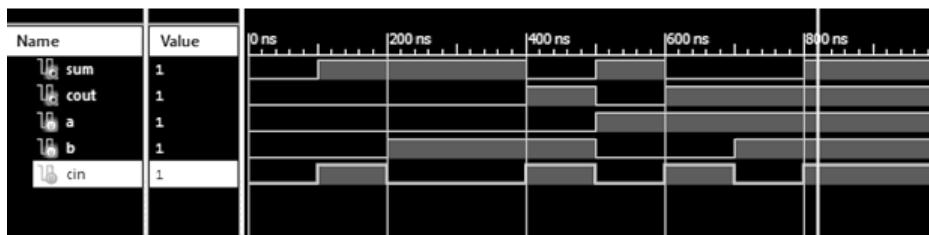
```
module Half_Subtractor(input a,b,output d,bout);
assign d = a ^ b;
assign bout = (~a & b);
endmodule
```

**SIMULATION OUTPUT:****3. VERILOG PROGRAM FOR FULL- ADDER USING GATE LEVEL MODELING**

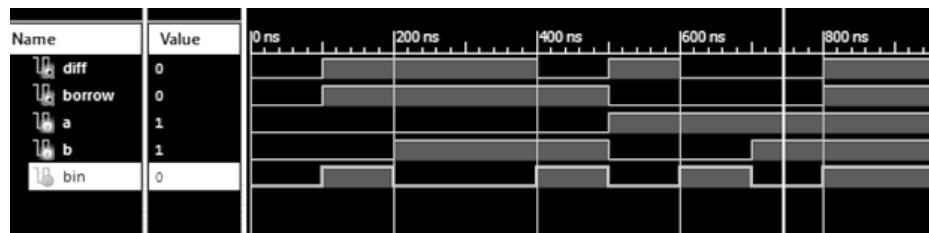
```
module full_adder(sum, cout, a, b, cin);
output sum;
output cout;
input a;
input b;
input cin;
wire s1,t1,t2;
xor x1(s1,a,b);
xor x2(sum,s1,cin);
and a1(t1,a,b);
and a2(t2,s1,cin);
or o1(cout,t1,t2);
endmodule
```

**SIMULATION OUTPUT:****4. VERILOG PROGRAM FOR FULL- ADDER USING DATA FLOW MODELING**

```
module Full_Adder(a,b,cin, sum, cout);
input a,b,cin;
output sum, cout;
assign sum = a ^ b ^ cin;
assign cout = (a & b|b & cin|cin & a);
endmodule
```

**SIMULATION OUTPUT:****5. VERILOG PROGRAM FOR FULL-SUBTRACTOR USING DATA FLOW MODELING**

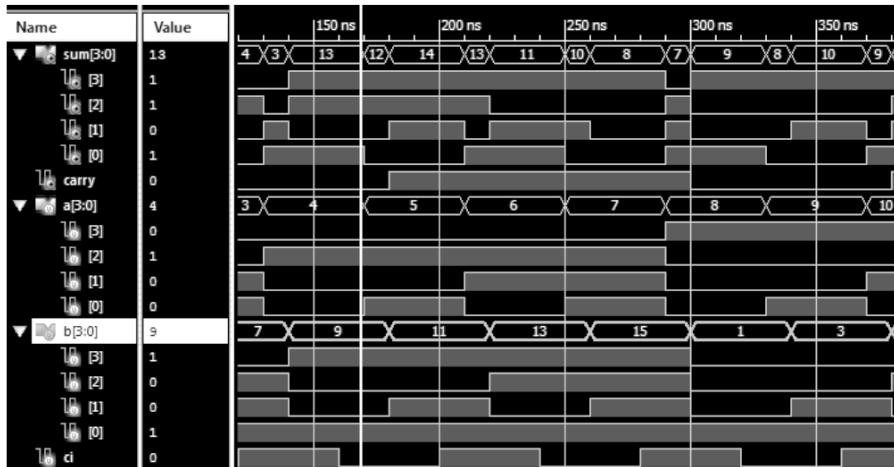
```
module FULL_SUBTRACTOR(
    input a,b,bin,
    output diff,
    output borrow );
assign diff = a^b^bin;
assign borrow = ((~a&b) | (~a&bin) | (b&bin));
endmodule
```

**SIMULATION OUTPUT:****6. VERILOG PROGRAM FOR 4-BIT BINARY ADDER USING STRUCTURAL MODELING**

```
module fourbit_adder(a,b,ci,sum,carry);
    input [3:0] a;
    input [3:0] b;
    input ci;
    output [3:0] sum;
    output carry;
    wire temp1,temp2,temp3;
fulladder fa1(a[0],b[0],ci,sum[0],temp1);
fulladder fa2(a[1],b[1],temp1,sum[1],temp2);
fulladder fa3(a[2],b[2],temp2,sum[2],temp3);
fulladder fa4(a[3],b[3],temp3,sum[3],carry);
endmodule
```

```
module fulladder( input a,b,ci,  output s,c ) ;
assign s = a^b;
assign c = ((a&b) | (b&ci) | (ci&a));
endmodule
```

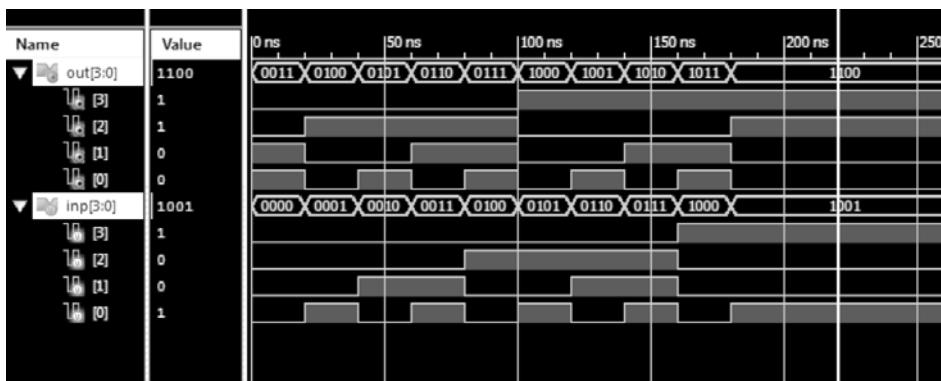
### SIMULATION OUTPUT:



### 7. VERILOG PROGRAM FOR BCD-TO-XS-3 CODE CONVERTER USING DATA FLOW MODELING

```
module bcdtoexcess3(out,inp);
input [3:0] inp;
output [3:0] out;
assign out=inp+4'b0011;
endmodule
```

### SIMULATION OUTPUT:



## **8. VERILOG PROGRAM TO FIND MIDDLE VALUE USING BEHAVIORAL MODELING**

```

module MIDDLE _ VALUE (A, B, C, MIDDLE);
input [3:0] A, B, C;
output MIDDLE;
reg [3:0] MIDDLE; // MIDDLE is the middle number of the three: A,
B, C
always @(A, B, C)
begin : Range_finder_block
if (A > B) // if A is bigger than B
begin
if (B > C) // if further B is bigger than C
begin
MIDDLE <= B;
end
else // B is the smallest
begin
if (A > C) // if A is bigger than C, then C is the middle number
begin
MIDDLE <= C;
end
else // else A is the middle number
begin
MIDDLE <= A;
end
end
end
else // if B is bigger than A
begin
if (A > C) // if B > A > C
begin
MIDDLE <= A;
end
else // A is the smallest
begin
if (B > C) // B > C > A
begin
MIDDLE <= C;
end
else // C > B > A
begin
MIDDLE <= B;
end
end

```

```
end
end
endmodule
```

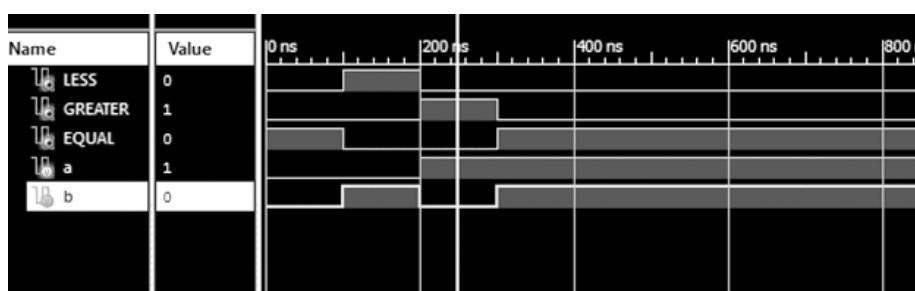
### SIMULATION OUTPUT:



### 9. VERILOG PROGRAM FOR 1-BIT COMPARATOR USING DATA FLOW MODELING

```
module One_Bit_Comparator(a,b,LESS,GREATER,EQUAL);
input a,b;
output LESS,GREATER,EQUAL;
assign LESS = a<b;
assign GREATER= a>b;
assign EQUAL = (a==b);
endmodule
```

### SIMULATION OUTPUT:



## 10. VERILOG PROGRAM FOR 4-BIT COMPARATOR USING DATA FLOW MODELING

```
module Four_Bit_Comparator(a,b,LESS,GREATER,EQUAL);
input [3:0]a,b;
output LESS,GREATER,EQUAL;
assign LESS = a<b;
assign GREATER= a>b;
assign EQUAL = (a==b);
endmodule
```

### SIMULATION OUTPUT:

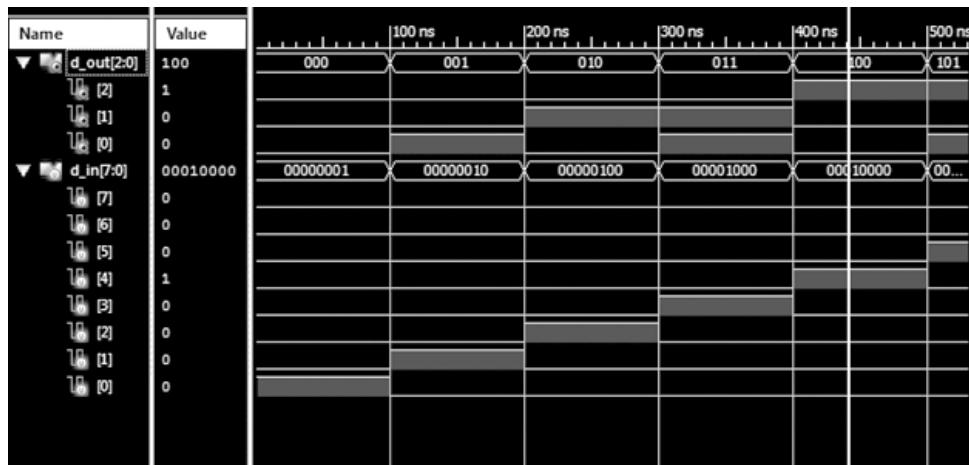
Name	Value	100 ns	200 ns	300 ns	400 ns
LESS	0				
GREATER	1				
EQUAL	0				
a[3:0]	1010	0010	1010		1110
[3]	1				
[2]	0				
[1]	1				
[0]	0				
b[3:0]	1000	0011	1000		1110
[3]	1				
[2]	0				
[1]	0				
[0]	0				

## 11. VERILOG PROGRAM FOR 8:3 ENCODER USING BEHAVIORAL MODELING

```
module encoder_8_3(d_in, d_out);
input [7:0] d_in;
output [2:0] d_out;
reg [2:0] d_out;
always @(d_in)
case (d_in)
8'b00000001 : d_out = 3'b000;
8'b00000010 : d_out = 3'b001;
8'b00000100 : d_out = 3'b010;
8'b00001000 : d_out = 3'b011;
8'b00010000 : d_out = 3'b100;
8'b00100000 : d_out = 3'b101;
8'b01000000 : d_out = 3'b110;
8'b10000000 : d_out = 3'b111;
```

```
default : d_out = 3'b000;
endcase
endmodule
```

### SIMULATION OUTPUT:

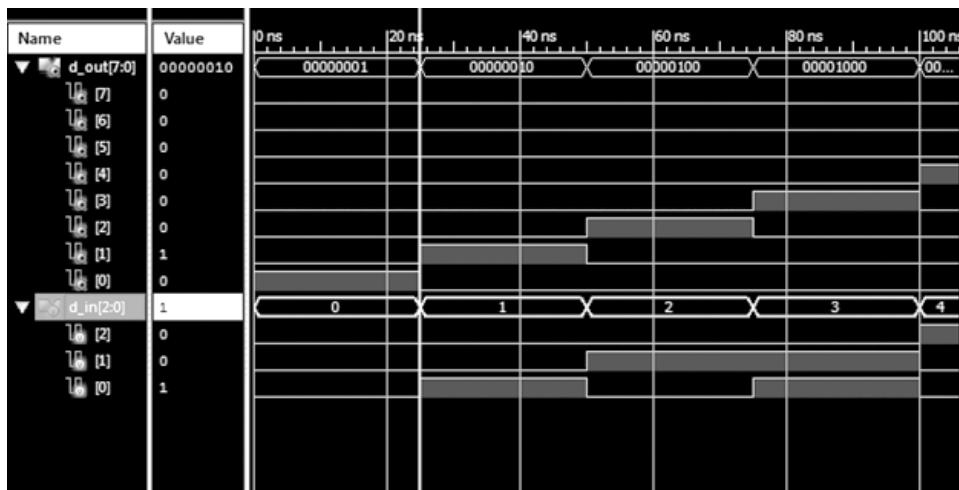


### 12. VERILOG PROGRAM FOR 8-BIT PRIORITY ENCODER USING BEHAVIORAL MODELING

```
module priority_encoder (sel, code);
input [7:0] sel;
output [2:0] code;
reg [2:0] code;
always @(sel)
begin
if (sel[0]) code = 3'b000;
else if (sel[1]) code = 3'b001;
else if (sel[2]) code = 3'b010;
else if (sel[3]) code = 3'b011;
else if (sel[4]) code = 3'b100;
else if (sel[5]) code = 3'b101;
else if (sel[6]) code = 3'b110;
else if (sel[7]) code = 3'b111;
else code = 3'bxxx;
end
endmodule
```

**SIMULATION OUTPUT:****13. VERILOG PROGRAM FOR 3:8 DECODER USING BEHAVIORAL MODELING**

```
module decoder_3_8(d_in, d_out);
input [2:0] d_in;
output [7:0] d_out;
reg [7:0] d_out;
always @(d_in)
case (d_in)
3'b000 : d_out = 8'b00000001;
3'b001 : d_out = 8'b00000010;
3'b010 : d_out = 8'b00000100;
3'b011 : d_out = 8'b00001000;
3'b100 : d_out = 8'b00010000;
3'b101 : d_out = 8'b00100000;
3'b110 : d_out = 8'b01000000;
3'b111 : d_out= 8'b10000000;
default : d_out = 8'b00000000;
endcase
endmodule
```

**SIMULATION OUTPUT:****14. VERILOG PROGRAM IN STRUCTURAL MODELING FOR 4:16 DECODER USING 2:4 DECODERS**

```

module decoder_2x4 (w,y,en);
input [1:0] w;
input en;
output [0:3] y;
reg [0:3] y;
always @(w or en)
begin
case({en,w})
3'b100 : y = 4'b1000;
3'b101 : y = 4'b0100;
3'b110 : y = 4'b0010;
3'b111 : y = 4'b0001;
default : y = 4'b0000;
endcase
end
endmodule
module decoder_4x16 (w,y,en);
input [3:0] w;
input en;
output [0:15] y;
wire [0:3] m;
decoder_2x4 d2 (w[1:0],y[0:3],m[0]);
decoder_2x4 d3 (w[1:0],y[4:7],m[1]);
decoder_2x4 d4 (w[1:0],y[8:11],m[2]);

```

## 452 FUNDAMENTALS OF DIGITAL CIRCUITS

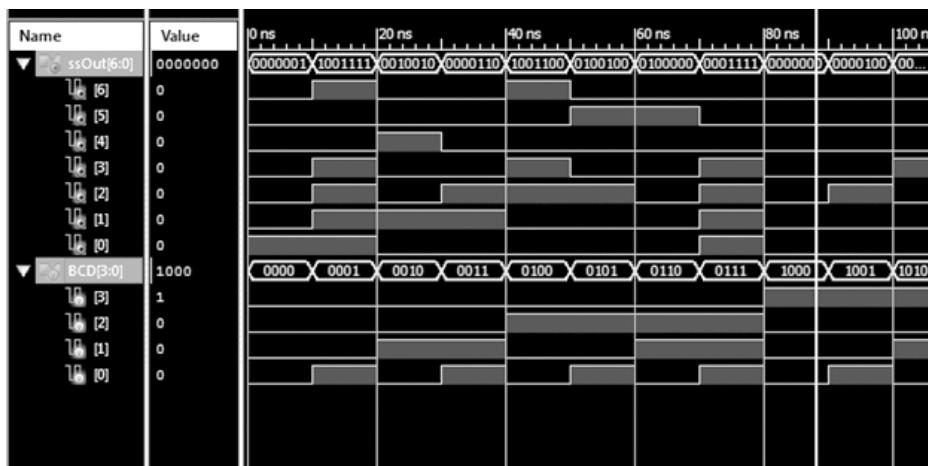
```
decoder_2x4 d5 (w[1:0],y[12:15],m[3]);
decoder_2x4 d6 (w[3:2],m[0:3],en);
endmodule
```

### SIMULATION OUTPUT:

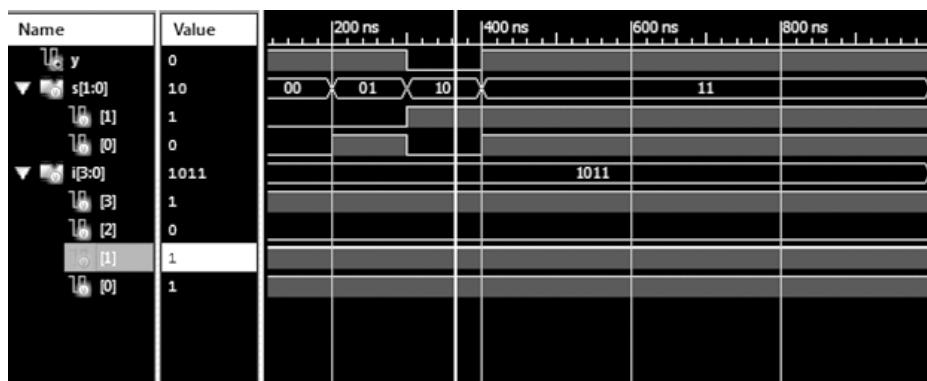


### 15. VERILOG PROGRAM FOR BCD-TO-SEVEN SEGMENT DISPLAY USING BEHAVIORAL MODELING

```
module SevenSegmentDisplayDecoder(ssOut, BCD);
output reg [6:0] ssOut;
input [3:0] BCD;
always @(BCD)
case (BCD)
4'h0: ssOut = 7'b0000001;
4'h1: ssOut = 7'b1001111;
4'h2: ssOut = 7'b0010010;
4'h3: ssOut = 7'b0000110;
4'h4: ssOut = 7'b1001100;
4'h5: ssOut = 7'b0100100;
4'h6: ssOut = 7'b0100000;
4'h7: ssOut = 7'b0001111;
4'h8: ssOut = 7'b0000000;
4'h9: ssOut = 7'b0000100;
4'hA: ssOut = 7'b0001000;
4'hB: ssOut = 7'b1100000;
4'hC: ssOut = 7'b0110001;
4'hD: ssOut = 7'b1000010;
4'hE: ssOut = 7'b0110000;
4'hF: ssOut = 7'b0111000;
default: ssOut = 7'b1111111;
endcase
endmodule
```

**SIMULATION OUTPUT:****16. VERILOG PROGRAM FOR 4:1 MULTIPLEXER USING DATA FLOW MODELING**

```
module mux(s, i, y);
input [1 : 0]s;
input [3 : 0]i;
output y;
wire s0bar, s1bar, p, q, r, t;
assign s0bar = ~ s[0];
assign s1bar = ~ s[1];
assign p= s0bar & s1bar &i[0];
assign q= s[0] & s1bar &i[1];
assign r= s0bar & s[1] &i[2];
assign t= s[0] & s[1] &i[3];
assign y = p | q | r | t;
endmodule
```

**SIMULATION OUTPUT:**

## 17. VERILOG PROGRAM FOR 4:1 MULTIPLEXER USING BEHAVIORAL MODELING

```
module mux41(
    input i0,i1,i2,i3,sel0,sel1,
    output reg y);

    always @(*)
    begin
        case ({sel0,sel1})
            2'b00 : y = i0;
            2'b01 : y = i1;
            2'b10 : y = i2;
            2'b11 : y = i3;
        endcase
    end

endmodule
```

### SIMULATION OUTPUT:



## 18. VERILOG PROGRAM FOR 8:1 MULTIPLEXER USING BEHAVIORAL MODELING

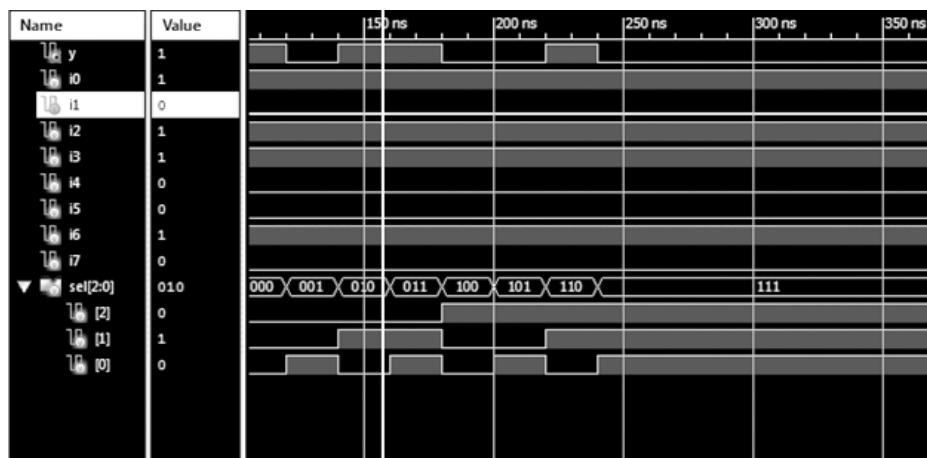
```
module Mux_8_1(i0, i1, i2, i3, i4, i5, i6, i7, sel, y);
    input i0;
    input i1;
    input i2;
    input i3;
    input i4;
    input i5;
    input i6;
```

```

input i7;
input [2:0] sel;
output y;
reg y;
always@(sel,i0,i1,i2,i3,i4,i5,i6,i7)
begin
case (sel)
3'b000 : y <= i0;
3'b001 : y <= i1;
3'b010 : y <= i2;
3'b011 : y <= i3;
3'b100 : y <= i4;
3'b101 : y <= i5;
3'b110 : y <= i6;
3'b111 : y <= i7;
endcase
end
endmodule

```

#### SIMULATION OUTPUT:



#### 19. VERILOG PROGRAM IN STRUCTURAL MODELING FOR 4:1 MULTIPLEXER USING THREE 2:1 MULTIPLEXERS

```

module mux21(i0, i1, s, op);
input i0,i1,s;
output op;
assign op = s ? i1 : i0;
endmodule
module mux41(ip, sel, op);

```

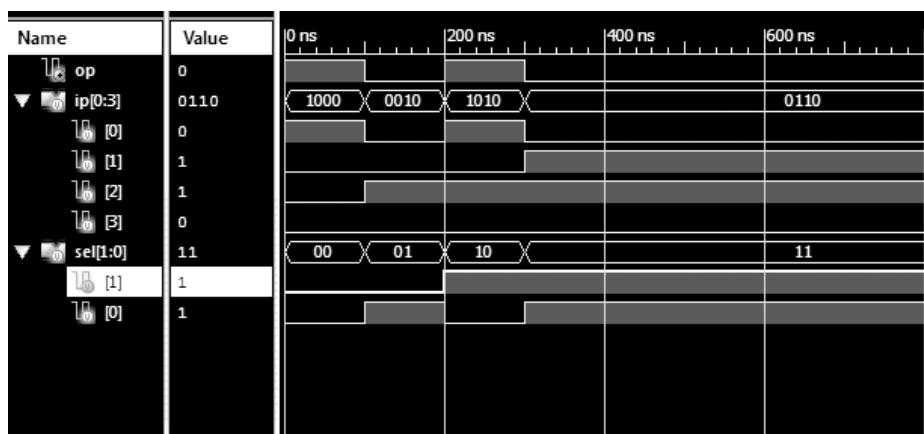
## 456 FUNDAMENTALS OF DIGITAL CIRCUITS

```

input [0:3] ip;
input [1:0] sel;
output op;
wire i1, i2;
mux21 m1(ip[0], ip[1], sel[0], i1);
mux21 m2(ip[2], ip[3], sel[0], i2);
mux21 m3(i1, i2, sel[1], op);
endmodule

```

### SIMULATION OUTPUT:



### 20. VERILOG PROGRAM IN STRUCTURAL MODELING FOR 8:1 MULTIPLEXER USING TWO 4:1 MULTIPLEXERS & ONE 2:1 MULTIPLEXER

```

module mux21(i0, i1, s, op);
input i0,i1,s;
output op;
assign op = s ? i1 : i0;
endmodule
module mux41(ip, sel, op);
input [0:3] ip;
input [1:0] sel;
output op;
wire i1, i2;
mux21 m1(ip[0], ip[1], sel[0], i1);
mux21 m2(ip[2], ip[3], sel[0], i2);
mux21 m3(i1, i2, sel[1], op);
endmodule
module mux81(ip, sel, op);
input [0:7] ip;
input [2:0] sel;

```

```

output op;
wire [0:1] s;
mux41 m11 (ip[0:3], sel[1:0], s[0]);
mux41 m12 (ip[4:7], sel[1:0], s[1]);
mux21 m13 (s[0], s[1], sel[2], op);
endmodule

```

### SIMULATION OUTPUT:



### 21. VERILOG PROGRAM IN STRUCTURAL MODELING FOR 16:1 MULTIPLEXER USING 4:1 MULTIPLEXERS

```

module mux21(i0, i1, s, op);
input i0,i1,s;
output op;
assign op = s ? i1 : i0;
endmodule
module mux41(ip, sel, op);
input [0:3] ip;
input [1:0] sel;
output op;
wire i1, i2;
mux21 m1(ip[0], ip[1], sel[0], i1);
mux21 m2(ip[2], ip[3], sel[0], i2);
mux21 m3(i1, i2, sel[1], op);
endmodule

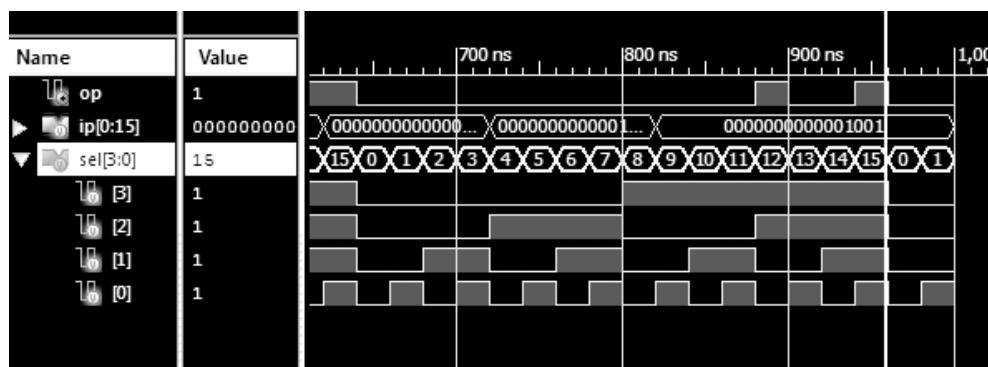
module mux161(ip, sel, op);
input [0:15] ip;
input [3:0] sel;

```

```

output op;
wire [0:3] s;
mux41 m11 (ip[0:3], sel[1:0], s[0]);
mux41 m12 (ip[4:7], sel[1:0], s[1]);
mux41 m13 (ip[8:11], sel[1:0], s[2]);
mux41 m14 (ip[12:15], sel[1:0], s[3]);
mux41 m15 (s, sel[3:2], op);
endmodule

```

**SIMULATION OUTPUT:****22. VERILOG PROGRAM FOR 1:4 DEMULTIPLEXER USING BEHAVIORAL MODELING**

```

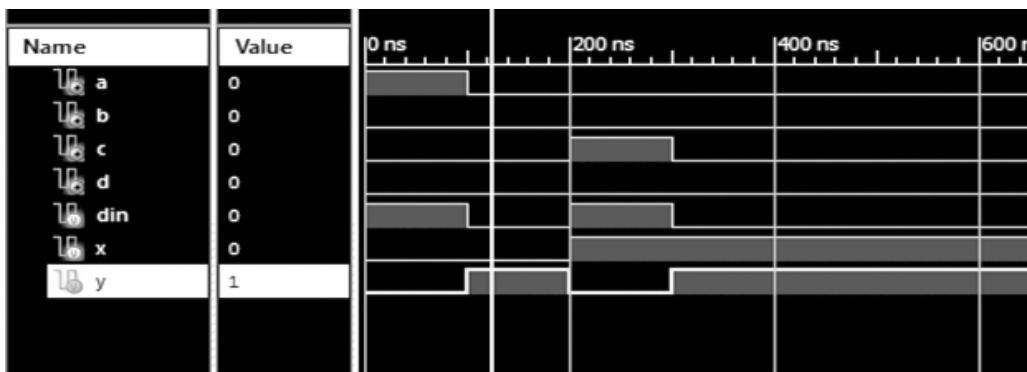
module demultiplexer1_4 ( din ,x ,y ,a ,b ,c ,d );
output a ;
output b ;
output c ;
output d ;

input din ;
input x ;
input y ;

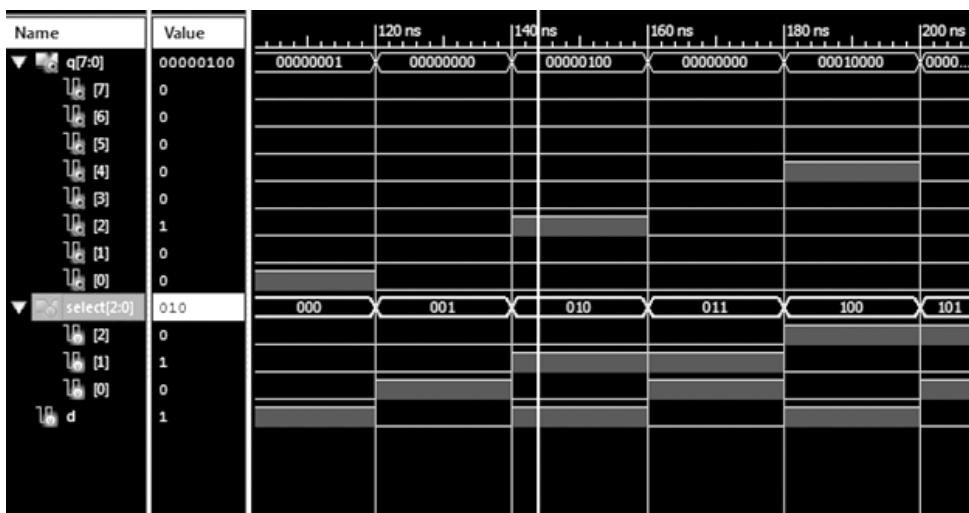
assign a = din & (~x) & (~y) ;
assign b = din & (~x) & y;
assign c = din & x & (~y) ;
assign d = din & x & y;

endmodule

```

**SIMULATION OUTPUT:****23. VERILOG PROGRAM FOR 1:8 DEMULTIPLEXER USING BEHAVIORAL MODELING**

```
module demux8_Bit(select,data,q);
input data;
input [2:0] select;
output [7:0] q;
assign q=data<<select;
endmodule
```



# 8

## PROGRAMMABLE LOGIC DEVICES

### 8.1 INTRODUCTION

Logic designers have a wide range of standard ICs available to them with numerous logic functions and logic circuit arrangements on a chip. In addition, these ICs are available from many manufacturers and at a reasonably low cost. For these reasons, designers have been interconnecting standard ICs to form an almost endless variety of different circuits and systems. Standard ICs (for example, Multiplexers, demultiplexers, decoders, encoders, adders, code converters, comparators, parity generators/checkers, ALUs, etc.) are also called fixed function ICs because each one of them performs a fixed digital operation.

However, there are some problems with circuit and system designs that use only standard ICs. Some system designs might require hundreds or thousands of these ICs.

Some of the disadvantages of this method are as follows:

1. Large board space requirements
2. Large power requirements
3. Lack of security
4. Additional cost, space, power requirements, etc. required to modify the design or to introduce more features

Usually the designs are too complex to be implemented using fixed function ICs.

To overcome the disadvantages of fixed-function ICs, *application specific integrated circuits* (ASICs) have been developed. The ASICs are designed by the users to meet the specific requirements of a circuit and are produced by an IC manufacturer as per the specifications supplied by the user.

The advantages of this method are as follows:

1. Reduced space requirement
2. Reduced power requirement
3. Considerable cost reduction if produced in large volumes
4. Large reduction in size through the use of high levels of integration
5. Designs implemented in this form are almost impossible to copy

The disadvantages of this method are the following:

1. Initial development cost may be enormous.
2. Testing methods may have to be developed which may also increase the cost and effort.

Another approach which has the advantages of both the above methods is the use of *programmable logic devices* (PLDs). A programmable logic device is an IC that is user configurable and is capable of implementing logic functions. It is an LSI chip that contains a ‘regular’ structure and allows the designer to customize it for any specific application, i.e. it is programmed by the user to perform a function required for his application.

A PLD contains a large number of gates, flip-flops, and registers that are interconnected on the chip. Many of the connections, however, are fusible links that can be broken. The IC is said to be programmable because the specific function of the IC for a given application is determined by the selective breaking of some of the interconnections while leaving others intact. The ‘fuse blowing’ process can be done either by the manufacturer in accordance with the customer’s instructions, or by the customer himself. This process is called ‘programming’ because it produces the desired circuit pattern interconnecting the gates, flip-flops, registers, and so on. PLDs can be reprogrammed in a few seconds and hence give more flexibility to experiment with designs. The reprogramming feature of PLDs also makes it possible to accept changes/modifications in the previously designed circuits. All these advantages make PLDs very popular in digital design.

The advantages of PLDs over fixed function ICs are as follows:

1. Low development cost
2. Less space requirement
3. Less power requirement
4. High reliability
5. Easy circuit testing
6. Easy design modification
7. High design security
8. Less design time
9. High switching speed

PLDs have many of the advantages of ASICs as given below:

1. Higher densities
2. Lower quantity production costs
3. Design security
4. Reduced power requirement
5. Reduced space requirement

## 8.2 READ-ONLY MEMORY (ROM)

A read-only memory (ROM) is essentially a memory device in which permanent binary information is stored. The binary information must be specified by the designer and is then embedded in the unit to form the required interconnection pattern. Once the pattern is established, it stays within the unit when the power is turned off and on again. As the name suggests it is meant only for reading the information from it.

A ROM which can be programmed is called a PROM. The process of entering information in a ROM is known as programming. ROMs are used to store information which is of fixed type, such as tables for various functions, fixed data and instructions.

ROMs can be used for designing combinational logic circuits. Depending on the type of ROM used, a user can design and modify the circuits easily.

The advantages of using a ROM as a PLD are the following:

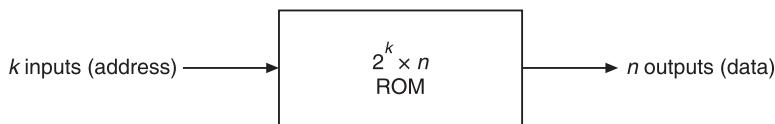
1. Ease of design since no simplification or minimization of logic function is required.
2. Designs can be changed, modified rapidly.
3. It is usually faster than discrete MSI/SSI circuit.
4. Cost is reduced.

There are a few disadvantages also of ROM based circuits, such as:

1. Non-utilization of complete circuit
2. Increased power requirement
3. Enormous increase in size with increase in the number of input variables making it impractical

## 8.3 ROM ORGANIZATION

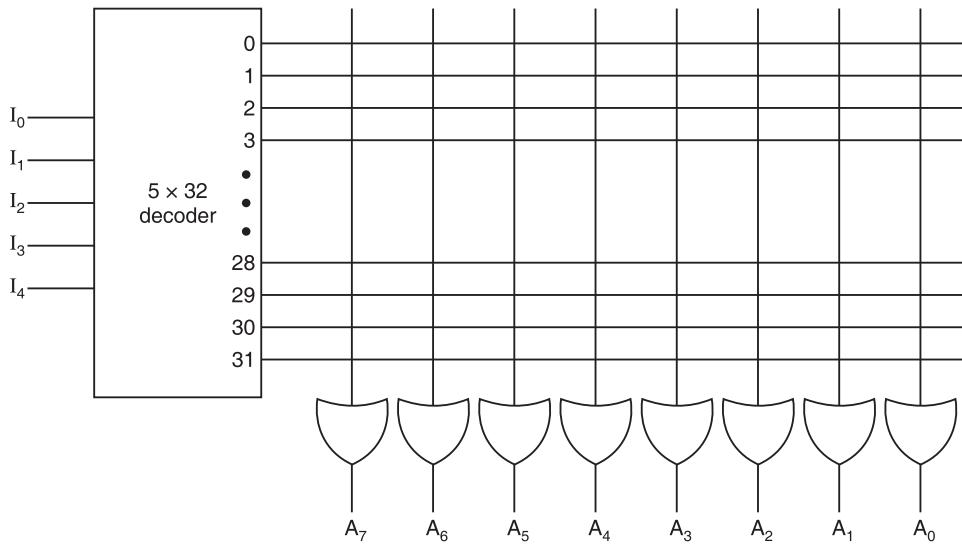
A block diagram of a ROM is shown in Figure 8.1. It consists of  $k$  inputs and  $n$  outputs. The inputs provide the address for the memory and the outputs give the data bits of the stored word which is selected by the address. The number of words in a ROM is determined from the fact that  $k$  address input lines are needed to specify  $2^k$  words. Note that ROM does not have data inputs because it does not have a write operation. Integrated circuit ROM chips have one or more enable inputs and sometimes come with three-state outputs to facilitate the construction of large arrays of ROM.



**Figure 8.1** ROM block diagram.

Consider, for example, a  $32 \times 8$  ROM. The unit consists of 32 words of 8 bits each. There are five input lines that form the binary numbers from 0 through 31 for the address. Figure 8.2 shows the internal logic construction of the ROM. The five inputs are decoded into 32 distinct outputs by means of a  $5 \times 32$  decoder. ROM is basically a decoder with  $k$  inputs and  $2^k$  output lines followed by a bank of OR gates. Each output of the decoder represents a memory address. It represents a

minterm of input variables. The 32 outputs of the decoder are connected to each of the 8 OR gates. The diagram shows the array logic convention used in complex circuits. Each OR gate must be considered as having 32 inputs. Each output of the decoder is connected to one of the inputs of each OR gate. Since each OR gate has 32 input connections and there are 8 OR gates, the ROM contains  $32 \times 8 = 256$  internal connections. In general, a  $2^k \times n$  ROM will have an internal  $k \times 2^k$  decoder and  $n$  OR gates. Each OR gate has  $2^k$  inputs, which are connected to each of the outputs of the decoder.



**Figure 8.2** Internal logic of a 32 × 8 ROM.

The 256 intersections in Figure 8.2 are programmable. A programmable connection between two lines is logically equivalent to a switch that can be altered to either be close (meaning that the two lines are connected) or open (meaning that the two lines are disconnected). The programmable intersection between two lines is sometimes called a *cross point*. Various physical devices are used to implement cross point switches. One of the simplest technologies employs a fuse that normally connects the two points, but is opened or ‘blown’ by applying a high voltage pulse into the fuse.

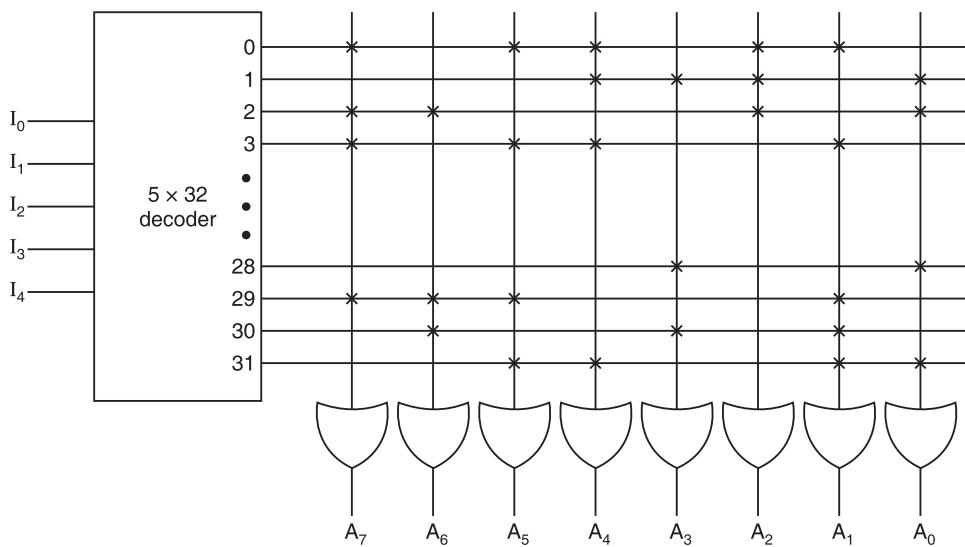
The internal binary storage of a ROM is specified by a truth table that shows the word content in each address. For example, the content of a 32 × 8 ROM may be specified with a truth table similar to the one shown in Table 8.1. The truth table shows the five inputs under which are listed all 32 addresses. At each address, there is stored a word of 8 bits, which is listed under the outputs columns. That table shows only the first four and the last four words in the ROM. The complete table must include the list of all 32 words.

The hardware procedure that programs the ROM results in blowing fuse links according to a given truth table. For example, programming the ROM according to the truth table given by Table 8.1 results in the configuration shown in Figure 8.3. Every 0 listed in the truth table specifies a no connection and every 1 listed specifies a path that is obtained by a connection. For example, the table specifies the 8-bit word 10110010 for permanent storage at address 3. The

four 0s in the word are programmed by blowing the fuse links between output 3 of the decoder and the inputs of the OR gates associated with outputs  $A_6$ ,  $A_3$ ,  $A_2$  and  $A_0$ . The four 1s in the word are marked in the diagram with a  $\times$  to denote a connection in place of a dot used for permanent connection in logic diagrams. When the input of the ROM is 00011, all the outputs of the decoder are 0 except for output 3, which is at logic 1. The signal equivalent to logic 1 at decoder output 3 propagates through the connections to the OR gate outputs of  $A_7$ ,  $A_5$ ,  $A_4$  and  $A_1$ . The other four outputs remain at 0. The result is that the stored word 10110010 is applied to the eight data outputs.

**Table 8.1** ROM truth table (Partial)

Inputs					Outputs							
$I_4$	$I_3$	$I_2$	$I_1$	$I_0$	$A_7$	$A_6$	$A_5$	$A_4$	$A_3$	$A_2$	$A_1$	$A_0$
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
.	.	.	.	.	.	.	.	.	.	.	.	.
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

**Figure 8.3** Programming the ROM according to Table 8.1.

## 8.4 COMBINATIONAL CIRCUIT IMPLEMENTATION

It was shown earlier that a decoder generates the  $2^k$  minterms of the  $k$  input variables. By inserting OR gates to sum the minterms of Boolean functions, we were able to generate any desired combinational circuit. The ROM is essentially a device that includes both the decoder and the OR gates within a single device. By choosing connections for those minterms that are included in the function, the ROM outputs can be programmed to represent the Boolean functions of the output variables in a combinational circuit.

The internal operation of a ROM can be interpreted in two ways. The first interpretation is that of a memory unit that contains a fixed pattern of stored words. The second interpretation is of a unit that implements a combinational circuit. From this point of view, each output terminal is considered separately as the output of a Boolean function expressed as a sum of minterms. For example, the ROM of Figure 8.3 may be considered as a combinational circuit with eight outputs, each being a function of the five-input variables. Output  $A_7$  can be expressed in sum of minterms as

$$A_7(I_4, I_3, I_2, I_1, I_0) = \Sigma(0, 2, 3, \dots, 29)$$

The three dots represent minterms 4 through 27 which are not specified in the Figure 8.3. A connection marked with  $\times$  in the figure produces a minterm for the sum. All other cross points are not connected and are not included in the sum.

In practice, when a combinational circuit is designed by means of a ROM, it is not necessary to design the logic or to show the internal gate connections inside the unit. All that the designer has to do is specify the particular ROM by its IC number and provide the ROM truth table. The truth table gives all the information for programming the ROM. No internal logic diagram is needed to accompany the truth table.

**EXAMPLE 8.1** Design a combinational circuit using a ROM. The circuit accepts a 3-bit number and generates an output binary number equal to the square of the input number.

### Solution

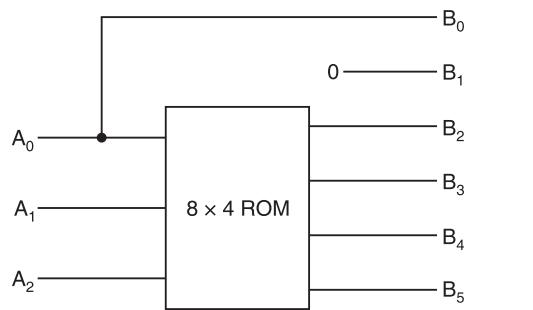
The first step in the design is to derive the truth table of the combinational circuit. In most cases this is all that is needed. In other cases, we can use a partial truth table for the ROM by utilizing certain properties in the output variables. Table 8.2 is the truth table for the combinational circuit. Three inputs and six outputs are needed to accommodate all possible binary numbers. We note that output  $B_0$  is always equal to input  $A_0$ ; so there is no need to generate  $B_0$  with a ROM since it is equal to an input variable. Moreover, output  $B_1$  is always 0, so this output is a known constant. We actually need to generate only four outputs with the ROM; the other two are readily obtained. The minimum size ROM needed must have three inputs and four outputs. Three inputs specify eight words, so the ROM must be of size  $8 \times 4$ . The ROM implementation is shown in Figure 8.4. The three inputs specify eight words of four bits each. The ROM truth table in Figure 8.4a specifies the information needed for programming the ROM. The block diagram of Figure 8.4b shows the required connections of the combinational circuit.

**Table 8.2** Example 8.1: Truth table for circuit

Inputs			Outputs						Decimal
A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	B <sub>5</sub>	B <sub>4</sub>	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49

A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	B <sub>5</sub>	B <sub>4</sub>	B <sub>3</sub>	B <sub>2</sub>
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

(a) ROM truth table



(b) Block diagram

**Figure 8.4** Example 8.1: ROM implementation.

**EXAMPLE 8.2** Give the logic implementation of a  $32 \times 4$  bit ROM using a decoder of a suitable size.

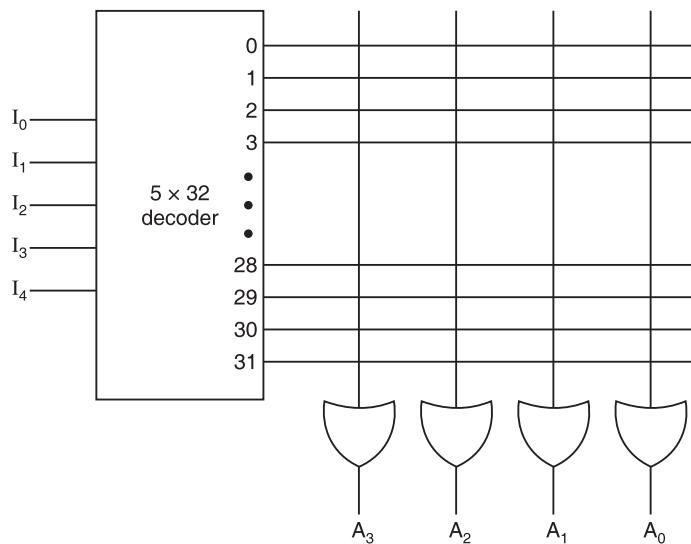
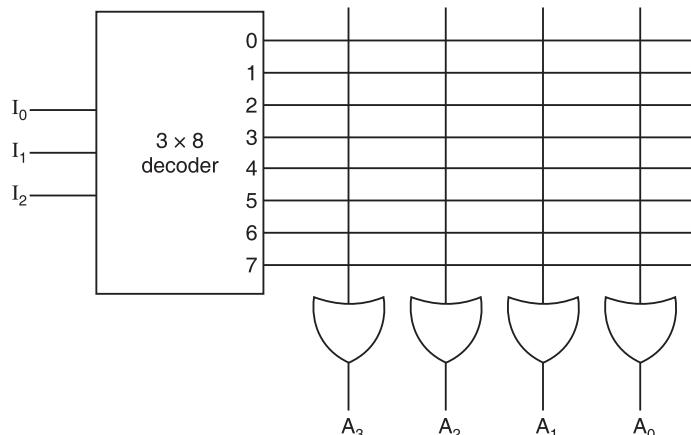
### Solution

A  $32 \times 4$  bit ROM is to be implemented. It consists of 32 words of four bits each. There must be five input lines that form the binary numbers from 0 through 31 for the address. The five inputs are decoded into 32 distinct outputs by means of a  $5 \times 32$  decoder. Each output of the decoder represents a memory address. The 32 outputs of the decoder are connected to each of the four OR gates. Figure 8.5 shows the logic implementation of the  $32 \times 4$  ROM using a  $5 \times 32$  decoder.

**EXAMPLE 8.3** Give the logic implementation of a  $8 \times 4$  bit ROM using a decoder of a suitable size.

### Solution

An  $8 \times 4$  bit ROM is to be implemented. It consists of eight words of four bits each. There must be three input lines that form the binary numbers from 0 through 7 for the address. The three inputs are decoded into 8 distinct outputs by means of a  $3 \times 8$  decoder. Each output of the decoder represents a memory address. The eight outputs of the decoder are connected to each of the four OR gates. Figure 8.6 shows the logic implementation of the  $8 \times 4$  ROM using a  $3 \times 8$  decoder.

Figure 8.5 Implementation of  $32 \times 4$  bit ROM.Figure 8.6 Implementation of  $8 \times 4$  bit ROM.

## 8.5 TYPES OF ROMs

Two types of semiconductor technologies are used for the manufacturing of ROM ICs. These are bipolar technology and MOS technology which use bipolar devices and MOS devices respectively. The process of entering information into a ROM is referred to as *programming* the ROM. Bipolar ROMs and MOS ROMs use different mechanisms of programming. Depending upon the programming process employed, the ROMs are categorized as follows:

### **Mask programmable read-only memory (MROM)**

In this type of read-only memory, the user specifies the data to be stored to the manufacturer of the memory. The data pattern specified by the user are programmed as a part of the fabrication process.

Once programmed, the data pattern can never be changed. This type of read-only memory is referred to as ROM. ROMs are highly suited for very high volume usage due to their low cost.

#### **Programmable read-only memory (PROM)**

This type of memory comes from the manufacturer without any data stored in it, i.e. empty. The data pattern is programmed electrically by the user using a special circuit known as *PROM programmer*. It can be programmed only once during its life time. Once programmed, the data cannot be altered. This type of memory is known as PROM. These are highly suited for high volume usage due to their low cost of production.

#### **Erasable programmable read-only memory (EPROM)**

In this type of memory, data can be written any number of times, i.e. they are reprogrammable. Before it is reprogrammed, the contents already stored are erased by exposing the chip to ultraviolet radiation for about 30 minutes. This type of memory is referred to as EPROM. EPROMs are possible only in MOS technology. Programming is done using a PROM programmer.

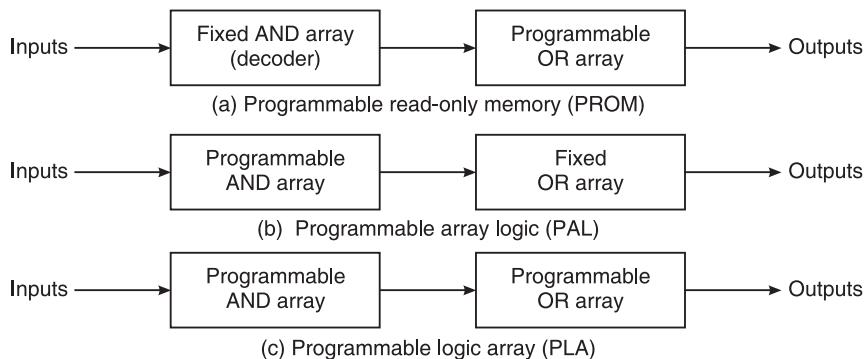
#### **Electrically erasable and programmable read-only memory (EEPROM or E<sup>2</sup>PROM)**

This is another type of reprogrammable memory in which erasing is done electrically rather than exposing the chip to the ultraviolet radiation. It is referred to as EEPROM or electrically alterable ROM (EAROM).

## **8.6 COMBINATIONAL PROGRAMMABLE LOGIC DEVICES**

A combinational PLD is an integrated circuit with programmable gates divided into an AND array and an OR array to provide an AND-OR sum of products implementation. There are three major types of combinational PLDs and they differ in the placement of the programmable connection in the AND-OR array. The various PLDs used are PALs (programmable array logics), PLAs (programmable logic arrays) and PROMs (programmable read only memories).

Figure 8.7 shows the configuration of the 3 PLDs. The programmable read-only memory (PROM) has a fixed AND array constructed as a decoder and a programmable OR array. The AND gates are programmed to provide the product terms for the Boolean functions, which are logically summed in each OR gate. The programmable array logic (PAL) has programmable AND

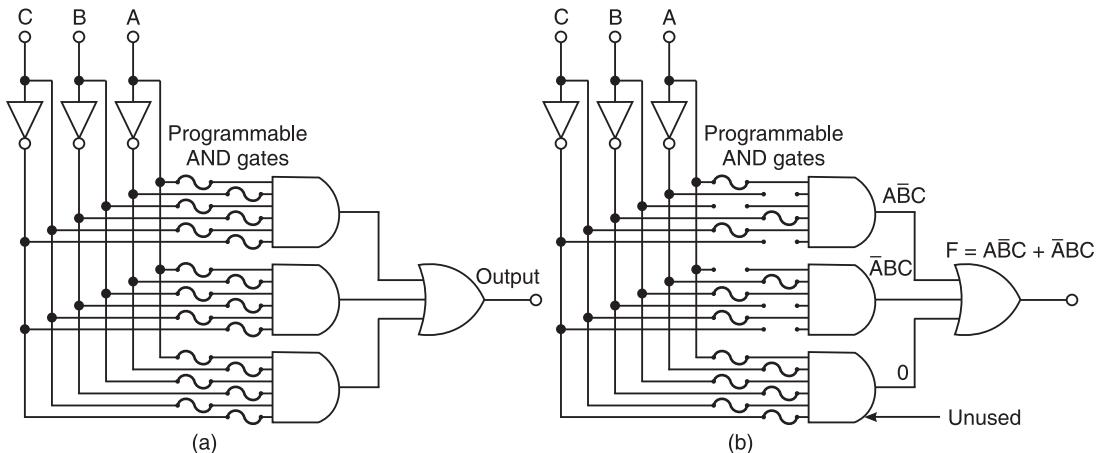


**Figure 8.7** Basic configuration of three PLDs.

array and a fixed OR array. The most flexible PLD is the programmable logic array (PLA) where both the AND and OR arrays can be programmed. The product terms in the AND array may be shared by any OR gate to provide the required sum of products implementation.

## 8.7 PROGRAMMABLE ARRAY LOGIC (PAL)

Programmable array logic (a registered trade mark of Monolithic Memories) is a particular family of programmable logic devices (PLDs) that is widely used and available from a number of manufacturers. The PAL circuits consist of a set of AND gates whose inputs can be programmed and whose outputs are connected to an OR gate, i.e. the inputs to the OR gate are hard-wired, i.e. PAL is a PLD with a fixed OR array and a programmable AND array. Because only the AND gates are programmable, the PAL is easier to program but is not as flexible as the PLA. Some manufacturers also allow output inversion to be programmed. Thus, like AND-OR and AND-OR-INVERT logic, they implement a sum of products logic function. Figure 8.8a shows a small example of the basic structure. The fuse symbols represent fusible links that can be burned open using equipment similar to a PROM programmer. Note that every input variable and its complement can be left either connected or disconnected from every AND gate. We then say that the AND gates are programmed. Figure 8.8b shows how the circuit is programmed to implement  $F = \bar{A}\bar{B}C + A\bar{B}C$ . Note this important point. All input variables and their complements are left connected to the unused AND gate, whose output is, therefore,  $A\bar{A}B\bar{B}C\bar{C} = 0$ . The 0 has no affect on the output of the OR gate. On the other hand, if all inputs to the unused AND gate were burned open, the output of the AND gate would ‘float’ HIGH (logic 1), and the output of the OR gate in that case would remain permanently 1. The actual PAL circuits have several groups of AND gates, each group providing inputs to separate OR gates.



**Figure 8.8** Basic structure of a PAL circuit, and implementation of  $F = \bar{A}\bar{B}C + A\bar{B}C$ .

Figure 8.9a shows a conventional means for abbreviating PAL connection diagrams. Note that the AND gate is drawn with a single input line, whereas in reality, it has three inputs. The vertical lines denote the inputs and the horizontal lines feed the AND gates. An  $\times$  sign denotes a connection through an intact fusible link and a dot sign represents a permanent connection. The

absence of any symbol represents an open or no connection by virtue of a burned-open link. In the example shown, input A is connected to the gate through a fusible link, input C is permanently connected, and input B is disconnected. Therefore, the output of the gate is AC.

Figure 8.9b shows an example of how the PAL structure is represented using the abbreviated connections. It is a 3-input 3-wide AND-OR structure. In this example, each function can have three minterms or product terms. Notice that there are nine AND gates, which implies only nine chosen products of not more than three variables ABC. Inputs to the OR gates at the outputs are fixed as shown by  $\times$ s marked on the vertical lines. The inputs to the AND gates are marked on the corresponding line by the  $\times$ s. Removing the  $\times$  implies blowing off the corresponding fuse which in turn implies that the corresponding input variable is not applied to the particular AND gate. In this example, the circuit is unprogrammed because all the fusible links are intact. Note that, the 3-input OR gates in Figure 8.9c are also drawn with a single input line.

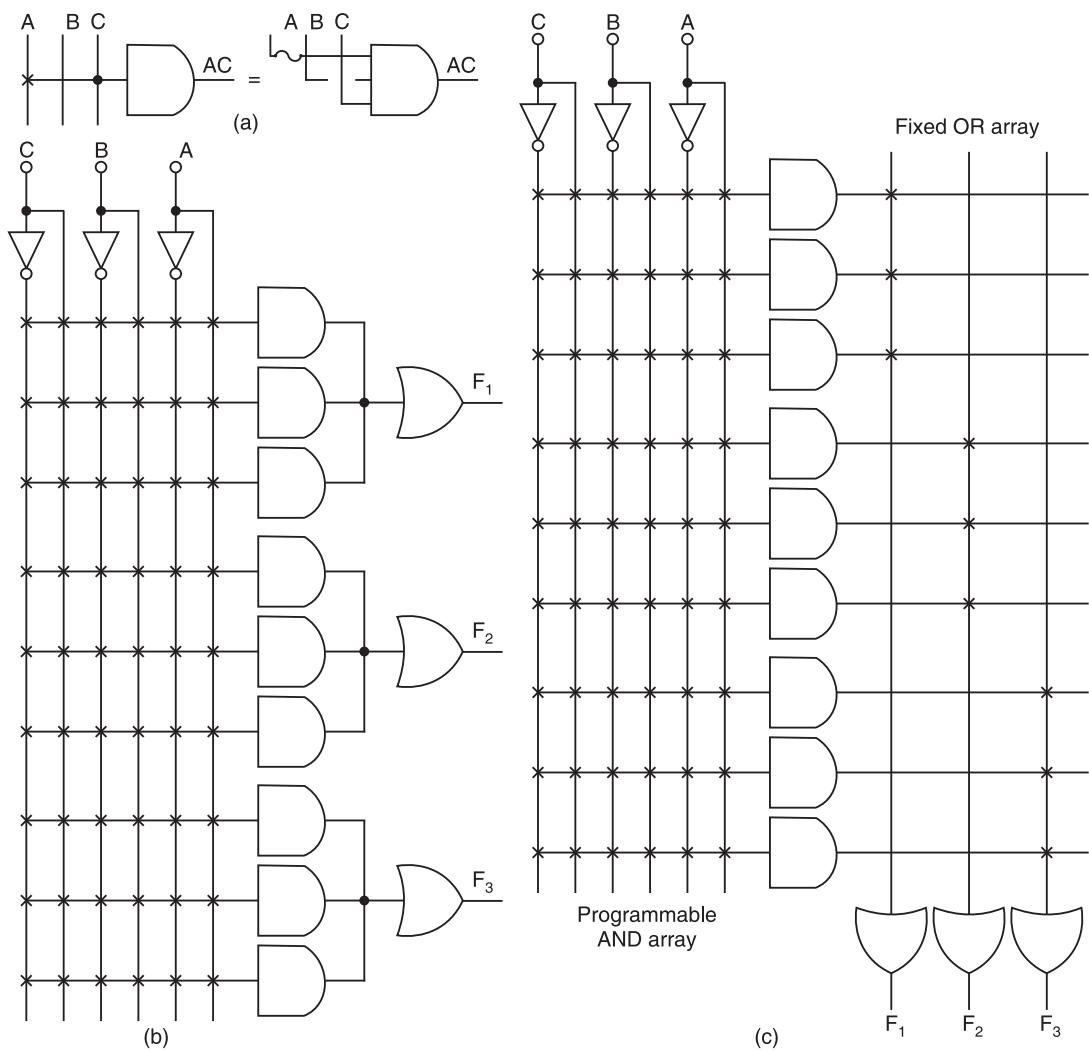
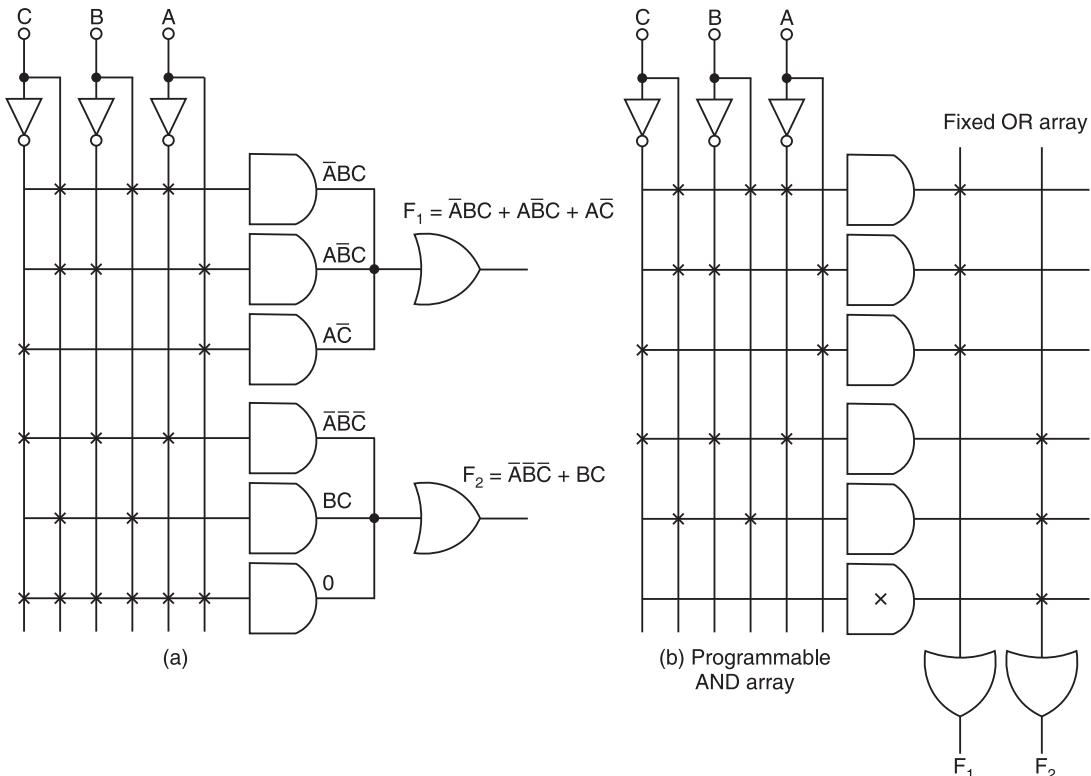


Figure 8.9 Simplified method for showing connections in PAL circuits.

**EXAMPLE 8.4** Using the connection abbreviations, redraw the circuit in Figure 8.9c to show how it can be programmed to implement  $F_1 = \bar{A}\bar{B}C + A\bar{C} + \bar{A}\bar{B}\bar{C}$  and  $F_2 = \bar{A}\bar{B}\bar{C} + BC$ .

**Solution**

The redrawn circuit to implement the given functions is shown in Figure 8.10a. Note that one unused AND gate has all its links intact. All links intact can be represented by a  $\times$  in the AND gate as shown in Figure 8.10b. Such a diagram is sometimes called the *fuse map*.



**Figure 8.10** Example 8.4: PAL programmed to implement  $F_1 = \bar{A}\bar{B}C + A\bar{C} + \bar{A}\bar{B}\bar{C}$  and  $F_2 = \bar{A}\bar{B}\bar{C} + BC$ .

An example of an actual PAL IC is the PAL 18L8A from Texas Instruments. It is manufactured using low power Schottky technology and has ten logic inputs and eight output functions. Each output OR gate is hard-wired to seven AND gate outputs and therefore it can generate functions that include up to seven terms. An added feature of this particular PAL is that six of the eight outputs are fed back into AND array, where they can be connected as inputs to any AND gate. This makes the device very useful in generating all sorts of combinational logic.

### 8.7.1 PAL Programming Table

The fuse map of a PAL can be specified in a tabular form. The PAL programming table consists of three columns. The first column lists the product terms numerically. The second column specifies

the required paths between inputs and AND gates. The third column specifies the outputs of the OR gates. For each product term the inputs are marked with 1, 0, or – (dash). If a variable in the product term appears in its true form, the corresponding input variable is marked with a 1. If it appears in complemented form, the corresponding input variable is marked with a 0. If the variable is absent in the product term, it is marked as a – (dash).

The paths between the inputs and the AND gates are specified under the column heading *inputs* in the programming table. A 1 in the input column specifies a connection from the input variable to the AND gate. A 0 in the input column specifies a connection from the complement of the variable to the input of the AND gate. A dash specifies a blown fuse in both the input variable and its complement. It is assumed that an open terminal in the input of an AND gate behaves like a 1.

The outputs of the OR gates are specified under the column heading *outputs*. The size of a PAL is specified by the number of inputs, the number of product terms, and the number of outputs. For  $n$  inputs,  $k$  product terms, and  $m$  outputs the internal logic of the PAL consists of  $n$  buffer inverter gates,  $k$  AND gates, and  $m$  OR gates.

When designing a digital system with a PAL, there is no need to show the internal connections of the unit. All that is needed is a PAL programming table from which the PAL can be programmed to supply the required logic. When implementing a combinational circuit with a PAL, careful investigation must be undertaken in order to reduce the number of distinct product terms. Since a PAL has a finite number of AND gates, this can be done by simplifying each Boolean function to a minimum number of terms.

**EXAMPLE 8.5** Implement the following Boolean functions using PAL with four inputs and 3-wide AND-OR structure. Also write the PAL programming table.

$$F_1(A, B, C, D) = \Sigma m(2, 12, 13)$$

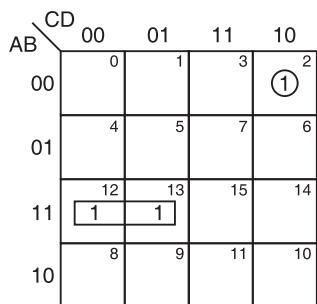
$$F_2(A, B, C, D) = \Sigma m(7, 8, 9, 10, 11, 12, 13, 14, 15)$$

$$F_3(A, B, C, D) = \Sigma m(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$$

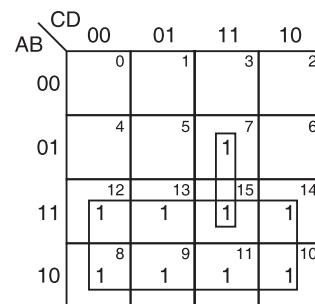
$$F_4(A, B, C, D) = \Sigma m(1, 2, 8, 12, 13).$$

### Solution

The K-maps for the above expressions, their minimization and the minimal expressions obtained from them are shown in Figure 8.11. Note that the function for  $F_4$  has four product terms. The logical sum of two of these terms is equal to  $F_1$ . By using  $F_1$  it is possible to reduce the number of terms for  $F_4$  from four to three. The implementation of the minimal logic expressions using PAL is shown in Figure 8.12b.



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



$$F_2 = A + BCD$$

Figure 8.11 Example 8.5: K-maps (Contd.)

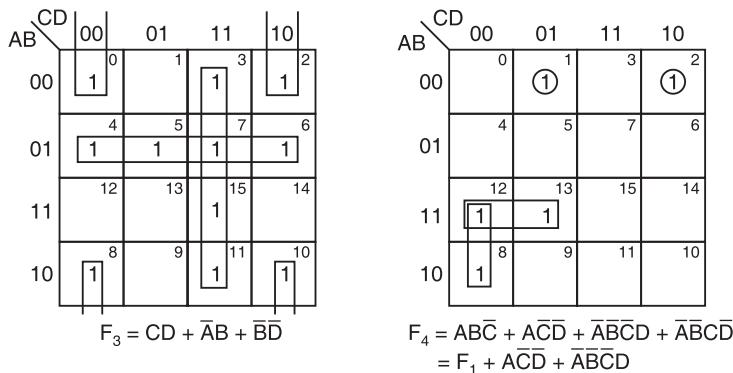


Figure 8.11 Example 8.5: K-maps.

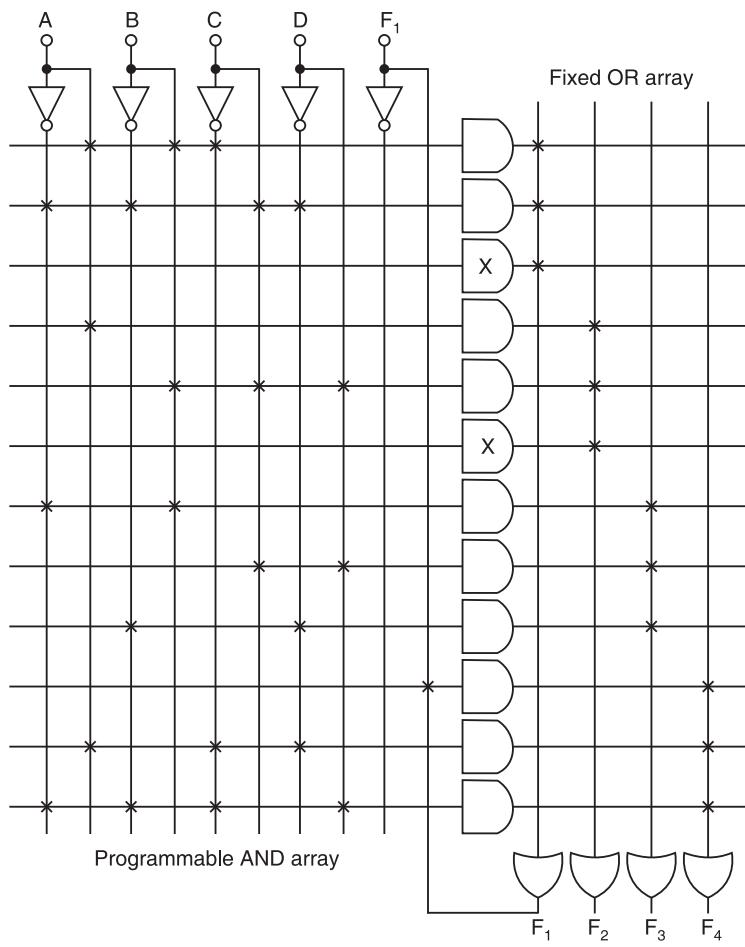
The programming table that specifies the PAL of Figure 8.12b is listed in Figure 8.12a. Since the given problem has four outputs the table is divided into four sections with three product terms in each to conform with the PAL of Figure 8.9b. The first two sections need only two product terms to implement the Boolean function. The last section for output  $F_4$  needs four product terms. Using the output from  $F_1$  we can reduce the function into a function with three terms.

The fuse map for the PAL as specified in the programming table is shown in Figure 8.12b. For each 1 or 0 in the table, we mark the corresponding intersection in the diagram with the symbol for an intact fuse. For each dash, we mark the diagram with blown fuses in both the true and complement inputs. If the AND gate is not used, we leave all its input fuses intact. Since the corresponding input receives both the true and the complement of each input variable, we have  $\bar{A}A = 0$  and the output of the AND gate is always 0. Usually a  $\times$  inside the AND gate is used to indicate that all its input fuses are intact.

Product term	AND Inputs					Outputs
	A	B	C	D	$F_1$	
1	1	1	0	—	—	$F_1 = ABC + \bar{A}\bar{B}\bar{D}$
2	0	0	1	0	—	
3	—	—	—	—	—	
4	1	—	—	—	—	
5	—	1	1	1	—	$F_2 = A + BCD$
6	—	—	—	—	—	
7	0	1	—	—	—	$F_3 = \bar{A}B + CD + \bar{B}\bar{D}$
8	—	—	1	1	—	
9	—	0	—	0	—	
10	—	—	—	—	1	$F_4 = F_1 + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D}$
11	1	—	0	0	—	
12	0	0	0	1	—	

(a) PAL programming table

Figure 8.12 Example 8.5 (Contd.)



(b) Realization of the example functions using PAL

**Figure 8.12** Example 8.5.

**EXAMPLE 8.6** Realize the following functions using a PAL with four inputs and 3-wide AND-OR structure. Also write the PAL programming table.

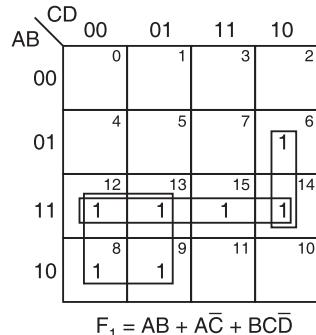
$$\begin{array}{ll} F_1(A, B, C, D) = \Sigma m(6, 8, 9, 12-15) & F_2(A, B, C, D) = \Sigma m(1, 4-7, 10-13) \\ F_3(A, B, C, D) = \Sigma m(4-7, 10-11) & F_4(A, B, C, D) = \Sigma m(4-7, 9-15) \end{array}$$

### Solution

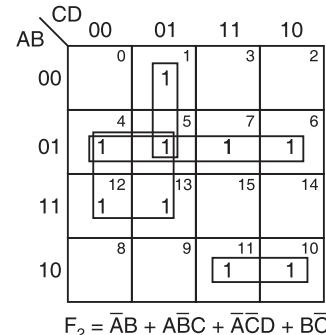
The first step in the realization is to obtain the minimal sum of products form of all the given functions. The K-maps for the given functions, their minimization, and the minimal SOP expressions obtained from them are shown in Figure 8.13.

Each section in the PAL comprises three AND gates feeding the given OR gate. There are four such sections. Notice that  $F_2$  has four product terms but the given PAL device has provision for three products only as inputs to OR gates. So some manipulation becomes necessary. Observe that out of four terms two of the terms of  $F_2$  are equal to  $F_3$  itself. So  $F_2$

can be written as the sum of  $F_3$  and the remaining two terms. The PAL programming table is shown in Figure 8.14a. The actual realization is shown in Figure 8.14b.

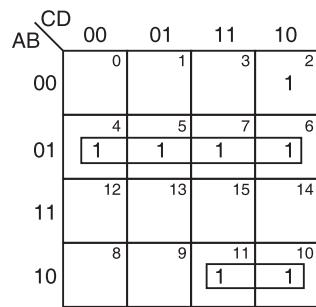


$$F_1 = AB + A\bar{C} + BCD$$

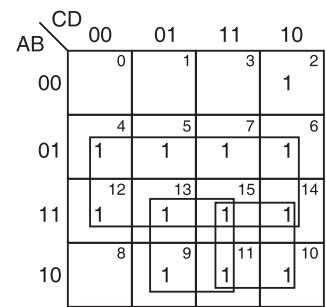


$$F_2 = \bar{A}B + A\bar{B}C + \bar{A}\bar{C}D + B\bar{C}$$

$$= F_3 + B\bar{C} + \bar{A}\bar{C}D$$



$$F_3 = \bar{A}B + A\bar{B}C$$



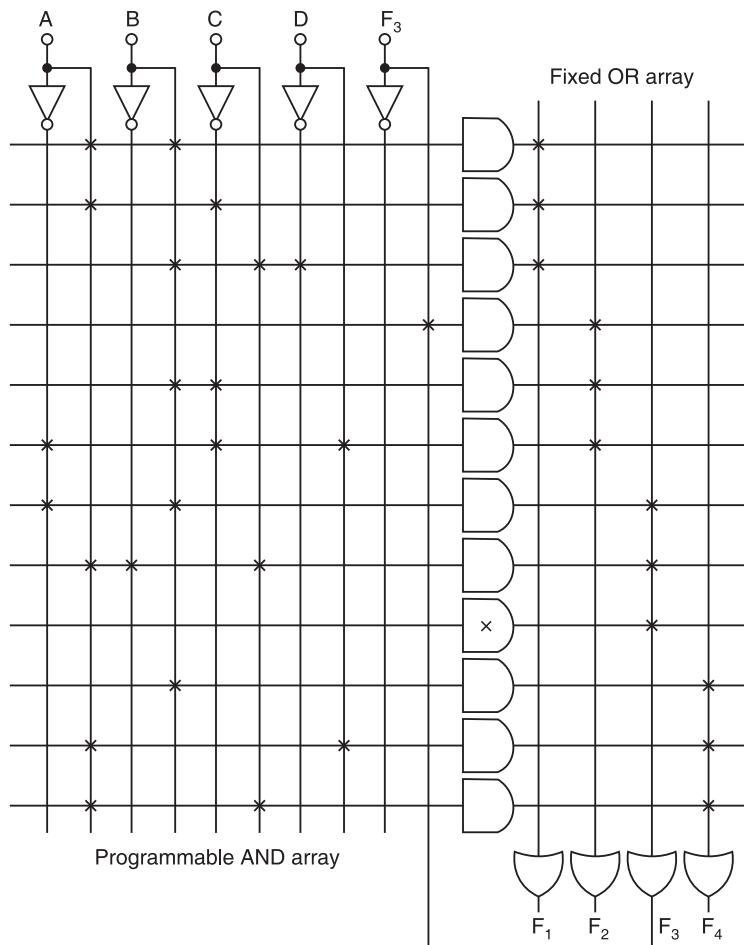
$$F_4 = B + AD + AC$$

**Figure 8.13** Example 8.6: K-maps.

Product term	AND Inputs					Outputs
	A	B	C	D	$F_3$	
1	1	1	—	—	—	
2	1	—	0	—	—	$F_1 = AB + A\bar{C} + BCD$
3	—	1	1	0	—	
4	—	—	—	—	1	$F_2 = \bar{A}B + A\bar{B}C + \bar{A}\bar{C}D + B\bar{C}$
5	0	—	0	1	—	$= F_3 + B\bar{C} + \bar{A}\bar{C}D$
6	—	1	0	—	—	
7	0	1	—	—	—	
8	1	0	1	—	—	$F_3 = \bar{A}B + A\bar{B}C$
9	—	—	—	—	—	
10	—	1	—	—	—	
11	1	—	—	1	—	$F_4 = B + AD + AC$
12	1	—	1	—	—	

(a) PAL programming table

**Figure 8.14** Example 8.6 (Contd.)



(b) Realization of the example functions using PAL

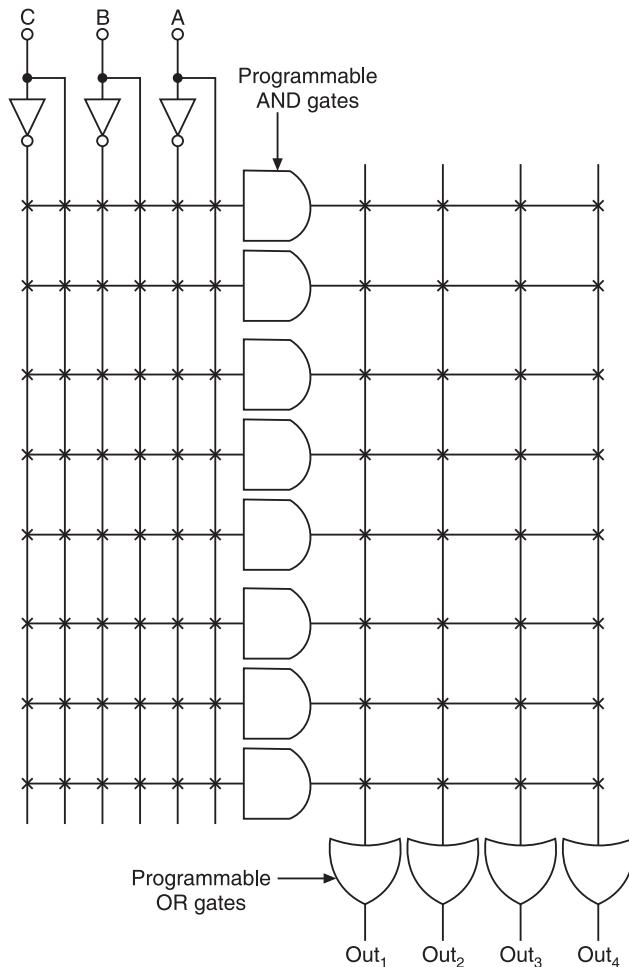
**Figure 8.14** Example 8.6.

## 8.8 PROGRAMMABLE LOGIC ARRAY (PLA)

The PLA represents another type of programmable logic but with a slightly different architecture. The PLA combines the characteristics of the PROM and the PAL by providing both a programmable OR array and a programmable AND array, i.e. in a PLA both AND gates and OR gates have fuses at the inputs. A third set of fuses in the output inverters allows the output function to be inverted if required. Usually X-OR gates are used for controlled inversion. This feature makes it the most versatile of the three PLDs. However, it has some disadvantages. Because it has two sets of fuses, it is more difficult to manufacture, program and test it than a PROM or a PAL. Figure 8.15 demonstrates the structure of a three-input, four-output PLA with every fusible link intact.

Like ROM, PLA can be mask programmable or field programmable. With a mask programmable PLA, the user must submit a PLA programming table to the manufacturer. This

table is used by the vendor to produce a user made PLA that has the required internal paths between inputs and outputs. A second type of PLA available is called a field programmable logic array or FPLA. The FPLA can be programmed by the user by means of certain recommended procedures. CPLAs can be programmed with commercially available programmer units.



**Figure 8.15** Structure of (an unprogrammed) PLA circuit.

**EXAMPLE 8.7** Show how the PLA circuit in Figure 8.15 would be programmed to implement the sum and carry outputs of a full adder.

#### *Solution*

The truth table of a full-adder is shown in Figure 8.16a . Drawing the K-maps for the sum and carry-out terms and minimizing them, the minimal expressions for the sum and carry-out terms are:

The sum is

$$S = \overline{A}\overline{B}C_{in} + \overline{A}B\overline{C}_{in} + A\overline{B}\overline{C}_{in} + ABC_{in}$$

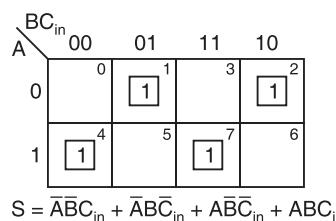
and the carry-out is

$$C_{\text{out}} = AB + AC_{\text{in}} + BC_{\text{in}}$$

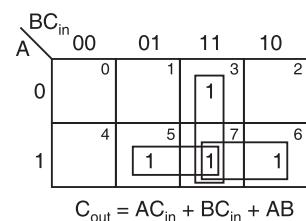
To implement these expressions, we need a 4-input OR gate and a 3-input OR gate. Since the inputs to the OR gates of the PLA can be programmed, we can implement the given expressions as shown in Figure 8.16c.

Inputs			Outputs	
A	B	$C_{\text{in}}$	S	$C_{\text{out}}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) Truth table



$$S = \bar{A}\bar{B}C_{\text{in}} + \bar{A}B\bar{C}_{\text{in}} + A\bar{B}\bar{C}_{\text{in}} + ABC_{\text{in}}$$



(b) K-maps

$$C_{\text{out}} = AC_{\text{in}} + BC_{\text{in}} + AB$$

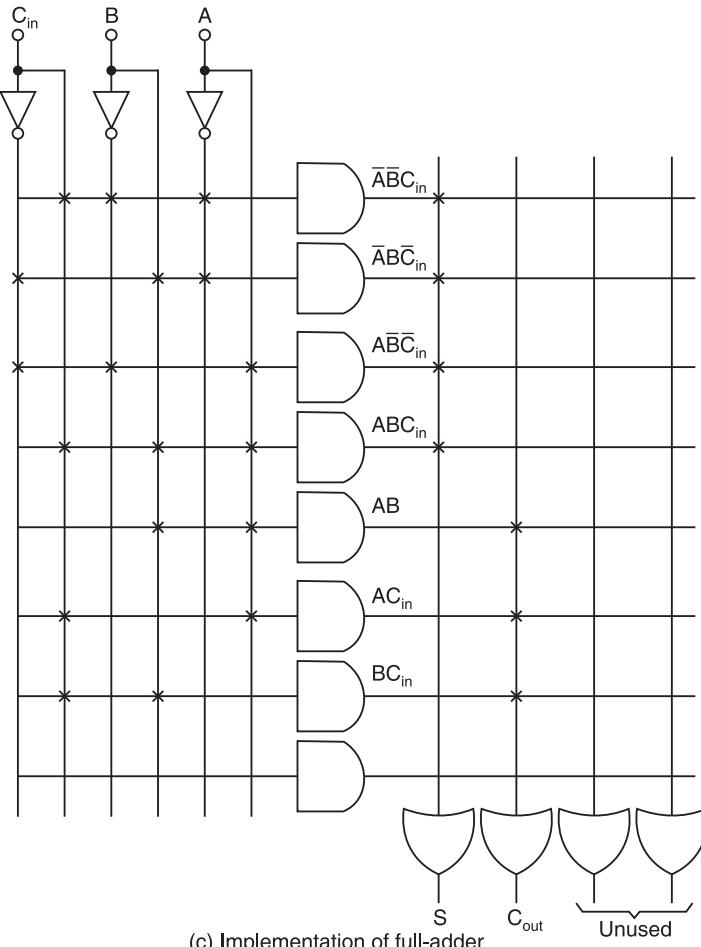


Figure 8.16 Example 8.7.

**EXAMPLE 8.8** Show how the PLA circuit in Figure 8.15 can be programmed to implement the 3-bit binary-to-gray conversion.

**Solution**

The conversion table of 3-bit binary ( $B_3, B_2, B_1$ )-to-gray ( $G_3, G_2, G_1$ ) is shown in Figure 8.17a. From the conversion table we observe that the expressions for the outputs are:

$$G_3 = \Sigma m(4, 5, 6, 7)$$

$$G_2 = \Sigma m(2, 3, 4, 5)$$

$$G_1 = \Sigma m(1, 2, 5, 6)$$

Drawing the K-maps for the Gray code outputs  $G_3, G_2$ , and  $G_1$  in terms of binary inputs  $B_3, B_2, B_1$  and minimizing them as shown in Figure 8.17b, the minimal expressions for  $G_3, G_2$ , and  $G_1$  are:

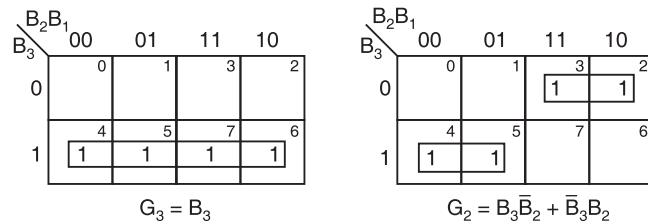
$$G_3 = B_3$$

$$G_2 = B_3\bar{B}_2 + \bar{B}_3B_2$$

$$G_1 = B_2\bar{B}_1 + \bar{B}_2B_1$$

The programming of the PLA circuit to implement the conversion is shown in Figure 8.17c.

The product term generated in each AND gate is listed along the output of the gate in the diagram. The product term is determined from the inputs whose cross points are connected and marked with a  $\times$ . The output of an OR gate gives the logic sum of the selected product terms. The output may be complemented or left in its true form depending on the connection for one of the X-OR gate inputs. A PLD with a programmable polarity feature is shown in Figure 8.18.

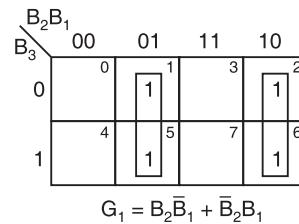


$$G_3 = B_3$$

$$G_2 = B_3\bar{B}_2 + \bar{B}_3B_2$$

Binary			Gray		
$B_3$	$B_2$	$B_1$	$G_3$	$G_2$	$G_1$
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

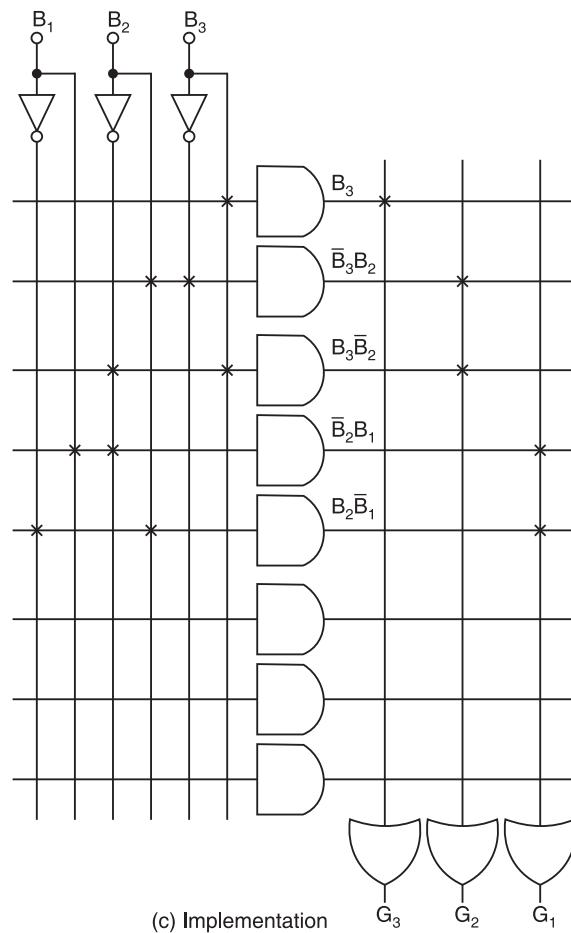
(a) Conversion table



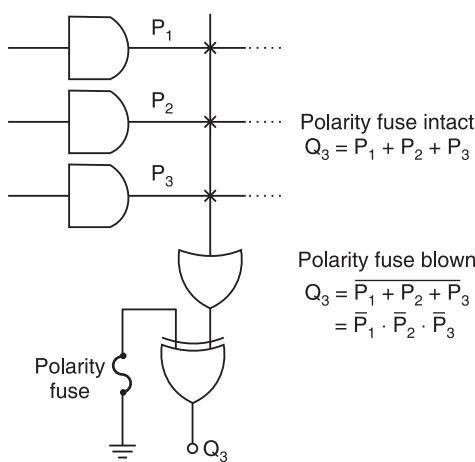
$$G_1 = B_2\bar{B}_1 + \bar{B}_2B_1$$

(b) K-maps

**Figure 8.17** Example 8.8: Use of PLA as a 3-bit binary-to-Gray code converter (Contd.)



**Figure 8.17** Example 8.8: Use of PLA as a 3-bit binary-to-Gray code converter.



**Figure 8.18** PLD with a programmable polarity feature.

### 8.8.1 PLA Programming Table

The fuse map of a PLA can be specified in a tabular form. The PLA programming table consists of three columns. The first column lists the product term numerically. The second column specifies the required paths between inputs and AND gates. The third column specifies the paths between the AND and OR gates. For each output variable, we may have a T (for true) or C (for complement) for programming the X-OR gate. The product terms listed on the left are not part of the table; they are included for reference only. For each product term the inputs are marked with 1, 0, or – (dash). If a variable in the product term appears in its true form, the corresponding input variable is marked with a 1. If it appears complemented, the corresponding input variable is marked with a 0. If the variable is absent in the product term, it is marked as a – (dash).

The paths between the inputs and the AND gates are specified under the column heading *inputs* in the programming table. A 1 in the input column specifies a connection from the input variable to the AND gate. A 0 in the input column specifies a connection from the complement of the variable to the input of the AND gate. A – (dash) specifies a blown fuse in both the input variable and its complement. It is assumed that an open terminal in the input of an AND gate behaves like a 1.

The paths between the AND and OR gates are specified under the column heading *outputs*. The output variables are marked with 1s for those product terms that are included in the function. Each product term that has a 1 in the output column requires a path from the output of the AND gate to the input of the OR gate. Those marked with a – (dash) specify a blown fuse. It is assumed that an open terminal in the input of an OR gate behaves like a 0. Finally, a T (true) output dictates that the other input of the corresponding X-OR gate be connected to 0, and a C (complement) specifies a connection to 1.

The size of a PLA is specified by the number of inputs, the number of product terms, and the number of outputs. For  $n$  inputs,  $k$  product terms, and  $m$  outputs the internal logic of the PLA consists of  $n$  buffer inverter gates,  $k$  AND gates,  $m$  OR gates, and  $m$  X-OR gates.

The implementation of the Boolean functions

$$\begin{aligned} F_1 &= \overline{\overline{A}B} + A\overline{C} + \overline{A}\overline{B}\overline{C} \\ F_2 &= \overline{A}\overline{C} + \overline{B}C \end{aligned}$$

using PLA is shown in Figure 8.19b. The programming table that specifies the PLA of Figure 8.19b is listed in Figure 8.19a.

When designing a digital system with a PLA, there is no need to show the internal connections of the unit as was done in Figure 8.19b. All that is needed is a PLA programming table from which the PLA can be programmed to supply the required logic.

When implementing a combinational circuit with a PLA, careful investigation must be undertaken in order to reduce the number of distinct product terms. Since a PLA has a finite number of AND gates this can be done by simplifying each Boolean function to a minimum number of terms. The number of literals in a term is not important since all the input variables are available any way. Both the true and the complement of each function should be simplified to see which one can be expressed with a fewer product terms and which one provides product terms that are common to other functions.

Product term	Inputs			Outputs	
	A	B	C	(T)	(C)
1	$\bar{A}B$	0	1	—	1 —
2	$A\bar{C}$	1	—	0	1 1
3	$\bar{A}B\bar{C}$	0	1	0	1 —
4	$\bar{B}C$	—	0	1	— 1

(a) PLA programming table

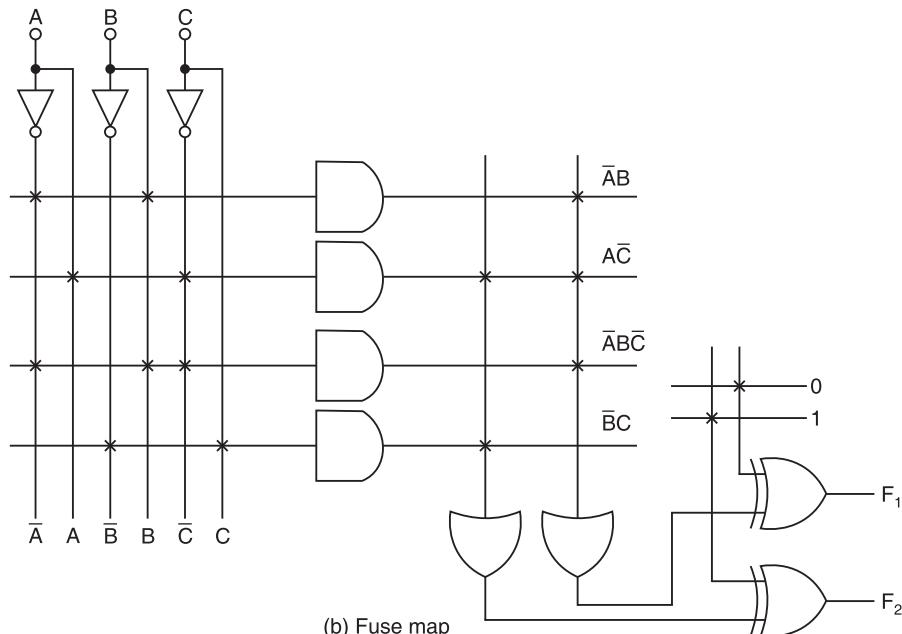


Figure 8.19 Programmable logic array.

**EXAMPLE 8.9** Implement the following two Boolean functions with a PLA:

$$F_1(A, B, C) = \Sigma m(0, 1, 2, 4)$$

$$F_2(A, B, C) = \Sigma m(0, 5, 6, 7)$$

### Solution

The K-maps for the functions  $F_1$  and  $F_2$ , their minimization, and the minimal expressions for both the true and complement forms of those in sum of products are shown in Figure 8.20. For finding the minimal in true form, consider the 1s on the map and for finding the minimal in complement form consider the 0s on the map.

Considering the 1s of  $F_1$

$$F_1(T) = \bar{A}\bar{C} + \bar{B}\bar{C} + \bar{A}\bar{B}$$

Considering the 0s of  $F_1$

$$\bar{F}_1 = AB + AC + BC$$

Therefore,

$$F_1(C) = \overline{(AB + AC + BC)}$$

Considering the 1s of  $F_2$

$$F_2(T) = \bar{A}\bar{B}\bar{C} + AB + AC$$

Considering the 0s of  $F_2$

$$\bar{F}_2 = A\bar{B}\bar{C} + \bar{A}B + \bar{A}C$$

Therefore,

$$F_2(C) = \overline{\bar{A}\bar{B}\bar{C} + \bar{A}B + \bar{A}C}$$

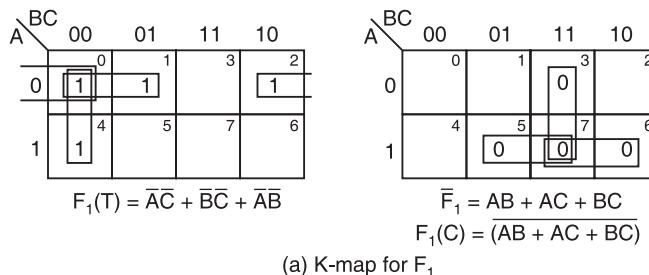
Out of  $F_1(T)$ ,  $F_1(C)$ ,  $F_2(T)$ ,  $F_2(C)$ , the combination that gives the minimum number of product terms is

$$F_1(C) = \overline{(AB + AC + BC)}$$

$$F_2(T) = AB + AC + \bar{A}\bar{B}\bar{C}$$

This gives four distinct terms:  $AB$ ,  $AC$ , and  $BC$  and  $\bar{A}\bar{B}\bar{C}$ . The PLA programming table for this combination is shown in Figure 8.21a. The implementation using a PLA is shown in Figure 8.21b.

$F_1$  is the true output even though a  $C$  is marked over it in the table. This is because  $F_1$  is generated with an AND-OR circuit and is available at the output of the OR gate. The X-OR gate complements the function to produce the true  $F_1$  output.



(a) K-map for  $F_1$

$$\bar{F}_1 = AB + AC + BC$$

$$F_1(C) = \overline{(AB + AC + BC)}$$

(b) K-map for  $F_2$

$$\bar{F}_2 = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B} + \bar{A}C$$

$$F_2(C) = \overline{\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B} + \bar{A}C}$$

Figure 8.20 Example 8.9: K-maps.

Product term	Inputs			Outputs	
	A	B	C	(C)	(T)
				$F_1$	$F_2$
1      AB	1	1	—	1	1
2      AC	1	—	1	1	1
3      BC	—	1	1	1	—
4 $\bar{A}\bar{B}\bar{C}$	0	0	0	—	1

(a) PLA programming table

Figure 8.21 Example 8.9: Programming table and fuse map (Contd.)

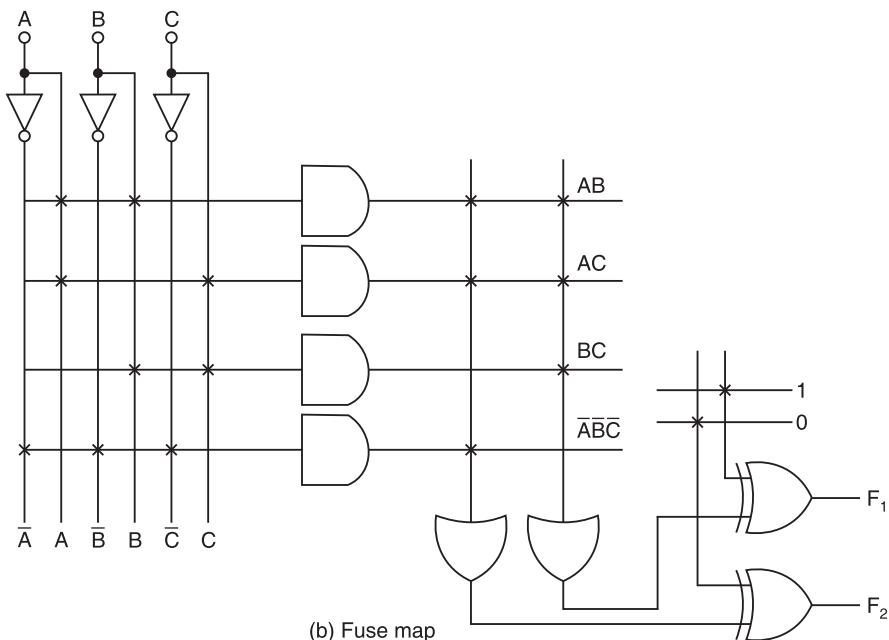


Figure 8.21 Example 8.9: Programming table and fuse map.

**EXAMPLE 8.10** Write the program table to implement a BCD to XS-3 code conversion using a PLA.

**Solution**

The BCD to XS-3 code conversion table and the expressions for the XS-3 outputs are shown in Figure 8.22.

Decimal	BCD Code				XS-3 Code			
	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	E <sub>3</sub>	E <sub>2</sub>	E <sub>1</sub>	E <sub>0</sub>
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

(a) BCD to XS-3 conversion table

$$E_0 = \Sigma m(0, 2, 4, 6, 8)$$

$$E_1 = \Sigma m(0, 3, 4, 7, 8)$$

$$E_2 = \Sigma m(1, 2, 3, 4, 9)$$

$$E_3 = \Sigma m(5, 6, 7, 8, 9)$$

The don't cares are

$$d = \Sigma d(10, 11, 12, 13, 14, 15)$$

(b) Expressions for outputs

Figure 8.22 Example 8.10: Conversion table and expressions for outputs.

To write the PLA program table, obtain the true and complement form of the minimal expressions for outputs using K-maps as shown in Figure 8.23. For true form consider the 1s on the K-maps, and for complement form consider the 0s on the K-maps and obtain the expressions in SOP form in both cases.

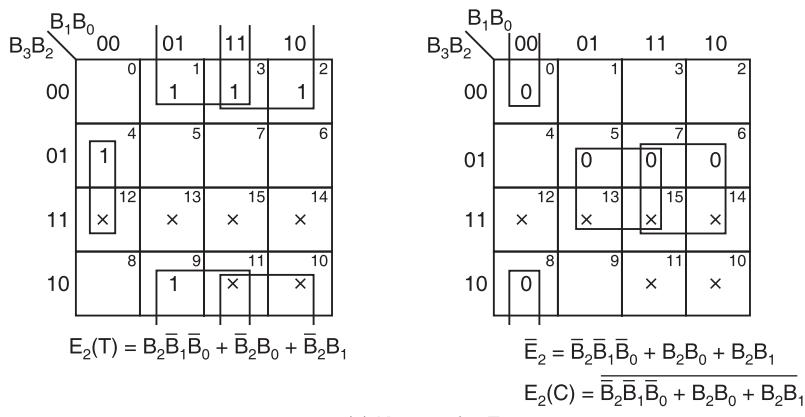
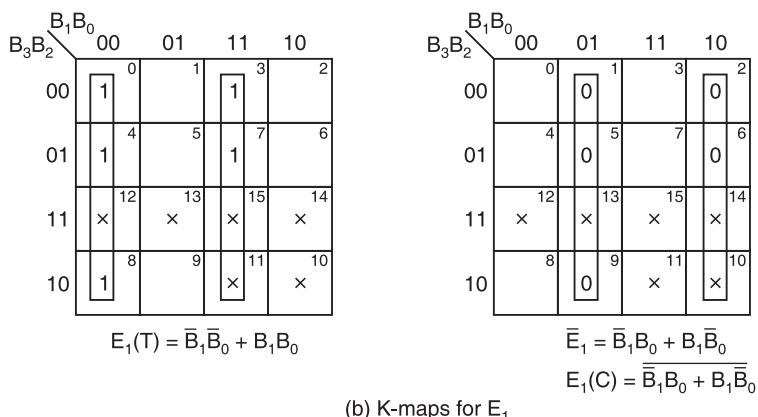
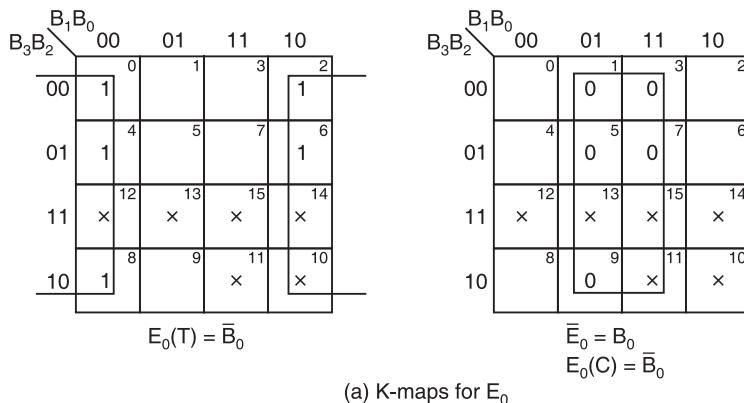


Figure 8.23 Example 8.10: K-maps (Contd.)

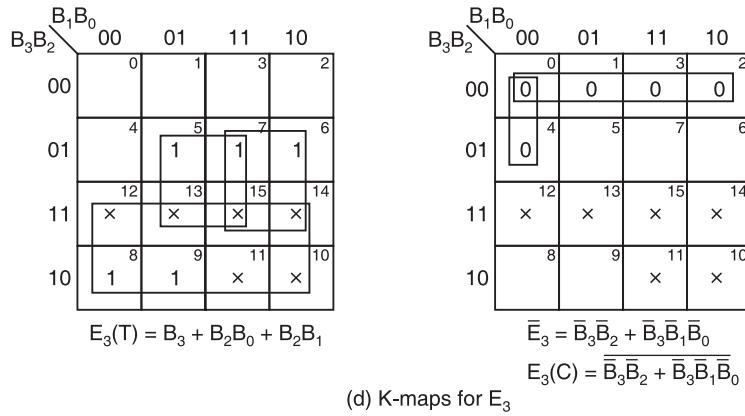
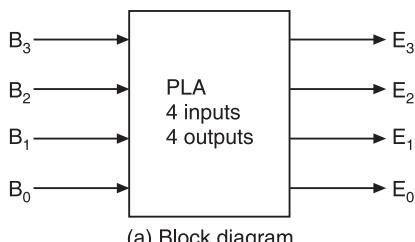


Figure 8.23 Example 8.10: K-maps.

Observing the expressions in true and complement form for  $E_0$ ,  $E_1$ ,  $E_2$  and  $E_3$ , we notice that two terms in  $E_2(C)$  and  $E_3(T)$  are common. So minimal combinations are as follows:

$$E_0(T) = \bar{B}_0, E_1(T) = \bar{B}_1\bar{B}_0 + B_1B_0, E_2(C) = \overline{\bar{B}_2\bar{B}_1\bar{B}_0} + B_2B_0 + B_2B_1, E_3(T) = B_3 + B_2B_0 + B_2B_1$$

The block diagram and the PLA program table to realize the above expressions are shown in Figure 8.24.



Product term	Inputs				Outputs			
	$B_3$	$B_2$	$B_1$	$B_0$	$E_3$	$E_2$	$E_1$	$E_0$
	(T)	(C)	(T)	(T)				
1	$B_3$	1	—	—	1	—	—	—
2	$B_2B_0$	—	1	—	1	1	—	—
3	$B_2B_1$	—	1	1	—	1	1	—
4	$\bar{B}_2\bar{B}_1\bar{B}_0$	—	0	0	0	—	1	—
5	$B_1\bar{B}_0$	—	—	0	0	—	—	1
6	$B_1B_0$	—	—	1	1	—	—	1
7	$\bar{B}_0$	—	—	—	0	—	—	1

(b) PLA programming table

Figure 8.24 Example 8.10: Block diagram and programming table.

**EXAMPLE 8.11** Tabulate the PLA programming table for the four Boolean functions listed below.

$$A(x, y, z) = \Sigma m(1, 2, 4, 6)$$

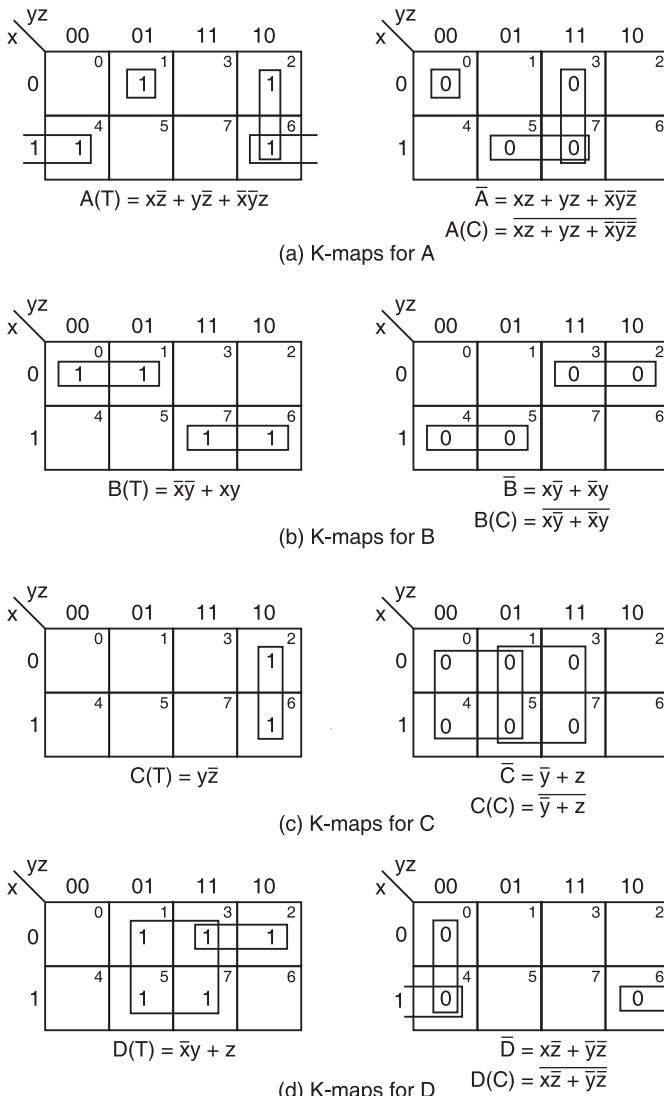
$$B(x, y, z) = \Sigma m(0, 1, 6, 7)$$

$$C(x, y, z) = \Sigma m(2, 6)$$

$$D(x, y, z) = \Sigma m(1, 2, 3, 5, 7)$$

### Solution

The K-maps for the functions A, B, C and D, their minimization, and the minimal expressions for both the true and complement of those in sum of products are shown in Figure 8.25.

**Figure 8.25** Example 8.11: K-maps.

Observing the expressions in true and complement form for A, B, C and D, we notice that the combination that gives a minimum number of product terms is:

$$A(T) = x\bar{z} + y\bar{z} + \bar{x}\bar{y}z \quad B(T) = \bar{x}\bar{y} + xy \quad C(T) = y\bar{z} \quad D(T) = \bar{x}\bar{z} + \bar{y}\bar{z}$$

This gives 6 distinct terms:  $x\bar{z}$ ,  $y\bar{z}$ ,  $\bar{x}\bar{y}z$ ,  $\bar{x}\bar{y}$ ,  $xy$ ,  $\bar{y}\bar{z}$ .

The PLA programming table and the fuse map for this combination are shown in Figure 8.26.

Product term	Inputs			Outputs				
	x	y	z	(T)	(T)	(T)	(C)	
	A	B	C	D				
1	x̄z	1	—	0	1	—	—	1
2	ȳz	—	1	0	1	—	1	—
3	xyz	0	0	1	1	—	—	—
4	xy	0	0	—	—	1	—	—
5	xy	1	1	—	—	1	—	—
6	ȳz	—	0	0	—	—	—	1

(a) PLA programming table

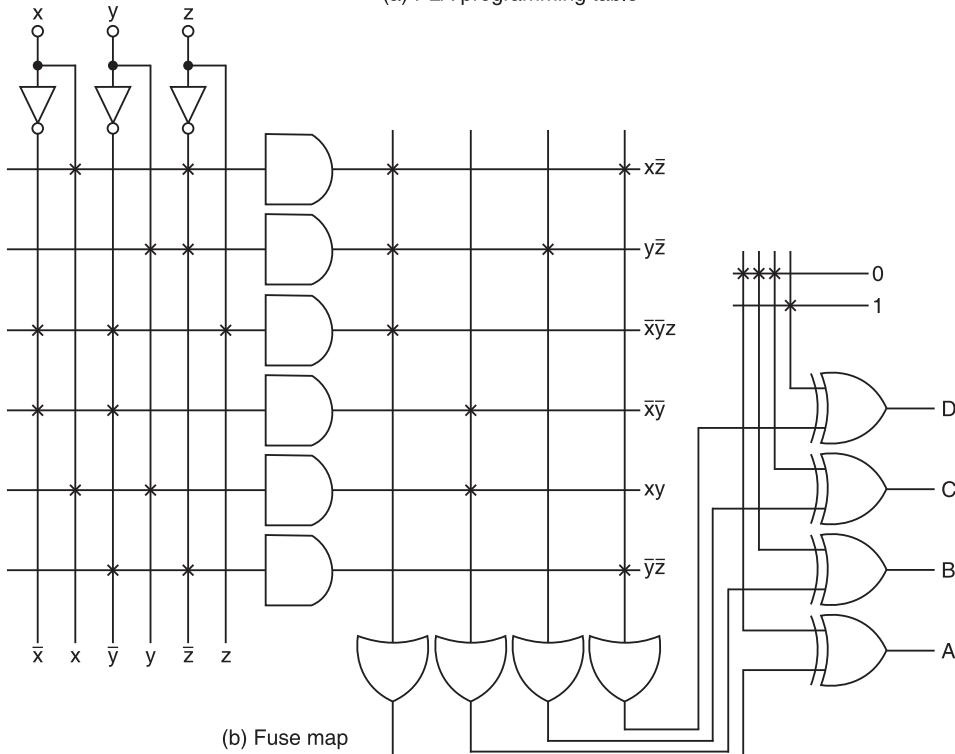


Figure 8.26 Example 8.11: Programming table, and fuse map.

**EXAMPLE 8.12** Give the PLA realization of the following functions using a PLA with 5 inputs, 4 outputs and 8 AND gates:

$$f_1(A, B, C, D, E) = \Sigma m(0, 1, 2, 3, 11, 12, 13, 14, 15, 16, 17, 18, 19, 27, 28, 29, 30, 31)$$

$$f_2(A, B, C, D, E) = \Sigma m(4, 5, 6, 7, 8, 9, 10, 11, 20, 21, 22, 23, 30)$$

### Solution

Drawing the 5-variable K-maps for  $f_1$  and  $f_2$  and simplifying them as shown in Figure 8.27, the minimal expressions for  $f_1$  and  $f_2$  are:

$$f_1(A, B, C, D, E) = \overline{B}\overline{C} + BC + BDE$$

$$f_2(A, B, C, D, E) = \overline{B}C + \overline{A}BC + ACD\overline{E}$$

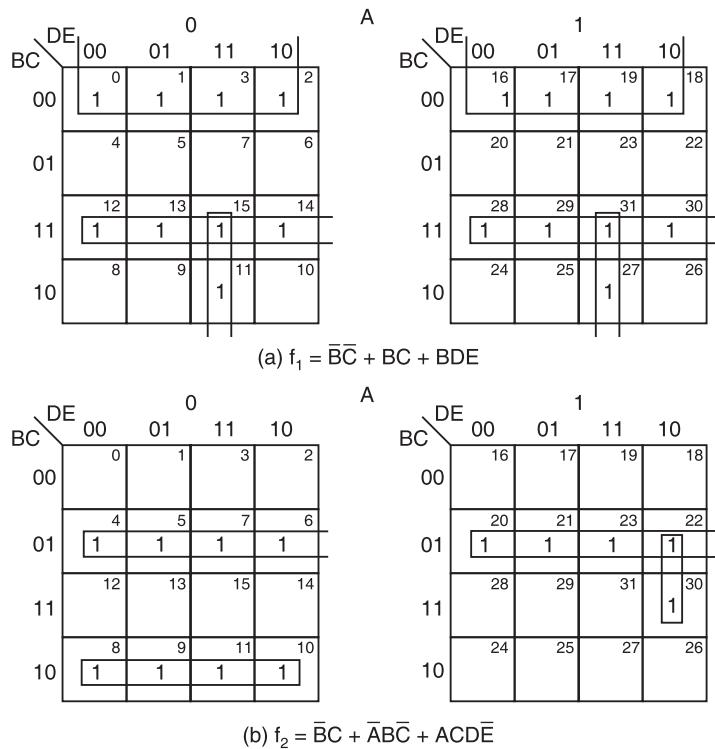


Figure 8.27 Example 8.12: K-maps.

The realization of the minimal expressions using a PLA with 5 inputs, 4 outputs and 8 AND gates is shown in Figure 8.28.

## 8.9 PROGRAMMABLE ROM (PROM)

A programmable ROM can be viewed as a type of programmable logic array and thus used for that purpose. The address inputs to the PROM serve as logic variable inputs and the data output as the node where the output of a logic function is realized. For example, stating that a 1 is stored at address 1001 is the same as stating that the logic function being implemented equals 1 when the input combination is  $A\bar{B}\bar{C}\bar{D}$ . In both the cases, the output will be a 1 when the input is 1001. When we regard a PROM as a PLA, we realize that the AND gates are not programmed. In effect, an AND gate is already in place for every possible combination of the inputs corresponding to every possible address of the PROM. Therefore, to program a PROM as a PLA, we must have a truth table that specifies the value of the function being implemented for every possible combination of the inputs. For each combination where  $F = 1$ , we leave the output of the corresponding AND gate connected to the output OR gate. For each combination where  $F = 0$ , we burn open the connection to the OR gate. We see that a PROM is a PLD with fixed AND gates and programmable OR gates. An  $m \times n$  PROM can be regarded as a PLA having  $n$  programmable OR gates, capable of implementing  $n$  different logic functions of  $m$  variables. A PROM is ideally suited for implementing a logic function directly from a truth table.

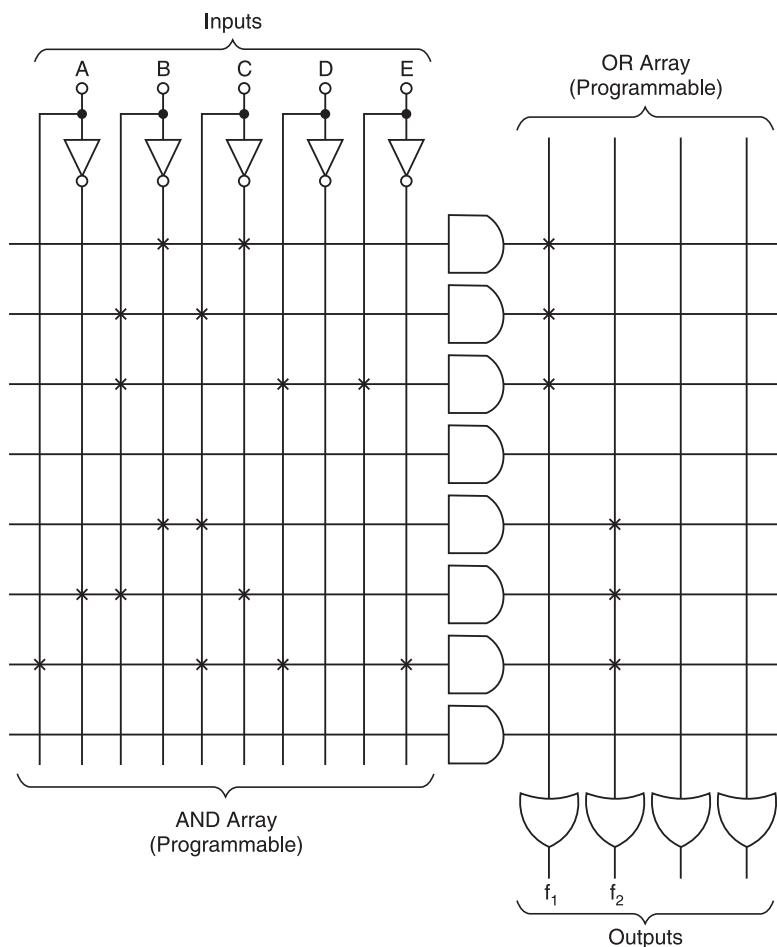


Figure 8.28 Example 8.12: Fuse map.

**EXAMPLE 8.13** Show how an  $8 \times 1$  PROM can be programmed to implement the logic function whose truth table is shown in Figure 8.29a.

**Solution**

From the truth table shown in Figure 8.29a, we observe that the logic function is

$$F = \Sigma m(2, 3, 5, 7) = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}C + ABC$$

No simplification of the expression is to be done for realization with a PROM. Eight AND gates (one for each minterm) and one OR gate (to obtain the SOP form of the single output) are required for its implementation. Figure 8.29c shows the programmed PROM in the simplified connection format of a PLA. To realize  $F$ , address lines 2, 3, 5 and 7 are connected to  $F$ . Logic 1 or a 0 is stored at every address combination corresponding to a combination of the input variables for which the function equals a 1 or a 0. An  $8 \times 1$  PROM means a PROM with 8 address lines and one output. So a  $3 \times 8$  decoder is required as shown in Figure 8.29b.

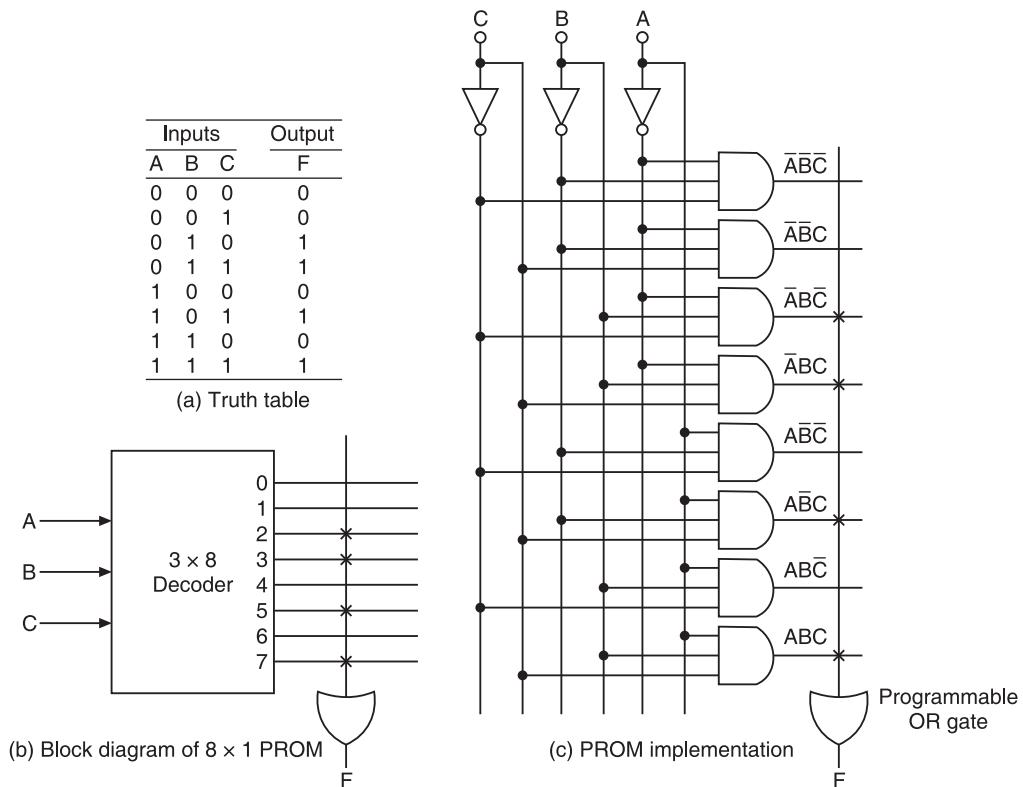


Figure 8.29 Example 8.13.

**EXAMPLE 8.14** Realize the following functions using a PROM of size  $8 \times 3$

$$F_1 = \Sigma m(0, 4, 7)$$

$$F_2 = \Sigma m(1, 3, 6)$$

$$F_3 = \Sigma m(1, 2, 4, 6)$$

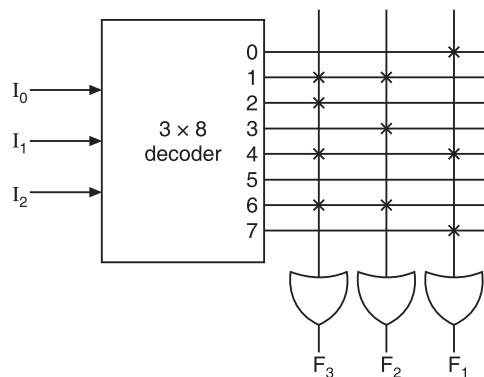
### Solution

An  $8 \times 3$  PROM means a PROM with 8 address lines and 3 outputs. Since the PROM has 8 address lines, we have to use a decoder of size  $3 \times 8$ , i.e. a decoder with 3 input lines. (Number of input lines  $k$  is such that  $8 = 2^k$ ). The number of output functions  $n = 3$ . Since the PROM has fixed AND gates no minimization is required. Realization of the given functions using an  $8 \times 3$  PROM is shown in Figure 8.30. The  $\times$  at the crossover points of the grid indicate the connections. To realize  $F_1$  address lines 0, 4 and 7 are connected to the output line  $F_1$ . To realize  $F_2$  address lines 1, 3 and 6 are connected to the output line  $F_2$ . To realize  $F_3$  address lines 1, 2, 4 and 6 are connected to output line  $F_3$ .

**EXAMPLE 8.15** Realize two outputs  $F_1$  and  $F_2$  using a  $4 \times 2$  PROM:

$$F_1(A_1, A_0) = \Sigma m(0, 2)$$

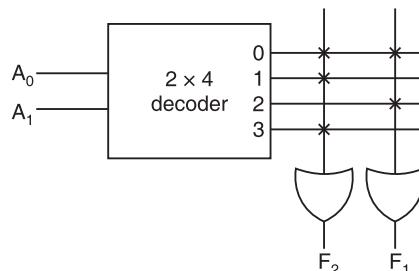
$$F_2(A_1, A_0) = \Sigma m(0, 1, 3)$$



**Figure 8.30** Example 8.14: Realization of functions using an  $8 \times 3$  PROM.

### Solution

Since  $F_1$  and  $F_2$  are functions of two variables  $A_1$  and  $A_0$ , a small PROM of size  $4 \times 2$  is required. A  $4 \times 2$  PROM means a PROM with 4 address lines and two outputs. 4 address lines means two inputs. So a decoder of size  $2 \times 4$  is required.  $F_1$  has two 1s and two 0s whereas  $F_2$  has three 1s and one 0. To realize  $F_1$ , address lines 0 and 2 are connected to the output line  $F_1$  and to realize  $F_2$ , the address lines 0, 1, and 3 are connected to the output line  $F_2$ . Since the PROM has fixed AND gates no minimization is required. The realization is shown in Figure 8.31.



**Figure 8.31** Example 8.15: Realization of functions using a  $2 \times 4$  decoder.

**EXAMPLE 8.16** Design a combinational circuit using a PROM. The circuit accepts a 3-bit binary number and generates its equivalent XS-3 code.

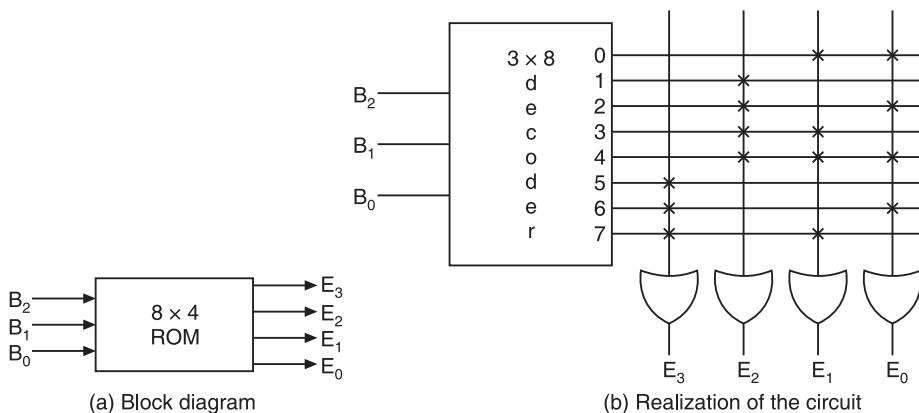
### Solution

The input is a 3-bit binary number. So the PROM requires a  $3 \times 8$  decoder. It generates the equivalent XS-3 code (i.e. 4 bits). So it has 4 outputs and requires 4 OR gates. Since the PROM has fixed AND gates, no minimization is required. Table 8.3 is the truth table for the combinational circuit.

The realization of the combinational circuit using the PROM is shown in Figure 8.32. To realize  $E_0$ , address lines 0, 2, 4 and 6 are connected to the output line  $E_0$ . To realize  $E_1$ , address lines 0, 3, 4 and 7 are connected to the output line  $E_1$ . To realize  $E_2$ , address lines 1, 2, 3 and 4 are connected to the output line  $E_2$ . To realize  $E_3$ , address lines 5, 6 and 7 are connected to the output line  $E_3$ .

**Table 8.3** Example 8.16: Truth table

Inputs			Outputs			
B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	E <sub>3</sub>	E <sub>2</sub>	E <sub>1</sub>	E <sub>0</sub>
0	0	0	0	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	1	0

**Figure 8.32** Example 8.16: Combinational circuit using a PROM.

The PROM can generate any possible logic function of the input variables because it generates every possible AND product term. In general, any application that requires every input combination to be available is a good candidate for a PROM. However, PROMs become impractical when a large number of input variables have to be accommodated, because the number of fuses doubles for each added input variable.

The advantages of using a PROM as a PLD are as follows:

1. Ease of design since no simplification or minimization of logic function is required.
2. Design can be changed, modified rapidly.
3. It is usually faster than discrete SSI/MSI circuit.
4. Cost is reduced.

There are also few disadvantages of ROM-based circuits such as non-utilization of complete circuit, increase of power requirement and enormous increase in size with increase in the number of input variables making it impractical.

## 8.10 PROGRAMMING

When PLDs were first introduced, the logic designer would develop a *fuse map* that showed which fuses to blow. The manufacturer would then program the device according to the fuse map, test it, and return it to the designer. In recent years, the availability of relatively inexpensive programming equipment has made it convenient for users to program their own PLDs. There are universal programmers in the market that can program the most common PROMs, PALs, and FPLAs. The device to be programmed is plugged into a socket on the programmer and the programmer programs and tests the device according to data that have been supplied by the user.

The programming and test data are typically developed by using the commonly available software that will run on standard PCs. Using this software, the user enters the data into the computer describing the logic functions to be programmed into the PLD, as well as information on how the device is to be tested. The software then generates a fuse map and the test data in a form that can be sent over a cable to the PLD programmer's memory. Once the programmer has the data, he can proceed to program and test the device. When finished, the programmer will indicate whether the device has passed or failed the test procedure. If it passes, it can be removed from the programmer's socket and placed in the prototype circuit for further testing.

## 8.11 PROGRAMMABLE LOGIC DEVICES—A COMPARISON

PROM, PLA and PAL have been discussed. These devices are used for synthesizing multiple-output functions. All these basically comprise an AND array followed by an OR array. The programming flexibility for the user in each case is summarized in Table 8.4.

**Table 8.4** Comparison of programmable logic devices

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

### Erasable PLDs

The PLDs we have been talking about are programmed by blowing fuses. Once a fuse is blown, it cannot be reconnected. Thus, if you make a mistake in programming or if you want to change the design, the device will no longer be useful. This drawback has been addressed by several manufacturers who have developed PLDs that can be erased and programmed over and over. These are called *erasable programmable logic devices* (EPLDs). These devices are programmed and erased much like EPROMs and EEPROMs.

## SHORT QUESTIONS AND ANSWERS

**1.** What is a PLD?

**A.** A PLD (programmable logic device) is an IC that contains a large number of gates, flip-flops and registers that are interconnected on the chip. Many of the connections, however, are fusible links which can be broken.

**2.** What is the principal advantage of a PLD?

**A.** The principal advantage of PLDs is that in many applications, they can replace a number of other circuits.

**3.** What is programming?

**A.** The fuse blowing process is called programming.

**4.** What are the advantages of PLDs over fixed function ICs?

**A.** The advantages of PLDs over fixed function ICs are:

- |                            |                              |
|----------------------------|------------------------------|
| (a) Low development cost   | (b) Less space requirement   |
| (c) Less power requirement | (d) High reliability         |
| (e) Easy circuit testing   | (f) Easy design modification |
| (g) High design security   | (h) Less design time         |

**5.** What is a ROM?

**A.** A ROM (Read Only Memory) is essentially a memory device in which permanent information is stored. Data can only be read from it.

**6.** What are the technologies used for the fabrication of ROMs.

**A.** MROMs and PROMs can be fabricated using bipolar or MOS technology but EPROMs and EEPROMs are possible only with MOS technology.

**7.** How is the memory size specified?

**A.** The memory size is specified as  $M \times N$  bits where M is the number of locations and N is the number of bits in each location.

**8.** Is there no provision of entering information in the read only memory? If no what can be read from the memory and if yes why it is called as read only memory.

**A.** There is a provision of entering information in ROM. This process is known as programming. In case of programmable ROMs, the ROM is removed from the circuit and is programmed using a PROM programmer. In the case of non-programmable ROM, the information is entered as a part of the fabrication process itself. Because of the requirement of programming it is known as read only memory.

**9.** A memory has 16-bit address bus. How many locations are there in this?

**A.** The number of memory locations in a memory with 16-bit address bus =  $2^{16} = 65,536 = 64\text{ K}$ .

**10.** What for is the letter 'K' used in memories?

**A.** Digital systems operate on binary numbers and  $2^{10} = 1024$  is represented by 1K.

**11.** What happens to the information stored in memory location after it has been read?

**A.** The reading operation is non-destructive, which means the stored information remains intact after it has been read and can be read any number of times.

**12.** Explain the programming of ROM.

**A.** A ROM is programmed at the time of manufacturing. The information to be entered is supplied by the user. The contents of this are fixed at the time of its fabrication and these can never be changed. That means it cannot be erased.

## 496 FUNDAMENTALS OF DIGITAL CIRCUITS

13. Is the ROM a volatile memory? Explain.  
A. Programming of ROM involves making of the required interconnections at the time of fabrication and, therefore, its contents are unaffected even when the power is off. Thus it is a non-volatile memory.
14. What is a PROM?  
A. A PROM is a ROM which can be programmed.
15. What does a  $32 \times 8$  ROM contain?  
A. A  $32 \times 8$  ROM contains 32 words (addresses) of 8 bits each.
16. Which decoder is contained in a  $32 \times 8$  ROM?  
A. A  $32 \times 8$  ROM contains a  $5 \times 32$  decoder.
17. How many OR gates are there in a  $32 \times 8$  ROM and how many inputs does each OR gate of a  $32 \times 8$  ROM have?  
A. A  $32 \times 8$  ROM contains 8 OR gates and each OR gate has 32 inputs.
18. What should be the size of a ROM to produce the square of a 3-bit input.  
A. The size of the ROM to produce the square of a 3-bit input is  $8 \times 6$ .
19. What is the size of the decoder in a  $32 \times 4$  ROM?  
A. A  $32 \times 4$  ROM contains a  $5 \times 32$  decoder.
20. What is the size of the decoder in a  $8 \times 4$  ROM?  
A. An  $8 \times 4$  ROM contains a  $3 \times 8$  decoder.
21. What are the types of ROMs?  
A. The various types of ROMs are:
  - (a) The mask programmed ROMs (MROMs)
  - (b) Programmable read only memories (PROMs)
  - (c) Erasable programmable read only memories (EPROMs)
  - (d) Electrically erasable programmable read only memories (EEPROMs)
22. What is an MROM?  
A. An MROM is a ROM which has its storage locations written into (programmed) by the manufacturer during the last fabrication process of the unit according to the customer's specifications. A major disadvantage of MROM is that it cannot be reprogrammed in the event of a design change requiring modification of stored data.
23. What is a PROM?  
A. A PROM is a field programmable ROM. It is not programmed during the manufacturing processes but is custom programmed by the user. Once programmed, the data cannot be altered. PROMs are manufactured with fusible links.
24. Is the PROM volatile or non-volatile?  
A. It is non-volatile similar to the ROM.
25. What are the technologies used for the fabrication of ROMs.  
A. MROMs and PROMs can be fabricated using bipolar or MOS technologies, but EPROMs and EEPROMs are possible only with MOS technology.
26. How can a ROM device be considered as a combinational circuit?  
A. A ROM device has M locations and N bits are stored at each location. Each location has its unique address. Therefore, if the signals corresponding to the input variables are applied at the address input pins of ROM, then the contents stored at that location are available at output pins. Thus, it operates as a combinational circuit.
27. Is it possible to locate any ROM location at random?  
A. Any ROM location can be selected for reading (or accessed) at random by applying the corresponding input variables at the address input pins.

- 28.** Is it possible to design multiple output circuits using ROM?
- A. Multiple output circuits can be designed using ROM by selecting a ROM which has the number of bits at each location at least equal to the number of outputs desired.
- 29.** What is an EPROM?
- A. An EPROM is a ROM whose contents can be erased and reprogrammed enabling the device to be used repeatedly. Its contents can be erased by exposing it to ultraviolet light. Selective erasure is not possible.
- 30.** Differentiate between PROM and EPROM.
- A. A PROM can be programmed only once whereas an EPROM can be programmed any number of times.
- 31.** In an EPROM chip, is it possible to erase the contents of only some of the locations? If not, why?
- A. No. When the chip is exposed to UV radiation, all the contents get erased simultaneously. It is not possible to erase some locations leaving the contents of remaining locations intact.
- 32.** Some information is stored in EPROM which is required to be modified. How will you do it?
- A. First, whatever is stored should be erased by exposing the EPROMs to UV radiation and then it is programmed to store the new information.
- 33.** What is an EEPROM?
- A. An EEPROM or EAROM is a ROM whose contents can be electrically erased and reprogrammed enabling the device to be used repeatedly. Selective erasure is possible.
- 34.** What are the two major disadvantages of EPROM?
- A. The two major disadvantages of EPROM are as follows:
- (a) They have to be removed from their sockets in order to be erased and reprogrammed.
  - (b) The erasure removes the complete memory contents. This necessitates complete reprogramming even when one memory word has to be changed.
- 35.** What is a combinational PLD?
- A. A combinational PLD is an IC with programmable gates divided into an AND array and an OR array to provide an AND-OR sum of products implementation.
- 36.** What are the major types of combinational PLDs?
- A. There are three major types of combinational PLDs and they differ in the placement of the programmable connection in the AND-OR array. The various PLDs used are PALs (programmable array logic), PLAs (programmable logic arrays), and PROMs (programmable read only memories).
- 37.** What is a PROM?
- A. A PROM is a combinational PLD with a fixed AND array and a programmable OR array.
- 38.** What is a PAL?
- A. A PAL is a combinational PLD with a programmable AND array and a fixed OR array.
- 39.** What is a PLA?
- A. A PLA is a combinational PLD with both programmable AND and OR arrays.
- 40.** What is FPLA?
- A. FPLA is a field programmable logic array. It can be programmed by the user by means of certain recommended procedures.
- 41.** What is the programming table of a PLA?
- A. The programming table of a PLA is a table specifying the fuse map of a PLA.
- 42.** How is the size of a PLA specified?
- A. The size of a PLA is specified by the number of inputs, the number of product terms and the number of outputs.

- 43.** What is a fuse map?  
 A. A fuse map is a map that shows which fuses to blow.
- 44.** What is the basic architecture of a PLA?  
 A. A PLA consists of an array of programmable AND and OR gates. The number of inputs to every AND gate is twice the number of input variables possible for a chip, and the number of inputs to every OR gate is equal to the number of AND gates. The outputs of the OR gates give the output of the realized logic functions.
- 45.** How is the capacity of a PLA specified?  
 A. The capacity of a PLA is specified as the number of inputs, product terms and outputs.
- 46.** Are PLAs and FPLAs volatile or non-volatile?  
 A. PLAs and FPLAs are non-volatile.
- 47.** Are erasable and programmable PLAs available?  
 A. No. Erasable and programmable PLAs are not available.
- 48.** Is it possible to share the product terms between different outputs in a PLA? If yes, how?  
 A. Yes. Since each OR gate may be connected to all the product terms, the output of the AND gate with product terms required for different output functions can be connected to corresponding OR gates.
- 49.** Is it possible to share the product terms between different outputs in a PAL? Justify your answer.  
 A. No. In a PAL the OR gates are non-programmable. Every AND gate can supply input to only one OR gate. Therefore, it is not possible to share the product terms between different outputs.
- 50.** What are the advantages and disadvantages of using a PROM as a PLD?  
 A. The advantages of using a PROM as a PLD are as follows:  
 (a) Ease of design since no simplification or minimization of logic function is required.  
 (b) Design can be changed, modified rapidly.  
 (c) It is usually faster than discrete SSI/MSI circuit.  
 (d) Cost is reduced.
- The disadvantages of using ROM-based circuits as PLDs are as follows:  
 (a) Non-utilization of complete circuit.  
 (b) Increase of power requirement and enormous increase in size with increase in the number of input variables making it impractical.
- 51.** Compare the three combinational PLDs—PROM, PLA, and PAL.  
 A. All the three combinational PLDs—PROM, PLA and PAL are used for synthesizing multiple-output functions. All these basically comprise an AND array followed by an OR array. The programming flexibility for the user in each case is summarized below.

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

**REVIEW QUESTION**

- Give the comparison between PROM, PLA and PAL.

**FILL IN THE BLANKS**

- ROM is a \_\_\_\_\_ in which permanent information is stored.
- A ROM which can be programmed is called a \_\_\_\_\_.
- A  $32 \times 10$  ROM contains a \_\_\_\_\_ decoder.
- A  $16 \times 5$  ROM contains a \_\_\_\_\_ decoder.
- The various types of ROMs are \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_.
- A ROM which has its storage locations written into by the manufacturer is called a \_\_\_\_\_.
- To have a ROM of size 16 KB, the number of  $1024 \times 4$  ROMs required is \_\_\_\_\_.
- A ROM of 128 KB can be used to design a combinational circuit of a maximum of \_\_\_\_\_ variables.
- The contents of an EPROM can be erased by exposing it to \_\_\_\_\_.
- The contents of an \_\_\_\_\_ can be erased electrically.
- Selective erasing is not possible in \_\_\_\_\_ but possible in \_\_\_\_\_.
- The \_\_\_\_\_ process is called programming.
- \_\_\_\_\_ is an IC with programmable gates divided into an AND array and an OR array to provide an AND-OR SOP implementation.
- The three major types of combinational PLDs are: \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_.
- A PROM is a combinational PLD with a fixed \_\_\_\_\_ array and programmable \_\_\_\_\_ array.
- A PAL is a combinational PLD with a fixed \_\_\_\_\_ array and a programmable \_\_\_\_\_ array.
- A PLA is a combinational PLD with \_\_\_\_\_ AND array and a \_\_\_\_\_ OR array.
- A PLA that can be programmed by the user is called a \_\_\_\_\_.
- A table specifying the fuse map is called a \_\_\_\_\_.

**OBJECTIVE TYPE QUESTIONS**

- A ROM which can be programmed is called a
 

(a) MROM	(b) PROM	(c) EPROM	(d) EEPROM
----------	----------	-----------	------------
- A  $32 \times 10$  ROM contains a decoder of size
 

(a) $5 \times 32$	(b) $32 \times 32$	(c) $32 \times 10$	(d) $10 \times 32$
-------------------	--------------------	--------------------	--------------------
- An  $8 \times 4$  ROM contains a decoder of size
 

(a) $3 \times 8$	(b) $8 \times 4$	(c) $8 \times 8$	(d) $3 \times 4$
------------------	------------------	------------------	------------------
- A  $16 \times 5$  ROM stores
 

(a) 4 words of 16 bits each	(b) 16 words of 5 bits each
(c) 16 words of 4 bits each	(d) 5 words of 16 bits each

**500** FUNDAMENTALS OF DIGITAL CIRCUITS



## PROBLEMS

- 8.1** Design a BCD-to-XS-3 code converter using a (a) PROM, (b) PLA and (c) PAL.

**8.2** Design an XS-3-to-BCD code converter using a (a) PROM, (b) PLA and (c) PAL.

**8.3** Implement the following Boolean functions using PLA:

$$f_1(A, B, C) = \Sigma m(0, 1, 3, 5), f_2(A, B, C) = \Sigma m(0, 3, 5, 7)$$

**8.4** Implement the following Boolean functions using PAL:

$$f_1(w, x, y, z) = \Sigma m(0, 2, 5, 7, 8, 10, 12, 13)$$

$$f_2(w, x, y, z) = \Sigma m(0, 2, 6, 8, 9, 14, 15)$$

$$f_3(w, x, y, z) = \Sigma m(0, 8, 14, 15)$$

$$f_4(w, x, y, z) = \Sigma m(0, 1, 2, 4, 5, 8, 9, 10)$$

**8.5** Tabulate the PLA programming table for the four Boolean functions listed below.

$$f_1(A, B, C) = \Sigma m(0, 1, 2, 4, 6)$$

$$f_2(A, B, C) = \Sigma m(0, 2, 6, 7)$$

$$f_3(A, B, C) = \Sigma m(3, 6)$$

$$f_4(A, B, C) = \Sigma m(1, 3, 5, 7)$$

## VHDL PROGRAMS

### 1. VHDL PROGRAM FOR 4:16 ROM USING BEHAVIORAL MODELING

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity rom16x8 is
    port ( address : in std_logic_vector(3 downto 0);
           data : out std_logic_vector(7 downto 0) );
end rom16x8;
architecture behavioral of rom16x8 is
type memory is array (0 to 2**4 - 1) of std_logic_vector(7 downto 0);
constant my_rom : memory := (
    0 => "00000000",
    1 => "00000001",
    2 => "00000010",
    3 => "00000011",
    4 => "00000100",
    5 => "11110001",
    6 => "11110010",
    7 => "11110011",
    8 => "11110100",
    9 => "11110101",
   10 => "11110110",
   11 => "11110111",
   12 => "11111000",
   13 => "11111001",
   14 => "11111010",
   15 => "11111011");
begin
process (address)
begin
    case address is
        when "0000" => data <= my_rom(0);
        when "0001" => data <= my_rom(1);
        when "0010" => data <= my_rom(2);
        when "0011" => data <= my_rom(3);
        when "0100" => data <= my_rom(4);
        when "0101" => data <= my_rom(5);
        when "0110" => data <= my_rom(6);
        when "0111" => data <= my_rom(7);
    end case;
end process;
end;

```

```

        when "1000" => data <= my_rom(8);
        when "1001" => data <= my_rom(9);
        when "1010" => data <= my_rom(10);
        when "1011" => data <= my_rom(11);
        when "1100" => data <= my_rom(12);
        when "1101" => data <= my_rom(13);
        when "1110" => data <= my_rom(14);
        when "1111" => data <= my_rom(15);
        when others => data <= "xxxxxxxx";
    end case;
end process;
end behavioral;
```

**SIMULATION OUTPUT:**

+ ◆ /rom16x8/address	0101	0001	1001	1111	1010	1101	)0101
+ ◆ /rom16x8/data	11110001	00000001	11110101	11110111	11110110	11110001	11110001

**2. VHDL PROGRAM IN STRUCTURAL MODELING FOR A COMBINATIONAL CIRCUIT USING ROM. THE CIRCUIT ACCEPTS A 3-BIT NUMBER AND GENERATES AN OUTPUT EQUAL TO THE SQUARE OF THE INPUT NUMBER.**

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity square_3bitnum is
    port ( a : in std_logic_vector (2 downto 0);
           b : out std_logic_vector (5 downto 0));
end square_3bitnum;
architecture structural of square_3bitnum is
component rom8x4 is
    port ( a : in std_logic_vector (2 downto 0);
           b : out std_logic_vector (3 downto 0));
end component;
begin
b(0)<=a(0);
b(1)<='0';
x1: rom8x4 port map(a,b(5 downto 2));
end structural;
```

**COMPONENT INTANTIATION**

**VHDL PROGRAM FOR ROM8X4**

```
library ieee;
```

```

use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity rom8x4 is
    port ( a : in std_logic_vector (2 downto 0);
           b : out std_logic_vector (3 downto 0));
end rom8x4;
architecture structural of rom8x4 is
component decoder3x8 is
    port ( i : in std_logic_vector (2 downto 0);
           y : out std_logic_vector (7 downto 0));
end component;
component orgate_3 is
    port ( a,b,c: in std_logic;
           y: out std_logic);
end component;
component orgate is
    port ( a,b : in std_logic;
           y : out std_logic);
end component;
signal y:std_logic_vector(7 downto 0);
begin
x1:decoder3x8 port map (a,y);
x2:orgate port map(y(2),y(6),b(0));
x3:orgate port map(y(3),y(5),b(1));
x4:orgate_3 port map(y(4),y(5),y(7),b(2));
x5:orgate port map(y(6),y(7),b(3));
end structural;

```

### VHDL PROGRAM FOR DECODER3:8

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity decoder3x8 is
    port ( i : in std_logic_vector (2 downto 0);
           y : out std_logic_vector (7 downto 0));
end decoder3x8;
architecture behavioral of decoder3x8 is
begin
process(i)
begin
    if (i="000") then y<="00000001";
    elsif (i="001") then y<="00000010";

```

## 506 FUNDAMENTALS OF DIGITAL CIRCUITS

```
elsif (i="010") then y<="00000100";
elsif (i="011") then y<="00001000";
elsif (i="100") then y<="00010000";
elsif (i="101") then y<="00100000";
elsif (i="110") then y<="01000000";
elsif (i="111") then y<="10000000";
else y<="00000000";
end if;
end process;
end behavioral;
```

### VHDL PROGRAM FOR 3-INPUT ORGATE

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity orgate_3 is
    port ( a,b,c: in std_logic;
           y: out std_logic);
end orgate_3;
architecture dataflow of orgate_3 is
begin
y<= a or b or c;
end dataflow;
```

### VHDL PROGRAM FOR 2-INPUT ORGATE

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity orgate is
    port ( a,b : in std_logic;
           y : out std_logic);
end orgate;
architecture dataflow of orgate is
begin
y<= a or b;
end dataflow;
```

### SIMULATION OUTPUT:

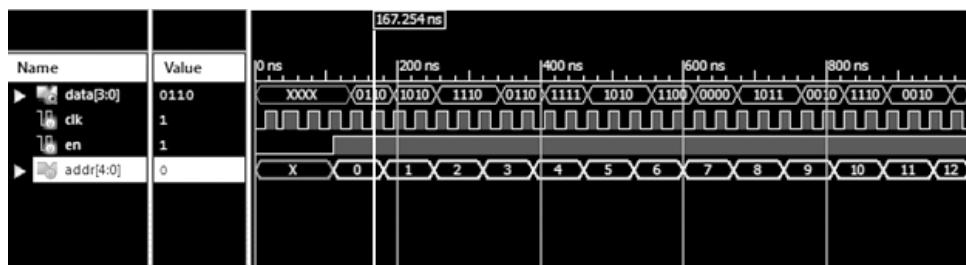
+---/square_3bitnum/a	111	001	)010	)011	)100	)111
+---/square_3bitnum/b	110001	000001	)000100	)001001	)010000	)110001

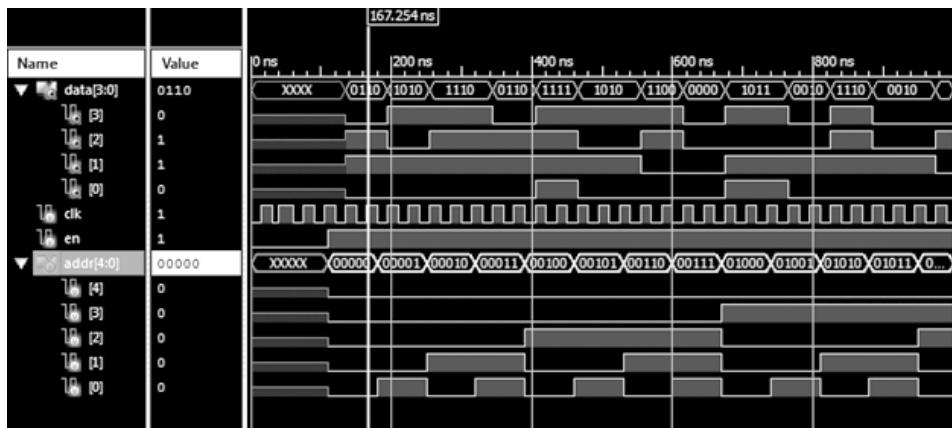
## VERILOG PROGRAMS

### 1. VERILOG PROGRAM FOR 4:16 ROM USING BEHAVIORAL MODELING

```
module ROM (clk, en, addr, data);
input clk,en;
input [4:0] addr;
output reg [3:0] data;
always @(posedge clk)begin
if (en)
case(addr)
5'b00000: data = 4'b0110;
5'b00001: data = 4'b1010;
5'b00010: data = 4'b1110;
5'b00011: data = 4'b0110;
5'b00100: data = 4'b1111;
5'b00101: data = 4'b1010;
5'b00110: data = 4'b1100;
5'b00111: data = 4'b0000;
5'b01000: data = 4'b1011;
5'b01001: data = 4'b0010;
5'b01010: data = 4'b1110;
5'b01011: data = 4'b0010;
5'b01100: data = 4'b0100;
5'b01101: data = 4'b1010;
5'b01110: data = 4'b1100;
5'b01111: data = 4'b0110;
default: data = 4'b0000;
endcase
end
endmodule
```

### SIMULATION OUTPUT:





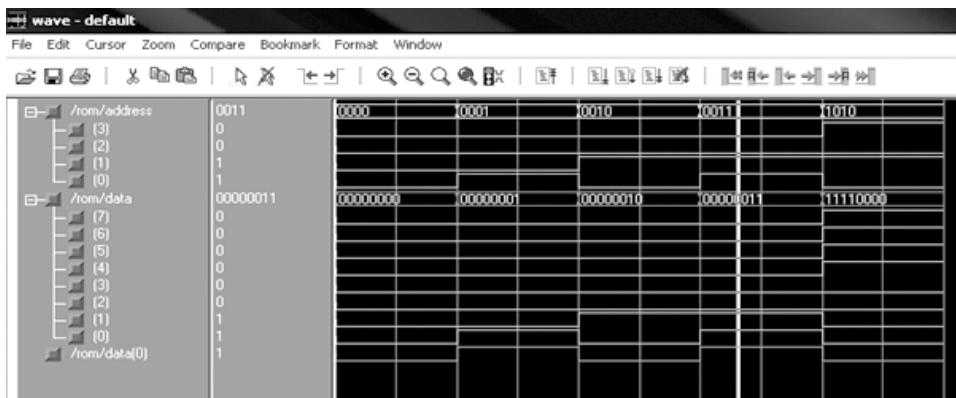
## 2. VERILOG PROGRAM FOR 8-BIT ROM USING BEHAVIORAL MODELING

```

module rom(data_out, address);
    input [7:0] address;
    output [7:0] data_out;
    reg [7:0]data_out;
    reg [0:7]mem[0:7];
    initial
    begin
        mem[0]=8'b00000000;
        mem[1]=8'b00000010;
        mem[2]=8'b00000100;
        mem[3]=8'b00001000;
        mem[4]=8'b00010000;
        mem[5]=8'b00100000;
        mem[6]=8'b01000000;
        mem[7]=8'b10000000;
    end
    always@(address)
    begin
        data_out=mem[address];
    end
endmodule

```

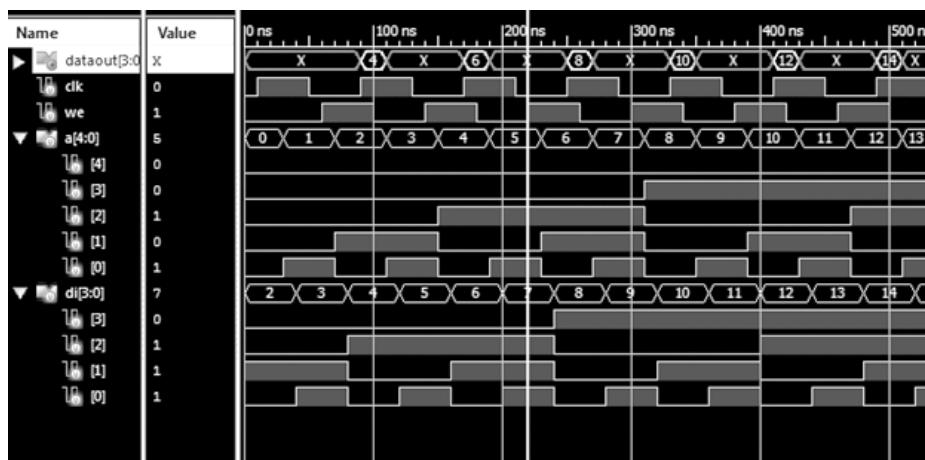
## SIMULATION OUTPUT:



## 3. VERILOG PROGRAM FOR 4-BIT RAM USING BEHAVIORAL MODELING

```
module ram_4Bit (clk, we, a, di, dataout);
input clk;
input we;
input [4:0] a;
input [3:0] di;
output [3:0] dataout;
reg [3:0] ram [31:0];
always @ (posedge clk) begin
if (we)
ram[a] <= di;
end
assign dataout = ram[a];
endmodule
```

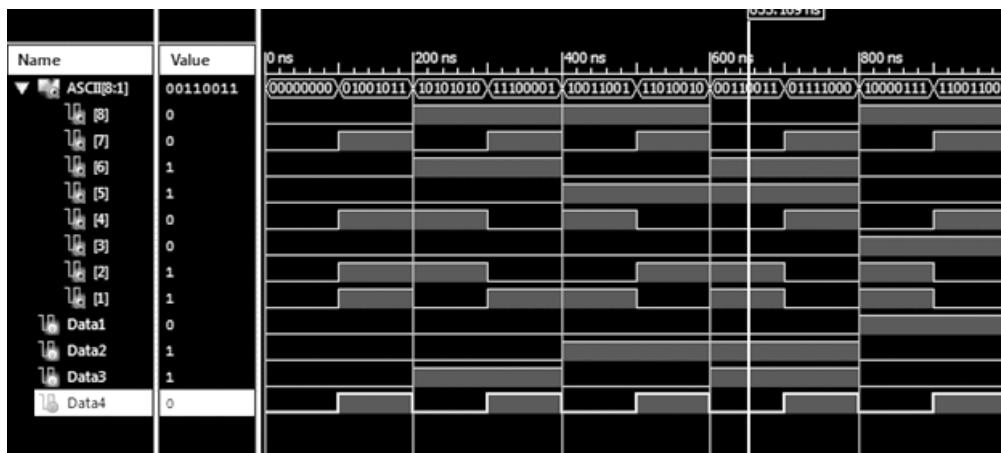
## SIMULATION OUTPUT:



#### 4. VERILOG PROGRAM FOR 4-BIT BCD-TO- ASCII CONVERTER USING BEHAVIORAL MODELING

```
module BCD_ASCII( input Data1,Data2,Data3,Data4,output reg [1:8]
ASCII );
reg P1;
reg P2;
reg P3;
reg TotalParity;
always @(Data1,Data2,Data3,Data4)
begin
P1 = (Data1^Data2^Data4);
P2 = (Data1^Data3^Data4);
P3 = (Data2^Data3^Data4);
TotalParity = (P1^P2^Data1^P3^Data2^Data3^Data4);
ASCII[1] = P1;
ASCII[2] = P2;
ASCII[3] = Data1;
ASCII[4] = P3;
ASCII[5] = Data2;
ASCII[6] = Data3;
ASCII[7] = Data4;
ASCII[8] = TotalParity;
end
endmodule
```

#### SIMULATION OUTPUT:



#### 5. VERILOG PROGRAM IN BEHAVIORAL MODELING FOR A COMBINATIONAL CIRCUIT USING ROM. THE CIRCUIT ACCEPTS A 3-BIT NUMBER AND GENERATES AN OUTPUT EQUAL TO THE SQUARE OF THE INPUT NUMBER

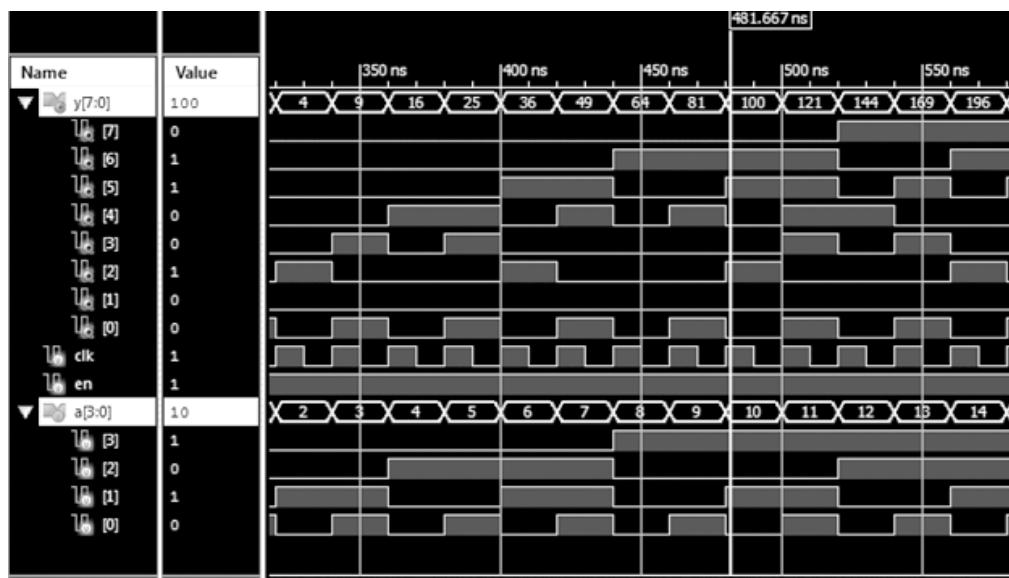
```
module rom_mem (clk, en, a, y);
input clk,en;
```

```

input [3:0] a;
output reg [7:0] y;
reg [3:0] data;
always @(posedge clk)begin
if (en)
case(a)
4'b0001: data = 4'b0001;
4'b0010: data = 4'b0010;
4'b0011: data = 4'b0011;
4'b0100: data = 4'b0100;
4'b0101: data = 4'b0101;
4'b0110: data = 4'b0110;
4'b0111: data = 4'b0111;
4'b1000: data = 4'b1000;
4'b1001: data = 4'b1001;
4'b1010: data = 4'b1010;
4'b1011: data = 4'b1011;
4'b1100: data = 4'b1100;
4'b1101: data = 4'b1101;
4'b1110: data = 4'b1110;
4'b1111: data = 4'b1111;
default: data = 4'b0000;
endcase
y = data * data;
end
endmodule

```

### SIMULATION OUTPUT:



# 9

## THRESHOLD LOGIC

### 9.1 INTRODUCTION

In the earlier chapters, the techniques of designing various logic circuits using AND, OR and NOT gates were discussed. Implementation using universal gates such as NAND gates and NOR gates was also discussed. In this chapter an entirely new concept is presented. The threshold element, also called the threshold gate (T-gate), is a much more powerful device than any of the conventional logic gates such as NAND, NOR and others. Complex, large Boolean functions can be realized using much fewer threshold gates. Frequently a single threshold gate can realize a very complex function which otherwise might require a large number of conventional gates. Also the interconnection of components is much simpler than the circuits using gates. Eventhough the T-gate offers incomparably economical realization, it has not found extensive use with the digital system designers mainly because of the following limitations.

1. It is very sensitive to parameter variations.
2. It is difficult to fabricate it in IC form.
3. The speed of switching of threshold elements is much lower than that of conventional gates.

The basic principles of threshold logic are presented in this chapter.

### 9.2 THE THRESHOLD ELEMENT

A pictorial definition of a T-gate is given in Figure 9.1. Like conventional gates, a threshold element or gate has  $n$  binary inputs  $x_1, x_2, \dots, x_n$ ; and a single binary output  $F$ . But in addition to those, it has

two more parameters. Its parameters are a threshold  $T$  and weights  $w_1, w_2, \dots, w_n$ . The weights  $w_1, w_2, \dots, w_n$  are associated with the input variables  $x_1, x_2, \dots, x_n$ . The value of the threshold ( $T$ ) and weights ( $w_i$ s) may be real, finite, positive or negative numbers. The symbol of a threshold element is shown in Figure 9.1. It is represented by a circle partitioned into two parts, one part represents the weights and the other represents  $T$ . The behaviour of a threshold element is specified by a relationship between its inputs and output involving the parameters  $w_i$ s and  $T$ . It is defined as

$$F(x_1, x_2, \dots, x_n) = 1, \text{ if and only if } \sum_{i=1}^n w_i x_i \geq T,$$

otherwise

$$F(x_1, x_2, \dots, x_n) = 0$$

Here the sum and product operations are normal arithmetic operations and the sum  $\sum_{i=1}^n w_i x_i$  is called the weighted sum of the element or gate.

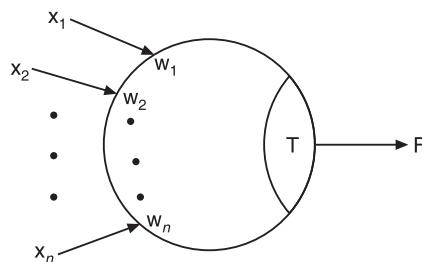


Figure 9.1 Threshold gate.

### 9.3 CONSTRUCTION OF THRESHOLD GATE

A threshold gate or element is constructed in many ways. Construction of threshold gate using resistor transistor logic is shown in Figure 9.2. When all the inputs are at logic 0 level (positive logic), the base emitter junction is reverse biased, so the transistor will be off and the outputs will

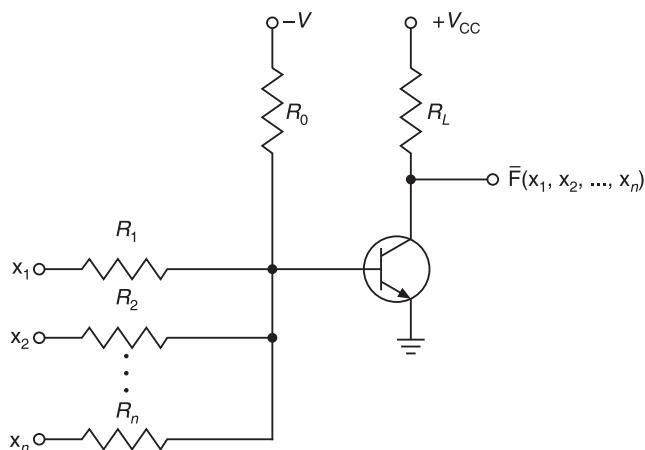


Figure 9.2 Threshold gate using resistor transistor logic.

be at logic 1 level. The base to emitter voltage depends on the weighted sum of the inputs determined by  $R_0$  and the input resistors. When this weighted sum equals or exceeds a certain value, the transistor goes into saturation to make the output logical 0. This is exactly opposite to the definition of threshold gate. Hence this threshold gate is called a complemented threshold gate. The threshold is determined by  $R_0$  since all resistances have only positive values; this gate is capable of providing only positive weights. Using magnetic core as a threshold gate, it is possible to realize positive as well as negative weights depending on the direction of the windings.

**EXAMPLE 9.1** Obtain the minimal Boolean expression from the threshold gate shown in Figure 9.3.

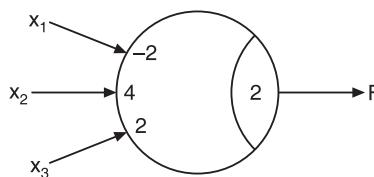


Figure 9.3 Example 9.1: Threshold gate.

### Solution

Figure 9.3 shows the threshold gate with three inputs  $x_1, x_2, x_3$  with weights  $-2$  ( $w_1$ ),  $4$  ( $w_2$ ), and  $2$  ( $w_3$ ) respectively. The value of threshold is  $2$  ( $T$ ). Table 9.1 shows the weighted sums and outputs for all input combinations. For this threshold gate, the weighted sum is

$$\begin{aligned} w &= w_1 x_1 + w_2 x_2 + w_3 x_3 \\ &= (-2)x_1 + (4)x_2 + (2)x_3 \\ &= -2x_1 + 4x_2 + 2x_3 \end{aligned}$$

The output  $F$  is logic 1 for  $w \geq 2$  and it is logic 0 for  $w < 2$ .

Table 9.1 Input-output relation of the gate shown in Figure 9.3

Input variables			Weighted sum	Output
$x_1$	$x_2$	$x_3$	$w = -2x_1 + 4x_2 + 2x_3$	$F$
0	0	0	0	0
0	0	1	2	1
0	1	0	4	1
0	1	1	6	1
1	0	0	-2	0
1	0	1	0	0
1	1	0	2	1
1	1	1	4	1

From the input-output relation given in Table 9.1, we can see that the Boolean expression for the output is

$$F = \Sigma m(1, 2, 3, 6, 7)$$

The K-map for F, its minimization and the minimal expression obtained from it are shown in Figure 9.4.

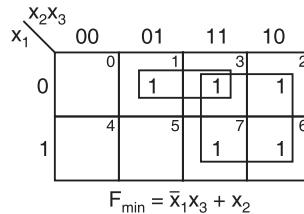


Figure 9.4 Example 9.1: K-map for F.

#### 9.4 CAPABILITIES OF THRESHOLD GATE

The capability of a single threshold gate in realizing various Boolean functions is demonstrated by means of a numerical example in Figure 9.5. Note that Figure 9.5a shows a T gate with 4 inputs A, B, C, D, and F is the function of these input variables. The weights assigned are  $-1, 1, 2$ , and  $4$  as shown in the figure. By altering the value of T, the same gate produces different functions. Figure 9.5b shows the weighted sum figures in each cell. For cell 10 in the map  $A = 1, B = 0, C = 1$  and  $D = 0$ . The weighted sum  $\sum w_i x_i$  is given by  $(-1) \times 1 + 0 \times 1 + 1 \times 2 + 0 \times 4 = 1$ . While writing the weighted sum in each cell, it is convenient and quicker if one remembers the adjacency relationships in the map. For instance, the row labelled 01 is adjacent to the row labelled 00 and note that the only variable which changes is B which goes from 0 to 1. Hence by adding the weight of B to the corresponding entry in the earlier row one gets the values in the row 01. Similarly one gets the entries in the row 11 by simply adding the weight of A. Finally the row labelled 10 is filled by simply subtracting the weight of B from the corresponding entry of row 11 since B is changing from 1 to 0.

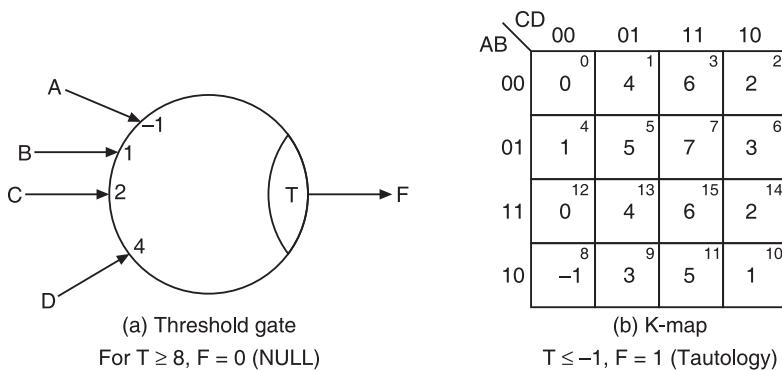


Figure 9.5 Threshold gate and K-map to show weighted sum figures in cells.

Observe that this gate produces the NULL function.  $F = 0$  if  $T \geq 8$  because the weighted sum is less than 8 for every combination of input variables. As the threshold T is gradually decreased to  $7, 6, 5, 4, 3, 2, 1, 0$  the function realized will contain the minterms as indicated. Eventually if T is further decreased to  $(-1)$  or less, the function will be 1 (called tautology), regardless of the input variables.

## 516 FUNDAMENTALS OF DIGITAL CIRCUITS

By merely changing the values of resistance, we can change the value of the threshold and realize different Boolean functions. The weights can also be changed in a similar fashion. It can therefore be concluded that a large number of functions can be realized using a single T gate. This capability has to be attributed to the hybrid nature of the T gate having analog weights and threshold but digital inputs and output. Figure 9.6 illustrates the power of T.

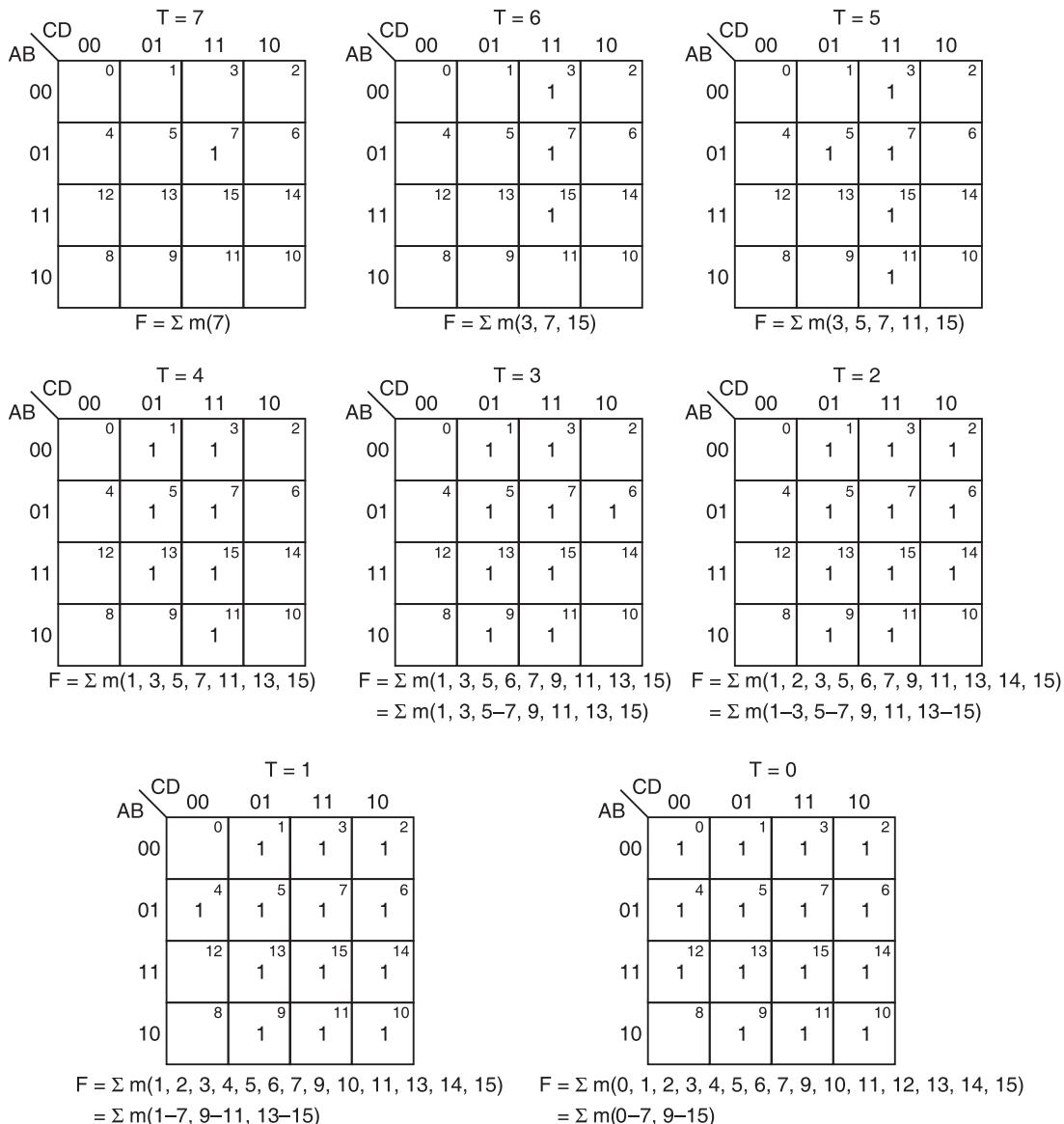


Figure 9.6 K-maps for illustration of the power of T.

**EXAMPLE 9.2** Determine the switching function of the threshold element shown in Figure 9.7.

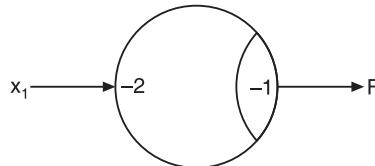


Figure 9.7 Example 9.2: Threshold element.

**Solution**

For the given threshold element

$$w_1 = -2 \text{ and } T = -1$$

The weighted sum

$$w = w_1 x_1 = -2 x_1$$

The input-output relation for this threshold element is shown in Table 9.2.

Table 9.2 Example 9.2: Input-output relation

Input variable	Weighted sum	Output
$x_1$	$-2x_1$	F
0	0	1
1	-2	0

From the table we observe that the output is the complement of the input. Hence the threshold gate realizes the NOT gate. The switching function is NOT:

$$F = \bar{x}_1$$

**EXAMPLE 9.3** Determine the switching function of the threshold element shown in Figure 9.8.

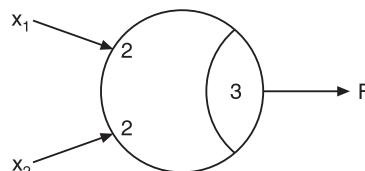


Figure 9.8 Example 9.3: Threshold element.

**Solution**

For the given threshold element

$$w_1 = 2, w_2 = 2, \text{ and } T = 3$$

The weighted sum

$$w = w_1 x_1 + w_2 x_2 = 2x_1 + 2x_2$$

The value of the weighted sum is determined for each of the possible input combinations of  $x_1$  and  $x_2$  and compared with the value of T and the corresponding value of F is obtained as shown in Table 9.3.

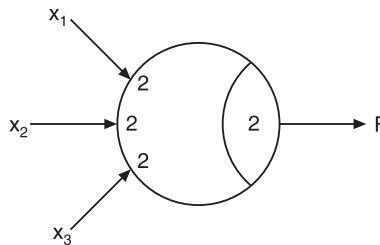
**Table 9.3** Example 9.3: Input-output relation

Input variables		Weighted sum	Output
$x_1$	$x_2$	$w = 2x_1 + 2x_2$	F
0	0	0	0
0	1	2	0
1	0	2	0
1	1	4	1

From Table 9.3, we observe that the input-output relation of this threshold element is the same as the operation of the AND gate. Thus the threshold element realizes the AND operation. The switching function is AND:

$$F = x_1 \cdot x_2$$

**EXAMPLE 9.4** Determine the switching function of the threshold element shown in Figure 9.9.



**Figure 9.9** Example 9.4: Threshold element.

### Solution

For the given threshold element

$$w_1 = 2, \quad w_2 = 2, \quad w_3 = 2, \quad \text{and} \quad T = 2$$

The weighted sum

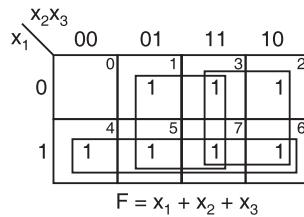
$$w = w_1 x_1 + w_2 x_2 + w_3 x_3 = 2x_1 + 2x_2 + 2x_3$$

The value of the weighted sum is determined for each of the possible input combinations of  $x_1$ ,  $x_2$  and  $x_3$  and compared with the value of T and the corresponding value of F is obtained as shown in Table 9.4.

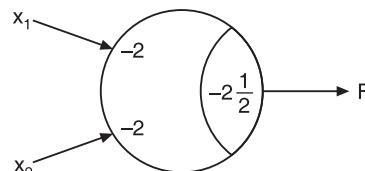
**Table 9.4** Example 9.4: Input-output relation

Input variables			Weighted sum	Output
$x_1$	$x_2$	$x_3$	$w = 2x_1 + 2x_2 + 2x_3$	$F$
0	0	0	0	0
0	0	1	2	1
0	1	0	2	1
0	1	1	4	1
1	0	0	2	1
1	0	1	4	1
1	1	0	4	1
1	1	1	6	1

The table shows that the output is 1 even if one of the inputs is a 1. So the threshold element realizes a 3-input OR gate. Therefore, the switching function performed by this threshold gate is OR. The K-map, its minimization and the minimal expression obtained from it are shown in Figure 9.10.

**Figure 9.10** Example 9.4: K-map.

**EXAMPLE 9.5** Determine the switching function of the threshold element shown in Figure 9.11.

**Figure 9.11** Example 9.5: Threshold element.

### Solution

For the given threshold element

$$w_1 = -2, \quad w_2 = -2, \quad \text{and} \quad T = -2 \frac{1}{2}$$

The weighted sum

$$w = w_1 x_1 + w_2 x_2 = -2x_1 - 2x_2$$

The input-output relation for this threshold element is shown in Table 9.5.

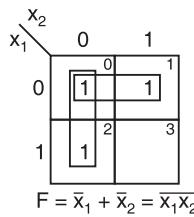
**Table 9.5** Example 9.5: Input-output relation

Input variables		Weighted sum	Output
$x_1$	$x_2$	$w = -2x_1 - 2x_2$	$F$
0	0	0	1
0	1	-2	1
1	0	-2	1
1	1	-4	0

From the table we observe that the output is 0 only when all the inputs are 1. Hence the threshold element realizes a NAND gate. The switching function is NAND:

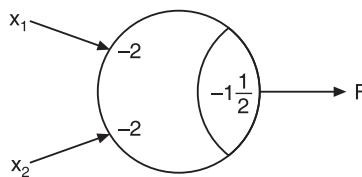
$$F = \overline{x_1 x_2}$$

The K-map for  $F$ , its minimization, and the minimal expression obtained from it are shown in Figure 9.12.



**Figure 9.12** Example 9.5: K-map for  $F$ .

**EXAMPLE 9.6** Determine the switching function of the threshold element shown in Figure 9.13.



**Figure 9.13** Example 9.6: Threshold element.

### Solution

For the given threshold element

$$w_1 = -2, \quad w_2 = -2, \quad \text{and} \quad T = -1 \frac{1}{2}$$

The weighted sum

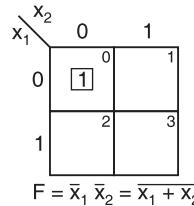
$$w = w_1 x_1 + w_2 x_2 = -2x_1 - 2x_2$$

The input-output relation for this threshold element is shown in Table 9.6.

**Table 9.6** Example 9.6: Input-output relation

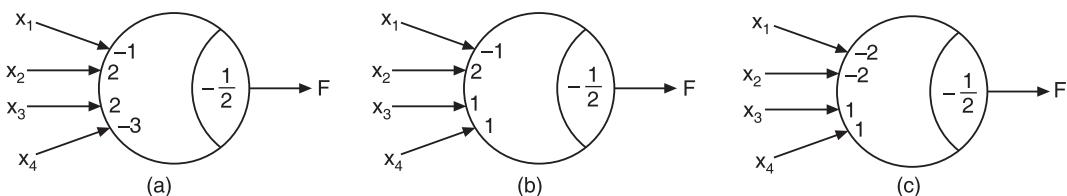
Input variables		Weighted sum	Output
$x_1$	$x_2$	$w = -2x_1 - 2x_2$	$F$
0	0	0	1
0	1	-2	0
1	0	-2	0
1	1	-4	0

From the table we observe that the output is 1 only when all the inputs are 0. Hence the threshold element realizes a NOR gate. The switching function is NOR:  $F = \overline{x_1 + x_2}$ . The K-map for F, its minimization, and the minimal expression obtained from it are shown in Figure 9.14.



**Figure 9.14** Example 9.6: K-map for F.

**EXAMPLE 9.7** Find the function  $F(x_1, x_2, x_3, x_4)$  realized by each of the threshold networks shown in Figures 9.15a, b and c.



**Figure 9.15** Example 9.7: Threshold gates.

### Solution

(a) For the given threshold element of Figure 9.15a

$$w_1 = -1, \quad w_2 = 2, \quad w_3 = 2, \quad w_4 = -3, \quad \text{and} \quad T = -\frac{1}{2}$$

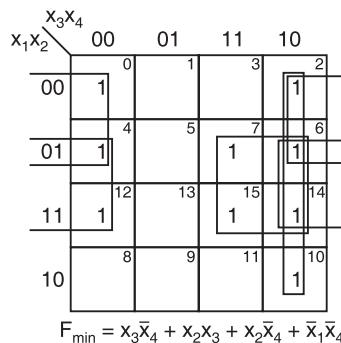
The weighted sum

$$w = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 = -x_1 + 2x_2 + 2x_3 - 3x_4$$

The input-output relation for this threshold element is shown in Table 9.7a.

**Table 9.7a** Example 9.7a: Input-output relation

Input variables				Weighted sum	Output
$x_1$	$x_2$	$x_3$	$x_4$	$w = -x_1 + 2x_2 + 2x_3 - 3x_4$	F
0	0	0	0	0	1
0	0	0	1	-3	0
0	0	1	0	2	1
0	0	1	1	-1	0
0	1	0	0	2	1
0	1	0	1	-1	0
0	1	1	0	4	1
0	1	1	1	1	1
1	0	0	0	-1	0
1	0	0	1	-4	0
1	0	1	0	1	1
1	0	1	1	-2	0
1	1	0	0	1	1
1	1	0	1	-2	0
1	1	1	0	3	1
1	1	1	1	0	1



**Figure 9.16** Example 9.7a: K-map.

From the table we observe that the Boolean expression for F is  $F = \sum m(0, 2, 4, 6, 7, 10, 12, 14, 15)$ .

Drawing a 4-variable K-map for F and minimizing it as shown in Figure 9.16 the minimal expression for F is

$$F_{\min} = x_3\bar{x}_4 + x_2x_3 + x_2\bar{x}_4 + \bar{x}_1\bar{x}_4$$

(b) For the given threshold element of Figure 9.15b

$$w_1 = -1, \quad w_2 = 2, \quad w_3 = 1, \quad w_4 = 1 \quad \text{and} \quad T = -\frac{1}{2}$$

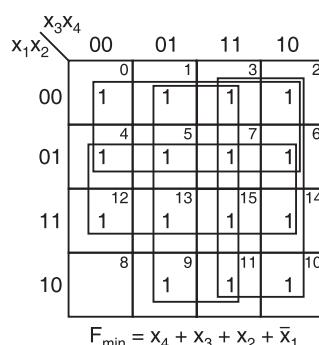
The weighted sum

$$w = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 = -x_1 + 2x_2 + x_3 + x_4$$

The input-output relation for this threshold element is shown in Table 9.7b.

**Table 9.7b** Example 9.7b: Input-output relation

Input variables				Weighted sum	Output
$x_1$	$x_2$	$x_3$	$x_4$	$w = -x_1 + 2x_2 + x_3 + x_4$	F
0	0	0	0	0	1
0	0	0	1	1	1
0	0	1	0	1	1
0	0	1	1	2	1
0	1	0	0	2	1
0	1	0	1	3	1
0	1	1	0	3	1
0	1	1	1	4	1
1	0	0	0	-1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	2	1
1	1	1	0	2	1
1	1	1	1	3	1



**Figure 9.17** Example 9.7b: K-map.

From the table we observe that the Boolean expression for F is

$$F = \Sigma m(0-7, 9-15)$$

Drawing a 4-variable K-map for F and minimizing it as shown in Figure 9.17 the minimal expression for F is

$$F_{\min} = x_4 + x_3 + x_2 + \bar{x}_1$$

(c) For the given threshold element of Figure 9.15c

$$w_1 = -2, \quad w_2 = -2, \quad w_3 = 1, \quad w_4 = 1, \quad \text{and} \quad T = -\frac{1}{2}$$

The weighted sum

$$w = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 = -2x_1 - 2x_2 + x_3 + x_4$$

The input-output relation for this threshold element is shown in Table 9.7c.

From the table we observe that the Boolean expression for F is

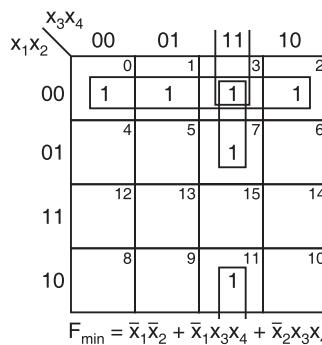
$$F = \Sigma m(0, 1, 2, 3, 7, 11)$$

Drawing a K-map for F and minimizing it as shown in Figure 9.18 the minimal expression for F is

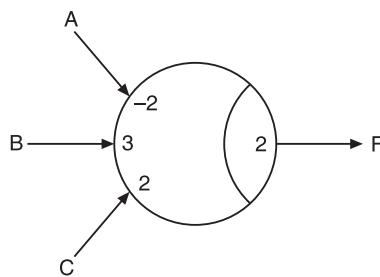
$$F_{\min} = \bar{x}_1\bar{x}_2 + \bar{x}_1x_3x_4 + \bar{x}_2x_3x_4$$

**Table 9.7c** Example 9.7c: Input-output relation

Input variables				Weighted sum	Output
$x_1$	$x_2$	$x_3$	$x_4$	$w = -2x_1 - 2x_2 + x_3 + x_4$	F
0	0	0	0	0	1
0	0	0	1	1	1
0	0	1	0	1	1
0	0	1	1	2	1
0	1	0	0	-2	0
0	1	0	1	-1	0
0	1	1	0	-1	0
0	1	1	1	0	1
1	0	0	0	-2	0
1	0	0	1	-1	0
1	0	1	0	-1	0
1	0	1	1	0	1
1	1	0	0	-4	0
1	1	0	1	-3	0
1	1	1	0	-3	0
1	1	1	1	-2	0

**Figure 9.18** Example 9.7c: K-map.

**EXAMPLE 9.8** Obtain the logic expression for the threshold element shown in Figure 9.19 and determine its equivalent gate circuit.

**Figure 9.19** Example 9.8: Threshold element.

### Solution

For the given threshold element

$$w_1 = -2, \quad w_2 = 3, \quad w_3 = 2, \quad \text{and} \quad T = 2$$

The weighted sum

$$w = w_1 A + w_2 B + w_3 C = -2A + 3B + 2C$$

The input-output relation for this threshold element is shown in Table 9.8.

**Table 9.8** Example 9.8: Input-output relation

Input variables			Weighted sum	Output
A	B	C	$-2A + 3B + 2C$	F
0	0	0	0	0
0	0	1	2	1
0	1	0	3	1
0	1	1	5	1
1	0	0	-2	0
1	0	1	0	0
1	1	0	1	0
1	1	1	3	1

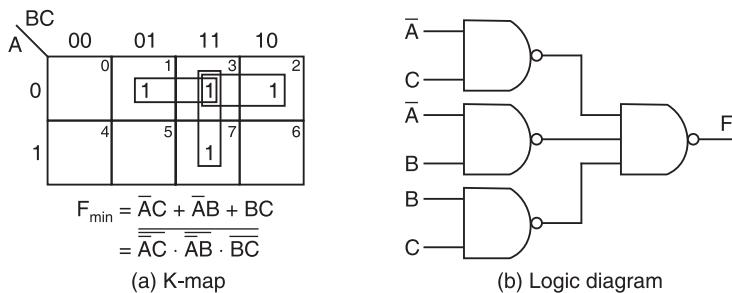


Figure 9.20 Example 9.8.

From the table we observe that the Boolean expression for  $F$  is

$$F = \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}\overline{C} + \overline{A}BC + ABC = \Sigma m(1, 2, 3, 7)$$

Drawing a K-map for  $F$  and minimizing it as shown in Figure 9.20a, the minimal expression for  $F$  is

$$F_{\min} = \overline{A}\overline{C} + \overline{A}\overline{B} + BC$$

## 9.5 UNIVERSALITY OF A T-GATE

We have seen that a threshold gate is more useful than conventional logic gates. A single T-gate can realize a large number of functions by merely changing either the weights or the threshold or both, which can be done by altering the value of the corresponding resistors. Since a threshold gate can realize universal gates, i.e. NAND gates and NOR gates, a threshold gate is also a universal gate. A single threshold gate cannot realize all the functions. For example, an X-OR function cannot be realized by a single T-gate. Realization of logic gates using T gates is illustrated in Figure 9.21.

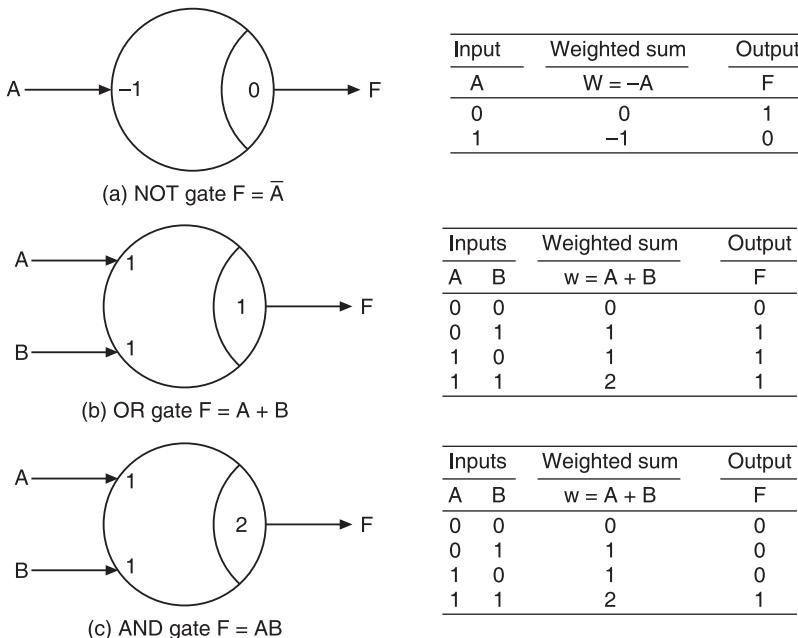
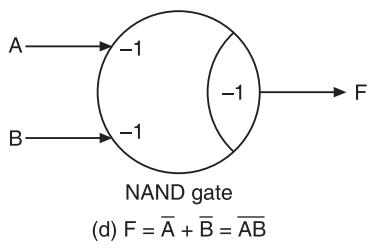
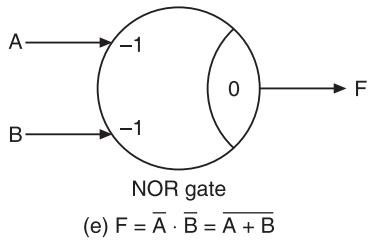


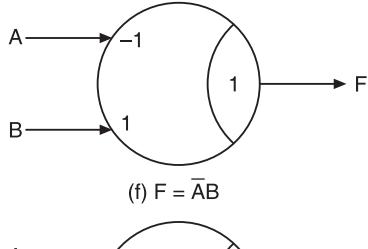
Figure 9.21 Realization of logic gates using T-gates (Contd.).



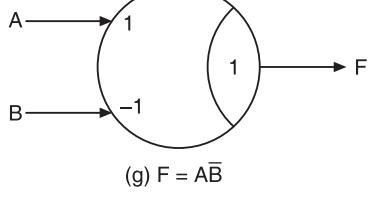
Inputs		Weighted sum	Output
A	B	$w = -A - B$	F
0	0	0	1
0	1	-1	1
1	0	-1	1
1	1	-2	0



Inputs		Weighted sum	Output
A	B	$w = -A - B$	F
0	0	0	1
0	1	-1	0
1	0	-1	0
1	1	-2	0



Inputs		Weighted sum	Output
A	B	$w = -A + B$	F
0	0	0	0
0	1	1	1
1	0	-1	0
1	1	0	0



Inputs		Weighted sum	Output
A	B	$w = A - B$	F
0	0	0	0
0	1	-1	0
1	0	1	1
1	1	0	0

Figure 9.21 Realization of logic gates using T-gates.

**Implementation of X-OR gate:** An X-OR gate cannot be implemented by using single threshold gate. We know that the logic expression for an X-OR gate is

$$F(A, B) = \overline{AB} + A\overline{B}$$

The output of the X-OR gate is logic 1 when the input combinations are  $\overline{AB}$  and  $A\overline{B}$  and it is logic 0 when the input combinations are  $A\overline{B}$  and  $AB$ .

Thus, we can write

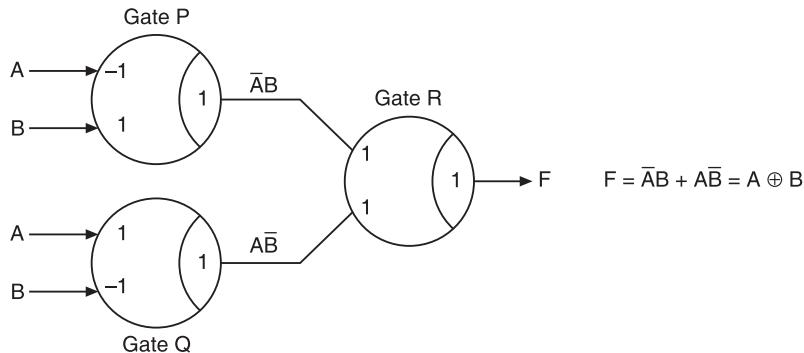
$$\begin{aligned} (\overline{AB}) & \quad -w_1 + w_2 \geq T \\ (A\overline{B}) & \quad w_1 - w_2 \geq T \end{aligned} \quad \Rightarrow \quad (-w_1 + w_2 + w_1 - w_2) \geq 2T, \text{ i.e. } 0 \geq T$$

$$\begin{aligned} (\overline{AB}) & \quad -w_1 - w_2 < T \\ (AB) & \quad w_1 + w_2 < T \end{aligned} \quad \Rightarrow \quad (-w_1 - w_2 + w_1 + w_2) < 2T, \text{ i.e. } 0 < T$$

The above two equations are contradictory; T cannot be greater than or equal to and less than 0 simultaneously. Therefore, we can say that the function  $F(A, B) = \overline{AB} + A\overline{B}$  cannot be realized using only one threshold gate.

This also shows that every Boolean function cannot be realized by only one threshold gate. The Boolean function which can be realized by a single threshold gate is called a threshold function.

**Implementation of X-OR gate using three threshold gates:** Figure 9.22 shows the implementation of X-OR gate using three threshold gates.

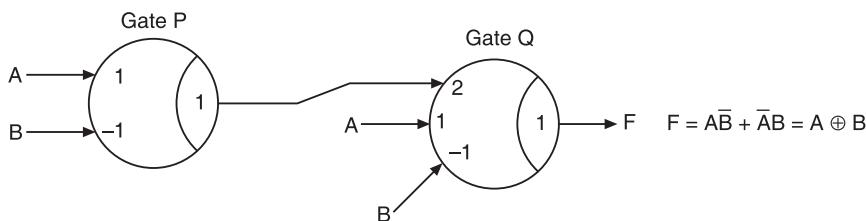


Inputs		Weighted sum of gate P	Weighted sum of gate Q	Inputs of gate R		Weighted sum of gate R	Output of gate R
A	B	$-A + B$	$A - B$	Output of P	Output of Q	$\bar{A}\bar{B} + A\bar{B}$	F
0	0	0	0	0	0	$0 + 0 = 0$	0
0	1	1	-1	1	0	$1 + 0 = 1$	1
1	0	-1	1	0	1	$0 + 1 = 1$	1
1	1	0	0	0	0	$0 + 0 = 0$	0

Figure 9.22 Implementation of X-OR gate using three threshold gates.

X-OR gate can also be implemented by using three threshold gates by interchanging the weights of A and B inputs of the gates P and Q as shown in Figure 9.22.

**Implementation of X-OR gate using two threshold gates:** Figure 9.23 shows the implementation of X-OR gate using two threshold gates.



Inputs		Weighted sum of gate P	Inputs of gate Q			Weighted sum of gate Q	Output of Q
A	B	$-A + B$	Output of P	A	B	$2\bar{A}\bar{B} + A - B$	F
0	0	0	0	0	0	$0 + 0 - 0 = 0$	0
0	1	1	1	0	1	$2 + 0 - 1 = 1$	1
1	0	-1	0	1	0	$0 + 1 - 0 = 1$	1
1	1	0	0	1	1	$0 + 1 - 1 = 0$	0

Figure 9.23 Implementation of X-OR gate using two threshold gates.

**Implementation of X-NOR gate:** Like X-OR gate, X-NOR gate also cannot be implemented using single threshold gate. Figure 9.24 shows the implementation of X-NOR gate using two threshold gates.

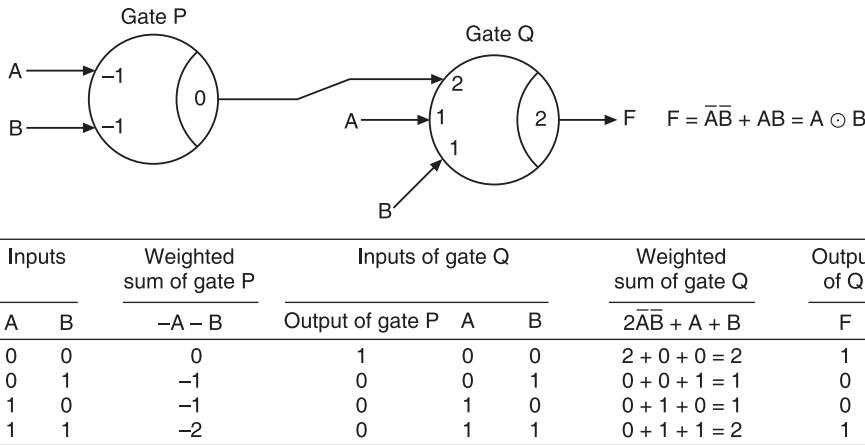


Figure 9.24 Implementation of X-NOR gate using two threshold gates.

Notice that in all the realizations shown above, the variables associated with positive weights appear in uncomplemented form and the variables with negative weights appear in complemented form in the output function.

## 9.6 IMPLEMENTATION OF BOOLEAN FUNCTION USING THRESHOLD GATE

The following procedure is used to implement the Boolean function using the threshold gate.

1. Write the truth table such that column 1 specifies the combination number, column 2 specifies the inputs, and column 3 specifies the function output (either 0 or 1).
2. Add column 4 specifying weighted sums and their relations with the threshold. For input combinations where  $F = 1$ , write the weighted sum to be equal to or greater than the threshold ( $T$ ). For input combinations where  $F = 0$ , write the weighted sum to be less than the threshold ( $T$ ).
3. Determine the values for weights and threshold that satisfy all the relations specified in column 4.
4. If solution exists, the Boolean function is a threshold function; otherwise the Boolean function is not a threshold function.

**EXAMPLE 9.9** Is it possible to realize the logic function  $F(A, B, C) = \Sigma m(1, 2, 4, 7)$  using a single threshold gate?

### Solution

Table 9.9 gives the truth table and the inequalities to be satisfied for  $F$  to be implemented using a single threshold element.  $w_1, w_2$  and  $w_3$  are the weights associated with the input variables  $A, B$  and  $C$  respectively. The weighted sum  $w = w_1A + w_2B + w_3C$ .

**Table 9.9** Example 9.9: Input-output relation

Combination number	Inputs			Output F	Inequality
	A	B	C		
0	0	0	0	0	$0 < T$
1	0	0	1	1	$w_3 \geq T$
2	0	1	0	1	$w_2 \geq T$
3	0	1	1	0	$w_2 + w_3 < T$
4	1	0	0	1	$w_1 \geq T$
5	1	0	1	0	$w_1 + w_3 < T$
6	1	1	0	0	$w_1 + w_2 < T$
7	1	1	1	1	$w_1 + w_2 + w_3 \geq T$

From the table we observe the following:

1. The first condition implies  $T$  must be positive.
2. The second and third conditions require both  $w_2$  and  $w_3$  to be greater than  $T$ .
3. The fourth condition is  $w_2 + w_3 < T$  which contradicts the inequalities 2 and 3.
4. The fifth condition requires  $w_1$  to be greater than  $T$ .
5. The sixth and seventh conditions require  $w_1 + w_3$  and  $w_1 + w_2$  to be less than  $T$  which is again contradictory to the condition that  $w_1$ ,  $w_2$  and  $w_3$  must be greater than  $T$ .

Therefore, the inequalities are not satisfied and hence this function is not a threshold function and so cannot be realized by a single threshold element.

**EXAMPLE 9.10** Test whether the following Boolean expression can be realized using a single threshold gate or not

$$F(A, B, C) = \Sigma m(0, 1, 4, 5, 7)$$

#### Solution

Table 9.10 gives the truth table and the inequalities to be satisfied for the given function  $F$  to be implemented using a single threshold element.

**Table 9.10** Example 9.10: Input-output relation

Combination number	Inputs			Output F	Relation between $w$ and $T$
	A	B	C		
0	0	0	0	1	$0 \geq T$
1	0	0	1	1	$w_3 \geq T$
2	0	1	0	0	$w_2 < T$
3	0	1	1	0	$w_2 + w_3 < T$
4	1	0	0	1	$w_1 \geq T$
5	1	0	1	1	$w_1 + w_3 \geq T$
6	1	1	0	0	$w_1 + w_2 < T$
7	1	1	1	1	$w_1 + w_2 + w_3 > T$

In the above table from row 0, we observe that  $0 \geq T$ , that means  $T$  must be zero or negative. From row 2 we observe that  $w_2 < T$ , that means  $w_2$  must be negative. From row 4 we observe that  $w_1 \geq T$ . From rows 5 and 6 we observe that  $w_3 > w_2$ . Take  $w_1 > w_3$ . Thus, we are able to establish the relations between weights and threshold as

$$w_1 > w_3 > T > w_2$$

For simplicity assigning integer values to weights and  $T$ , we get

$$w_1 = 2, w_3 = 1, T = -\frac{1}{2}, \text{ and } w_2 = -3$$

Here we get the solutions which satisfy all the relations in column 4. This result is verified in Table 9.11.

**Table 9.11** Example 9.10: Truth table for verification of result

Inputs			Weighted sum	Output
A	B	C	$w = 2A - 3B + C$	F
0	0	0	0	1
0	0	1	1	1
0	1	0	-3	0
0	1	1	-2	0
1	0	0	2	1
1	0	1	3	1
1	1	0	-1	0
1	1	1	0	1

Hence the given function can be realized using a single threshold gate.

**Limitations of threshold gates:** The limitations of threshold gates are as follows:

1. A threshold gate is very sensitive to parameter variations.
2. It is difficult to fabricate it in IC form.
3. The speed of switching of threshold elements is much lower than that of conventional gates.

## 9.7 UNATE FUNCTIONS

A function  $f(x_1, x_2, \dots, x_n)$  of binary variables  $x_1, x_2, \dots, x_n$  is said to be positive in a variable  $x_i$  if there exists a disjunctive (minimal sum of product) or conjunctive (minimal product of sum) expression for the function  $F$  in which  $x_i$  appears only in uncomplemented form. For example, the function  $f = x \bar{y}z + \bar{x}yz$  is positive in variable  $z$ , because  $z$  appears only in uncomplemented form in this expression.

Similarly, a function  $f(x_1, x_2, \dots, x_n)$  of binary variables  $x_1, x_2, \dots, x_n$  is said to be negative in a variable  $x_i$  if there exists a disjunctive (minimal sum of product) or conjunctive (minimal product of sum) expression for the function  $F$  in which  $x_i$  appears only in complemented form. For example,

the function  $f = x_1x_2\bar{x}_3 + \bar{x}_1\bar{x}_3$  is negative in variable  $x_3$ , since  $x_3$  appears only in complemented form in this expression.

If a function  $f(x_1, x_2, \dots, x_n)$  is either only positive or only negative in  $x_i$ , then it is said to be *unate* in  $x_i$ . Further, if it is unate in each one of its variables, then it is called a *unate function*. Thus, if a function can be represented by a disjunctive or conjunctive expression in which no variable appears in both its complemented and uncomplemented forms, then it is a unate function.

**EXAMPLE 9.11** Determine whether the function  $f = DC + BC + AB$  is a unate function.

**Solution**

Since no variable in this function is appearing both in complemented and uncomplemented forms, it is a unate function. Since all the variables in this function appear in uncomplemented form only, it is a positive function.

**EXAMPLE 9.12** Determine whether the function  $f = \bar{D}C + \bar{A}B + \bar{A}C$  is a unate function.

**Solution**

In this function each variable appears either only in complemented or only in uncomplemented form. Therefore, the function is unate in all of its variables and hence is a unate function, but it is neither a positive nor a negative function.

**EXAMPLE 9.13** Determine whether the function  $f = \bar{x}_1\bar{x}_2x_3 + \bar{x}_2\bar{x}_3x_4$  is a unate function.

**Solution**

In the given function  $f$ , the variables  $x_1$ ,  $x_2$  and  $x_4$  appear as  $\bar{x}_1$ ,  $\bar{x}_2$ , and  $x_4$  respectively, but the variable  $x_3$  appears as  $x_3$  in the first term and as  $\bar{x}_3$  in the second term. Therefore,  $f$  is unate in the variables  $x_1$ ,  $x_2$  and  $x_4$  and is not unate in the variable  $x_3$ . Hence the function  $f$  is not a unate function.

**EXAMPLE 9.14** Determine whether the following functions are unate functions:

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

$$F_2(A, B, C, D) = AB + AD + BC + AC$$

Comment on the results.

**Solution**

The function  $F_1$  has the variables  $A$ ,  $B$ ,  $C$ , and  $D$  as  $\bar{A}$ ,  $B$ ,  $\bar{C}$ , and  $D$  respectively. This shows that the function  $F_1$  is positive in  $B$  and  $D$  and negative in  $A$  and  $C$ . It is neither positive nor negative for all the variables but is a unate function.

The function  $F_2$  has all the variables present in uncomplemented form, therefore, it is positive in all its variables and is a unate function.

Comparing  $F_1$  and  $F_2$ , we observe that in  $F_1$  if  $\bar{A}$  is replaced by  $A$  and  $\bar{C}$  is replaced by  $C$ , the resulting function will be a positive function. Therefore, we can conclude that a unate function can be converted into a positive unate function or negative unate function by relabelling complemented variables as uncomplemented variables and uncomplemented variables as complemented variables respectively. For reconverting the latter function, the original function can be determined. It is important to note that every threshold function is unate.

## 9.8 SYNTHESIS OF THRESHOLD FUNCTION

To synthesize, that is, to realize the Boolean expressions, the following steps are to be carried out.

*Step 1. Check for unateness of the function*

Draw the K-map for the given function, minimize it, and write the minimal expression and check for the unateness of that minimal expression.

*Step 2. Convert the given unate function into positive function*

If the function is unate, test whether it is positive in all its variables or not. If not, convert it to another function that is positive in all its variables by replacing each complemented variable by its uncomplemented variable.

*Step 3. Determine the true and false prime implicants of the positive function*

To determine the true and false prime implicants of the positive function, draw the K-maps for it in SOP and POS forms, minimize them and obtain the minimal expressions from them.

*Step 4. Obtain the inequalities*

Obtain the inequalities such that the minimal weighted sum corresponding to each true prime implicant is greater than the maximal weighted sum of any false prime implicant.

The minimal weighted sum of a true prime implicant can be obtained by considering the weights of only those variables which appear in the prime implicant. The values of other variables are taken as 0. For a false prime implicant, the values of variables appearing in the prime implicant are 0s. Therefore, the maximal weighted sum of a false prime implicant is calculated by considering the weights of variables which are not present in the prime implicant.

*Step 5. Obtain the weight constraints*

From the inequalities obtain the weight constraints.

*Step 6. Assign the values to the weights*

Assign proper values to the weights such that all the inequalities are satisfied.

*Step 7. Assign the value for the threshold*

Assign the value for the threshold such that the threshold value is in between the minimal weighted sum of true prime implicants and the maximal weighted sum of false prime implicants.

*Step 8. Obtain the weights and threshold for the original function*

Obtain the weights of the original function by putting a negative sign to the weights of the variables which were in complemented form in the original function. Calculate the threshold for the original function by subtracting the weights of the complemented variables in positive function from the threshold for the positive function.

*Step 9. Realize the function using threshold gate*

Based on the weights and threshold of the original function realize it using a threshold gate.

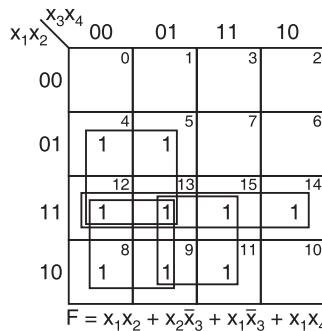
**EXAMPLE 9.15** Realize the following Boolean function using T-gate:

$$F(x_1, x_2, x_3, x_4) = \Sigma m(4, 5, 8, 9, 11, 12, 13, 14, 15)$$

**Solution**

*Step 1. Check for unateness of the function*

To check for unateness of the function first draw the K-map for the given function F, minimize it, and obtain the minimal expression for F as shown in Figure 9.25.

Figure 9.25 Example 9.15: K-map for  $F$ .

The minimal expression for  $F$  ( $F_{\min} = x_1x_2 + x_2\bar{x}_3 + x_1\bar{x}_3 + x_1x_4$ ) shows that all the variables in the function are unate and hence the given function is a unate function.

*Step 2. Convert the given unate function into a positive function*

To convert the given unate function into a positive function ( $F_P$ ) replace each complemented variable by its uncomplemented form.

Therefore,

$$F_P = x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3$$

*Step 3. Determine the true and false prime implicants of the function  $F_P$*

To determine the true and false prime implicants of  $F_P$  draw the K-maps in SOP and POS forms, minimize them, and obtain the minimal expressions from them as shown in Figure 9.26. Observe that the true prime implicants are the same as those of  $F_P$ .

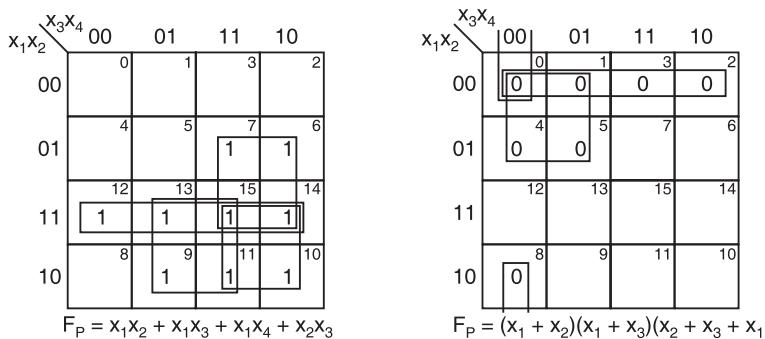


Figure 9.26 Example 9.15: K-maps to determine the true and false prime implicants.

*Step 4. Obtain the inequalities*

The inequalities are to be obtained such that the minimal weighted sum corresponding to each true prime implicant is greater than the maximal weighted sum of any false prime implicant. By applying the above criterion, we have the following inequalities

$$\left. \begin{array}{l} w_1 + w_2 \\ w_1 + w_3 \\ w_1 + w_4 \\ w_2 + w_3 \end{array} \right\} > \left. \begin{array}{l} w_2 + w_4 \\ w_3 + w_4 \\ w_1 \end{array} \right\}$$

where  $w_1, w_2, w_3$  and  $w_4$  are the weights of  $x_1, x_2, x_3$  and  $x_4$  respectively.

*Step 5. Obtain the weight constraints*

The weight constraints are to be obtained from the inequalities. Observing the inequalities of step 4, the following weight constraints can be obtained:

$$\begin{aligned} w_2 > 0 & \text{ since } w_1 + w_2 > w_1 \\ w_3 > 0 & \text{ since } w_1 + w_3 > w_1 \\ w_4 > 0 & \text{ since } w_1 + w_4 > w_1 \\ w_3 > w_4 & \text{ since } w_2 + w_3 > w_2 + w_4 \\ w_2 > w_4 & \text{ since } w_2 + w_3 > w_3 + w_4 \\ w_1 > w_4 & \text{ since } w_1 + w_3 > w_3 + w_4 \\ w_1 > w_3 & \text{ since } w_1 + w_4 > w_3 + w_4 \\ w_1 > w_2 & \text{ since } w_1 + w_4 > w_2 + w_4 \end{aligned}$$

Therefore, we have  $w_1 > w_2$  and  $w_3 > w_4$ . There is no clear relation between  $w_2$  and  $w_3$ .

*Step 6. Assign the values to the weights to satisfy the inequalities*

The weights are to be assigned proper values such that all the inequalities are satisfied.

Let us assign  $w_4 = 1$ ,  $w_2 = w_3 = 2$  and  $w_1 = 3$ . Since  $w_2 + w_3 > w_1$ ,  $w_1$  cannot be assigned 4 or more. Therefore, we have

$$\left. \begin{array}{c} 3+2 \\ 3+2 \\ 3+1 \\ 2+2 \end{array} \right\} > \left\{ \begin{array}{c} 2+1 \\ 2+1 \\ 3 \end{array} \right\}$$

*Step 7. Assign the value for the threshold*

The threshold value  $T$  is to be assigned such that it is in between the minimal weighted sum of true prime implicants and maximal weighted sum of false prime implicants.

As seen above, the minimal weighted sum of true prime implicant is 4 and the maximal weighted sum of false prime implicant is 3. The threshold should be in between these two values. Therefore, assign the threshold as 3.5, i.e. 7/2.

*Step 8. Obtain the weights and threshold for the original function*

The weights of the original function are obtained by putting a negative sign to the weights of the variables which were in complemented form in the original function.

The threshold for the original function can be calculated by subtracting the weights of the complemented variables in positive function from the threshold for the positive function.

In the original function,  $x_3$  is in complemented form. Therefore, its weight must be complemented, i.e.  $w_3 = -2$ . Therefore, we have

$$w_1 = 3, w_2 = 2, w_3 = -2 \text{ and } w_4 = 1$$

Threshold of positive function  $T_P = 7/2$ , weight of complemented variables  $w_3 = 2$ .

Therefore,

$$T = T_P - w_3 = 7/2 - 2 = 3/2$$

*Step 9. Realize the function using threshold gate*

The function is to be realized using the weights and threshold of the original function. The threshold gate shown in Figure 9.27 satisfies the given Boolean expression. This can be verified.

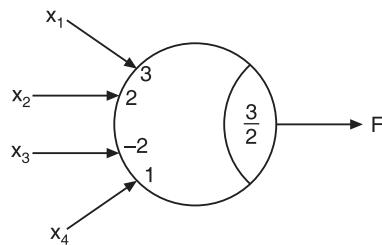


Figure 9.27 Example 9.15: Threshold gate.

## 9.9 MULTI-GATE SYNTHESIS

In the last section we have discussed how to realize a threshold function using a single threshold gate. We know that non-threshold functions cannot be realized using a single threshold gate. Such functions can be realized using two or more threshold gates. For the synthesis of non-threshold function, such a function is decomposed into two or more factors such that each one of them is a threshold function. The decomposed threshold functions are individually realized by threshold gates and then these gates are cascaded to get the realization of the original threshold function.

The non-threshold functions can be decomposed with the help of what are called admissible patterns.

An admissible pattern is a pattern of 1 cells which can be realized by a single threshold gate. Figure 9.28 shows the admissible patterns for functions having three variables. Each admissible pattern may be anywhere on the map, provided that its basic topological structure is preserved.

We can use admissible patterns for three or more variable functions. It is important to note that since the complement of a threshold function is also a threshold function, the patterns formed by 0 cells are also admissible.

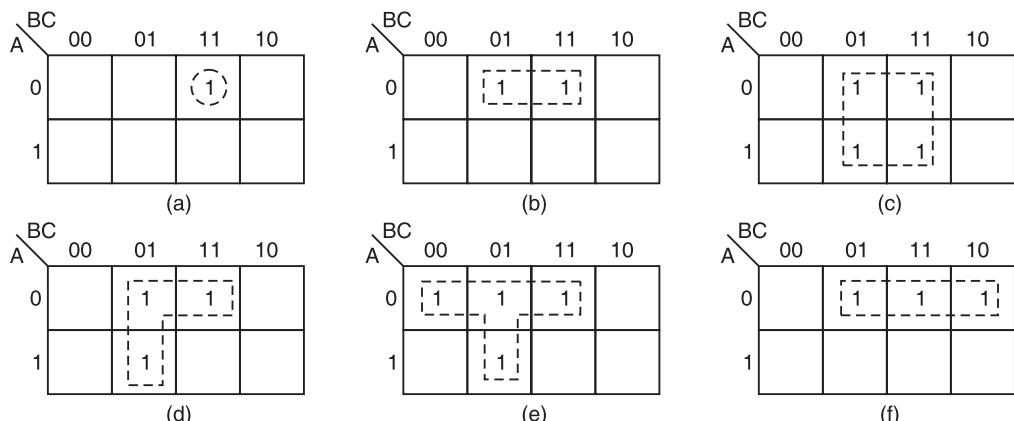


Figure 9.28 Admissible patterns for function having three variables.

**EXAMPLE 9.16** Realize the Boolean function

$$F(x_1, x_2, x_3, x_4) = \sum m(0, 1, 4, 5, 8, 9, 11, 13)$$

**Solution**

*Step 1. Check for unateness of the function*

Draw the K-map for F, minimize it and obtain the minimal expression for F as shown in Figure 9.29.

$$F_{\min} = x_1 \bar{x}_2 x_4 + \bar{x}_2 \bar{x}_3 + \bar{x}_1 \bar{x}_3 + \bar{x}_3 x_4$$

Since the variable  $x_1$  is not unate in F, the given function is not a unate function.

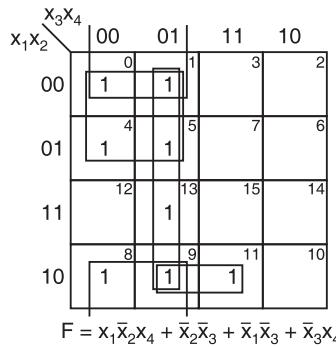


Figure 9.29 Example 9.16: K-map for F.

*Step 2. Decompose the non-unate function into unate functions*

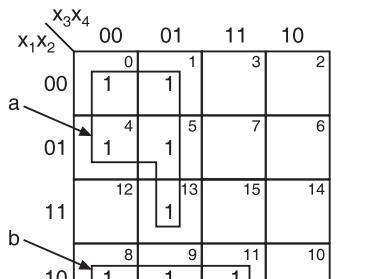
The given non-unate function F is decomposed into two unate functions  $F_a$  and  $F_b$  using admissible patterns as shown in Figure 9.30.

$$F_a = \Sigma m(0, 1, 4, 5, 13)$$

$$= \bar{x}_1 \bar{x}_3 + x_2 \bar{x}_3 x_4$$

$$F_b = \Sigma m(8, 9, 11)$$

$$= x_1 \bar{x}_2 \bar{x}_3 + x_1 \bar{x}_2 x_4$$



$$F_a = \Sigma m(0, 1, 4, 5, 13) = \bar{x}_1 \bar{x}_3 + x_2 \bar{x}_3 x_4$$

$$F_b = \Sigma m(8, 9, 11) = x_1 \bar{x}_2 \bar{x}_3 + x_1 \bar{x}_2 x_4$$

Figure 9.30 Example 9.16: Decomposition of non unate function into two unate functions.

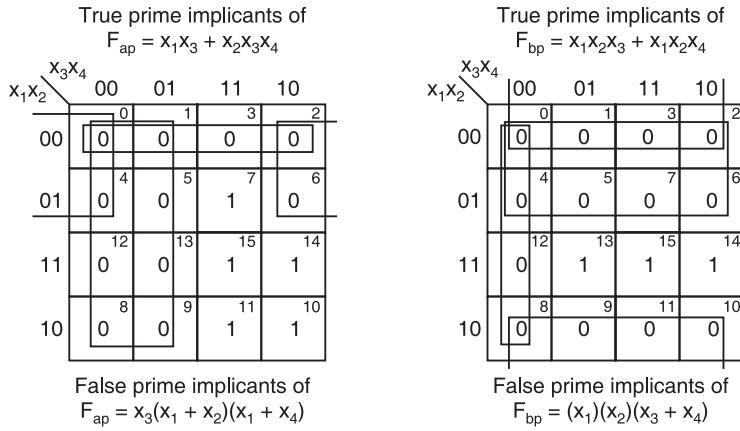
*Step 3. Convert the unate functions  $F_a$  and  $F_b$  into positive functions*

The functions  $F_a$  and  $F_b$  are converted into positive functions  $F_{ap}$  and  $F_{bp}$  by replacing the complemented variables by their uncomplemented form:

$$F_{ap} = x_1 x_3 + x_2 x_3 x_4 \text{ and } F_{bp} = x_1 x_2 x_3 + x_1 x_2 x_4$$

**Step 4.** Determine the true and false prime implicants

The true prime implicants of  $F_{ap}$  and  $F_{bp}$  are available from step 3. To obtain the false prime implicants draw the K-maps of  $F_{ap}$  and  $F_{bp}$ , minimize them in POS form and write the minimal expressions for them as shown in Figure 9.31.



**Figure 9.31** Example 9.16: K-maps to obtain the false prime implicants.

**Step 5.** Obtain the inequalities for functions  $F_{ap}$  and  $F_{bp}$ 

The inequalities for the functions  $F_{ap}$  and  $F_{bp}$  are obtained as given below.

Inequalities for function $F_{ap}$	Inequalities for function $F_{bp}$
$\left. \begin{array}{l} w_1 + w_3 \\ w_2 + w_3 + w_4 \end{array} \right\} > \left. \begin{array}{l} w_1 + w_2 + w_4 \\ w_3 + w_4 \\ w_2 + w_3 \end{array} \right\}$	$\left. \begin{array}{l} w_1 + w_2 + w_3 \\ w_1 + w_2 + w_4 \end{array} \right\} > \left. \begin{array}{l} w_2 + w_3 + w_4 \\ w_1 + w_3 + w_4 \\ w_1 + w_2 \end{array} \right\}$

**Step 6.** Obtain the weight constraints for functions  $F_{ap}$  and  $F_{bp}$ 

The weight constraints for the functions  $F_{ap}$  and  $F_{bp}$  are obtained as follows:

Weight constraints for function $F_{ap}$	Weight constraints for function $F_{bp}$
$w_1 > w_4$ since $w_1 + w_3 > w_3 + w_4$	$w_1 > w_4$ since $w_1 + w_2 + w_3 > w_2 + w_3 + w_4$
$w_1 > w_2$ since $w_1 + w_3 > w_2 + w_3$	$w_2 > w_4$ since $w_1 + w_2 + w_3 > w_1 + w_3 + w_4$
$w_3 > w_1$ since $w_2 + w_3 + w_4 > w_1 + w_2 + w_4$	$w_3 > 0$ since $w_1 + w_2 + w_3 > w_1 + w_2$
$w_4 > 0$ since $w_2 + w_3 + w_4 > w_2 + w_3$	$w_1 > w_3$ since $w_1 + w_2 + w_4 > w_2 + w_3 + w_4$
$w_2 > 0$ since $w_2 + w_3 + w_4 > w_3 + w_4$	$w_2 > w_3$ since $w_1 + w_2 + w_4 > w_1 + w_3 + w_4$
	$w_4 > 0$ since $w_1 + w_2 + w_4 > w_1 + w_2$

There is no clear relation between  $w_2$  and  $w_4$ . So we choose  $w_2 = w_4$ . Therefore, we have  $w_3 > w_1 > w_2 = w_4 > 0$

There is no clear relation between  $w_1$  and  $w_2$  and between  $w_3$  and  $w_4$ . So we choose  $w_1 = w_2$  and  $w_3 = w_4$ . Therefore, we have  $w_1 = w_2 > w_3 = w_4$

**Step 7. Assign the values to the weights to satisfy the inequalities**

Based on the weight constraints, the values to the weights of  $F_{ap}$  and  $F_{bp}$  are assigned as follows:

We assign $w_2 = w_4 = 1$	We assign $w_3 = w_4 = 1$
Therefore, assign $w_1 = 2$ and $w_3 = 3$ .	Therefore, assign $w_1 = w_2 = 2$ .
Therefore, we have	Therefore, we have
$\begin{matrix} 2+3 \\ 1+3+1 \end{matrix} > \begin{matrix} 2+1+1 \\ 3+1 \\ 1+3 \end{matrix}$	$\begin{matrix} 2+2+1 \\ 2+2+1 \end{matrix} > \begin{matrix} 2+1+1 \\ 2+2 \end{matrix}$

**Step 8. Assign the values for thresholds**

The values for the thresholds of  $F_{ap}$  and  $F_{bp}$  are assigned as follows:

For $F_{ap}$	For $F_{bp}$
Minimal weighted sum of true prime implicants is 5. Maximal weighted sum of false prime implicants is 4. Therefore, $T_{ap} = (5 + 4)/2 = 9/2 = 4.5$	Minimal weighted sum of true prime implicants is 5. Maximal weighted sum of false prime implicants is 4. Therefore, $T_{bp} = (5 + 4) / 2 = 9/2 = 4.5$

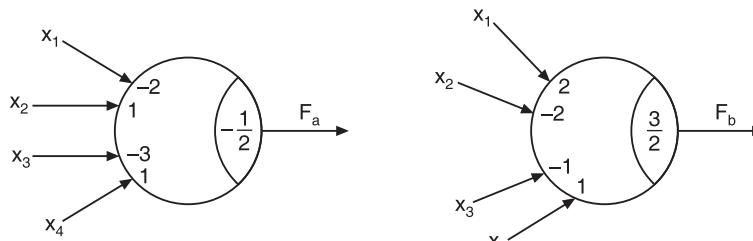
**Step 9. Obtain the weights and threshold for original functions**

The values for the weights and threshold of the original function are obtained as follows.

For $F_a$	For $F_b$
Since $\bar{x}_1 \rightarrow x_1$ and $\bar{x}_3 \rightarrow x_3$ $w_1 = -2$ and $w_3 = -3$ Therefore, $T_a = T_{ap} - w_1 - w_3 = 9/2 - 2 - 3 = -1/2$	Since $\bar{x}_2 \rightarrow x_2$ and $\bar{x}_3 \rightarrow x_3$ $w_2 = -2$ and $w_3 = -1$ Therefore, $T_b = T_{bp} - w_2 - w_3 = 9/2 - 2 - 1 = 3/2$

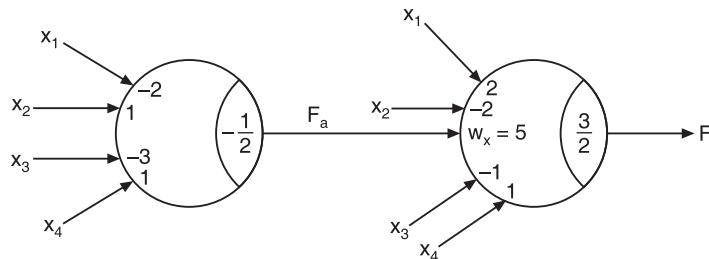
**Step 10. Realize the functions  $F_a$ ,  $F_b$  and  $F$  using threshold gates**

The realization of the two unate functions  $F_a$  and  $F_b$  separately and also the realization of the original function  $F$  are shown in Figures 9.32 and 9.33 respectively.



**Figure 9.32** Example 9.16: Realization of  $F_a$  and  $F_b$  using T-gates.

By cascading the two functions we have the threshold gate as shown in Figure 9.33.



**Figure 9.33** Example 9.16: Threshold gate.

*Value of  $w_x$ .* When the cascading is done as shown in Figure 9.33, it is desired to achieve OR operation in addition to the realization of  $F_b$  by the second threshold element since  $F = F_a + F_b$ . Now the weight  $w_x$  is to be determined. When  $F_a$  or  $F_b$  or both are 1,  $F$  must be 1 and  $w_x$  is to be determined to achieve this operation.

When  $F_a = 0$ , it does not affect the operation of the second threshold element, but when  $F_a = 1$ , for  $F$  to be 1, the weighted sum of the second element must be greater than  $T_b = 3/2$ . Therefore, the minimal weighted sum is to be obtained and  $F_a = 1$  corresponding to this must produce  $F = 1$ . For all other combinations of inputs, the weighted sum will be higher than this and this condition will be automatically satisfied.

The minimal weighted sum occurs when  $x_1 = x_4 = 0, x_2 = x_3 = 1$ . Therefore,  $w_x$  must satisfy the condition

$$-w_2 - w_3 + w_x = -2 - 1 + w_x = -3 + w_x \geq \frac{3}{2}$$

or

$$w_x \geq \frac{9}{2}$$

Therefore,  $w_x$  is chosen as 5.

In general, the above procedure can be employed for the realization of any arbitrary switching function.

### SHORT QUESTIONS AND ANSWERS

1. What is meant by the term ‘threshold logic’?
- A. Threshold logic is a form of logic realization using threshold elements.
2. What are the merits of a threshold gate?
- A. The merits of a threshold gate are as follows:
- (a) It is a much more powerful device than any of the conventional logic gates.
  - (b) Complex and large Boolean functions can be realized using much fewer threshold gates.
  - (c) Frequently a simple threshold gate can realize a very complex function which otherwise might require a large number of conventional gates.
  - (d) It offers incomparably economical realization.

3. What are the limitations of threshold gates?
- A. The limitations of the threshold gate are as follows:
  - (a) It is very sensitive to parameter variations.
  - (b) It is difficult to fabricate it in IC form.
  - (c) The speed of switching of threshold gates is much lower than that of conventional gates.
4. What can be the values of the threshold (T) and weights (ws) of a threshold gate?
- A. The values of the threshold (T) and weights (ws) of a threshold gate can be real, finite, positive or negative numbers.
5. How is the behaviour of a threshold element specified?
- A. The behaviour of a threshold element is specified by a relationship between its inputs and output involving the parameters ws and T. It is defined as  $F(x_1, x_2, \dots, x_n) = 1$  if and only if  $\sum_{i=1}^n w_i x_i \geq T$  otherwise  $F(x_1, x_2, \dots, x_n) = 0$ .
6. Why a threshold gate is called a universal gate?
- A. Since a threshold gate can realize universal gates, i.e. NAND gates and NOR gates, a threshold gate is also called a universal gate.
7. Can an X-OR gate be implemented using a single threshold gate? If not, how many T-gates are required?
- A. No. An X-OR gate cannot be implemented using a single threshold gate. It requires three T-gates. It can also be implemented using two T-gates.
8. Write the procedure for implementation of a Boolean function using the threshold gate.
- A. The following procedure is used to implement the Boolean function using the threshold gate.
  - Step 1.* Write the truth table such that column 1 specifies the combination number, column 2 specifies the inputs and column 3 specifies the function output ( either 0 or 1).
  - Step 2.* Add column 4 specifying weighted sums and their relation with the threshold. For input combinations where  $f = 1$ , write the weighted sum to be equal to or greater than the threshold (T). For input combinations where  $f = 0$ , write the weighted sum to be less than the threshold (T).
  - Step 3.* Determine the values for weights and threshold that satisfy all the relations specified in column 4.
  - Step 4.* If solution exists, the Boolean function is a threshold function; otherwise the Boolean function is not a threshold function.
9. When do you say that a function is positive in a variable?
- A. A function  $f(x_1, x_2, \dots, x_n)$  of binary variables  $x_1, x_2, \dots, x_n$  is said to be positive in a variable  $x_i$ , if there exists a disjunctive (minimal SOP) or conjunctive (minimal POS) expression for the function in which  $x_i$  appears only in uncomplemented form.
10. When do you say that the function is negative in a variable?
- A. A function  $f(x_1, x_2, \dots, x_n)$  of binary variables  $x_1, x_2, \dots, x_n$  is said to be negative in a variable  $x_i$ , if there exists a minimal SOP or minimal POS expression for the function f in which  $x_i$  appears only in complemented form.
11. When do you say that a function is unate in a variable  $x_i$ ?
- A. A function is said to be unate in a variable  $x_i$ , if that function is either only positive or only negative in  $x_i$ .

**12.** What do you mean by a unate function?

- A. A unate function is a function which is unate in each one of its variables, i.e. a unate function is a function which can be represented in minimal SOP form or minimal POS form in which no variable appears in both its uncomplemented and complemented forms.

**13.** What is a positive unate function?

- A. A positive unate function is one which has all its variables only in uncomplemented form.

**14.** What is a negative unate function?

- A. A negative unate function is one which has all its variables only in complemented form.

**15.** What is a threshold function?

- A. A logic function that can be realized by a single threshold element is known as a threshold function.

**16.** Is it necessary for a threshold function to be a unate function?

- A. Yes. A threshold function is always a unate function.

**17.** Is it necessary for a unate function to be a threshold function?

- A. No. A unate function may or may not be a threshold function.

**18.** Is it possible to convert a mixed unate function into a positive unate function?

- A. Yes. It is possible to convert a mixed unate function into a positive unate function. A unate function having some variables in complemented form and some in uncomplemented form can be converted into a positive unate function by converting each variable in complemented form to that in uncomplemented form.

**19.** Is every threshold function unate?

- A. Yes, every threshold function is unate.

**20.** Write the steps to be carried out for the synthesis of Boolean functions.

- A. To synthesize, i.e. to realize the Boolean expressions, the following steps are to be carried out.

*Step 1.* Draw the K-map for the given function, minimize it and write the minimal expression and check for the unateness of that minimal expression.

*Step 2.* If the function is unate, test whether it is positive in all its input variables. If not, convert it to another function that is positive in all its variables by replacing each complemented variable by its uncomplemented variable.

*Step 3.* Determine the true and false prime implicants of the positive function, by drawing the K-maps and obtaining the minimal expressions in SOP and POS forms.

*Step 4.* Obtain the inequalities such that the minimal weighted sum corresponding to each true prime implicant is greater than the maximal weighted sum of any false prime implicant.

*Step 5.* From the inequalities obtain the weighted constraints.

*Step 6.* Assign the values to the weights to satisfy the inequalities.

*Step 7.* Assign the value for threshold such that the threshold value is in between the minimal weighted sum of the true prime implicant and the maximal weighted sum of false prime implicant.

*Step 8.* Obtain the weights of the original function by putting a negative sign to the weights of the variables which were in complemented form in the original function. Calculate the threshold for the original function by subtracting the weights of the complemented variables in the positive function from the threshold for the positive function.

*Step 9.* Realize the function using the threshold gate.

**21.** Can non-threshold functions be realized using a single threshold gate?

- A. No. Non-threshold functions cannot be realized using a single threshold gate. Such functions can be realized using two or more threshold gates.

**REVIEW QUESTIONS**

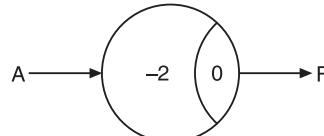
1. Is an X-OR function a threshold function? Justify.
2. Is an X-NOR function a threshold function? Justify.
3. Explain the steps to implement a Boolean function using threshold gate.

**FILL IN THE BLANKS**

1. The speed of switching threshold gates is much \_\_\_\_\_ than that of conventional gates.
2. It is \_\_\_\_\_ to fabricate threshold gates in IC form.
3. X-OR and X-NOR gates \_\_\_\_\_ be realized using a single threshold gate.
4. In a positive unate function all the variables are only in \_\_\_\_\_ form.
5. In a negative unate function all the variables are only in \_\_\_\_\_ form.
6. The minimal expression of a unate function is \_\_\_\_\_.
7. The complement of a unate function is \_\_\_\_\_.
8. A threshold function is having a variable B present as  $\bar{B}$ . It is converted into a positive unate function with associated weight 5. The weight associated with this variable in the original function will be \_\_\_\_\_.
9. A threshold function has  $T = 3$ . For a specific combination of values of variables for which output is 1, the weighted sum will be \_\_\_\_\_.
10. When two threshold elements are connected in cascade for the realization of a switching function, the interconnection between the two is required to perform \_\_\_\_\_ operation.
11. \_\_\_\_\_ functions cannot be realized using a single threshold gate.
12. \_\_\_\_\_ functions can be realized using a single threshold gate.

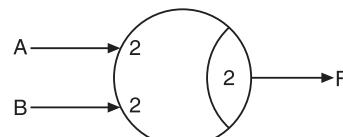
**OBJECTIVE TYPE QUESTIONS**

1. The T-gate shown below represents



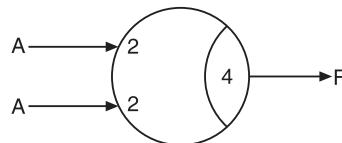
- (a) a NOT gate      (b) an AND gate      (c) an OR gate      (d) a NAND gate

2. The T-gate shown below represents



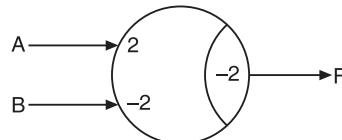
- (a) a NOR gate      (b) an AND gate      (c) an OR gate      (d) a NAND gate

3. The T-gate shown below represents



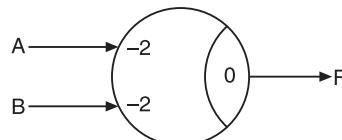
- (a) a NOR gate      (b) an AND gate      (c) an OR gate      (d) a NAND gate

4. The T-gate shown below represents a



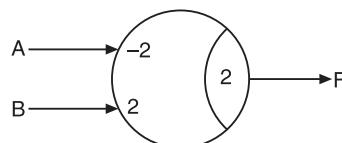
- (a) a NOR gate      (b) an AND gate      (c) an OR gate      (d) a NAND gate

5. The T-gate shown below represents



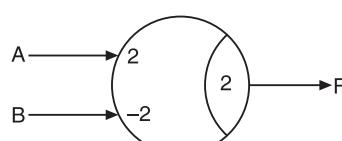
- (a) a NOR gate      (b) an AND gate      (c) an OR gate      (d) a NAND gate

6. The T-gate shown below represents  $F =$



- (a)  $\bar{A}\bar{B}$       (b)  $A\bar{B}$       (c)  $AB$       (d)  $\bar{A}\bar{B}$

7. The T-gate shown below represents  $F =$



- (a)  $\bar{A}\bar{B}$       (b)  $A\bar{B}$       (c)  $AB$       (d)  $\bar{A}\bar{B}$

8. The parameters of a threshold element are

- (a) weights assigned to input variables      (b) value of T  
 (c) weights assigned to inputs variables and T      (d) none of these

9. The number of outputs of a threshold element is

- (a) 1      (b) 2      (c) 3      (d) infinite

10. The weights assigned to the variables in a threshold element are

- (a) real      (b) finite  
 (c) positive or negative      (d) all of the above

PROBLEMS

- 9.1** Determine whether the following switching functions are unate functions:

  - (a)  $f(A, B, C, D) = \sum m(0, 1, 3, 4, 5, 6, 7, 12, 13)$
  - (b)  $f(A, B, C, D) = \sum m(5, 6, 7, 10, 11, 13, 14, 15)$

**9.2** Test whether the following Boolean expressions can be realized using a single threshold gate or not:

  - (a)  $F(A, B, C) = \sum m(1, 4, 6, 7)$
  - (b)  $F(A, B, C) = \sum m(0, 2, 4, 5, 7)$

**9.3** Realise the following Boolean functions using T-gate:

  - (a)  $f(w, x, y, z) = \sum m(0, 1, 4, 5, 6, 7, 12, 13)$
  - (b)  $f(x, y, z) = \sum m(0, 1, 4, 5, 7)$

**9.4** Realise the Boolean functions using T-gates:

  - (a)  $f(w, x, y, z) = \sum m(2, 3, 6, 7, 10, 14, 15)$
  - (b)  $f(w, x, y, z) = \sum m(3, 5, 7, 10, 12, 14, 15)$
  - (c)  $f(w, x, y, z) = \sum m(2, 3, 7, 8, 9, 12, 13, 14, 15)$

# 10

## FLIP-FLOPS

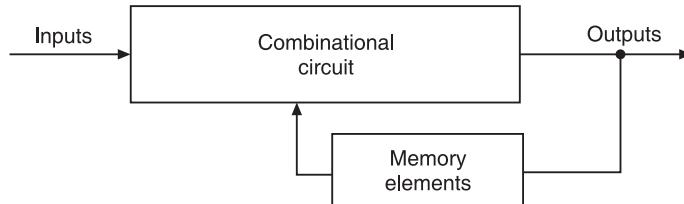
### 10.1 INTRODUCTION

Basically, switching circuits may be combinational switching circuits or sequential switching circuits. The switching circuits considered so far have been combinational switching circuits. Combinational switching circuits are those whose output levels at any instant of time are dependent only on the levels present at the inputs at that time. Any prior input level conditions have no effect on the present outputs, because combinational logic circuits have no memory. A circuit consisting of only logic gates is a combinational circuit. On the other hand, sequential switching circuits are those whose output levels at any instant of time are dependent not only on the levels present at the inputs at that time, but also on the state of the circuit, i.e. on the prior input level conditions (i.e. on its past inputs). The past history is provided by feedback from the output back to the input. It means that sequential switching circuits have memory. Sequential circuits are thus made of combinational circuits and memory elements. The past history is provided by feedback from the output back to the input.

There are many applications in which digital outputs are required to be generated in accordance with the sequence in which the input signals are received. This requirement cannot be satisfied using a combinational logic system. These applications require outputs to be generated that are not only dependent on the present input conditions, but also upon the past history of the inputs. Hence sequential circuits come into picture.

Parallel adders, subtractors, encoders, decoders, code converters, parity bit generators, etc. are examples of combinational circuits. Counters, shift registers, serial adders, sequence generators, logic function generators, etc. are examples of sequential circuits.

Figure 10.1 shows a block diagram of a sequential circuit. The memory elements are connected to the combinational circuit as a feedback path.



**Figure 10.1** Block diagram of a sequential circuit.

The information stored in the memory element at any given time defines the present state of the sequential circuit. The present state and the external inputs determine the outputs and the next state of the sequential circuit. Thus, we can specify the sequential circuit by a time sequence of external inputs, internal states (present state and next state) and outputs. Table 10.1 presents the comparison between combinational and sequential circuits.

**Table 10.1** Comparison between combinational and sequential circuits

Combinational circuits	Sequential circuits
<ol style="list-style-type: none"> <li>In combinational circuits, the output variables at any instant of time are dependent only on the present input variables.</li> <li>Memory unit is not required in combinational circuits.</li> <li>Combinational circuits are faster because the delay between the input and the output is due to propagation delay of gates only.</li> <li>Combinational circuits are easy to design.</li> </ol>	<ol style="list-style-type: none"> <li>In sequential circuits, the output variables at any instant of time are dependent not only on the present input variables, but also on the present state, i.e. on the past history of the system.</li> <li>Memory unit is required to store the past history of the input variables in sequential circuits.</li> <li>Sequential circuits are slower than combinational circuits.</li> <li>Sequential circuits are comparatively harder to design.</li> </ol>

## 10.2 CLASSIFICATION OF SEQUENTIAL CIRCUITS

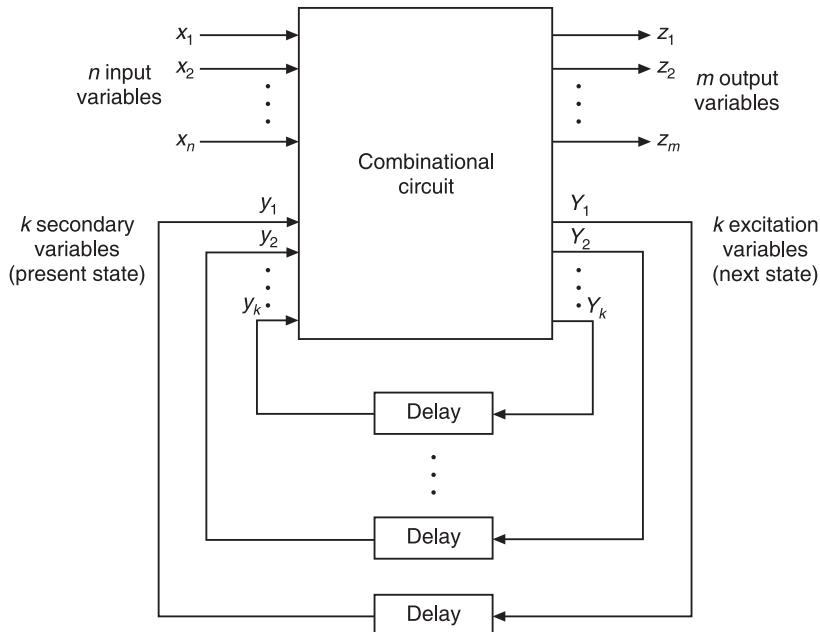
The sequential circuits may be classified as *synchronous sequential circuits* and *asynchronous sequential circuits* depending on the timing of their signals. The sequential circuits which are controlled by a clock are called synchronous sequential circuits. These circuits will be active only when clock signal is present. The sequential circuits which are not controlled by a clock are called asynchronous sequential circuits, i.e. the sequential circuits in which events can take place any time the inputs are applied are called asynchronous sequential circuits. Periodically, recurring pulse is called a clock. It is generated by a pulse generator. In sequential circuits the desired operations take place only when the clock pulse occurs, as they are enabled using AND gates wherever needed. All flow of information occurs only when the clock pulse occurs. Table 10.2 shows the comparison between synchronous and asynchronous sequential circuits.

**Table 10.2** Comparison between synchronous and asynchronous sequential circuits

Synchronous sequential circuits	Asynchronous sequential circuits
1. In synchronous circuits, memory elements are clocked FFs.	1. In asynchronous circuits, memory elements are either unclocked FFs or time delay elements.
2. In synchronous circuits, the change in input signals can affect memory elements upon activation of clock signal.	2. In asynchronous circuits, change in input signals can affect memory elements at any instant of time.
3. The maximum operating speed of the clock depends on time delays involved.	3. Because of the absence of the clock, asynchronous circuits can operate faster than synchronous circuits.
4. Easier to design.	4. More difficult to design.

### 10.3 LEVEL MODE AND PULSE MODE ASYNCHRONOUS SEQUENTIAL CIRCUITS

Figure 10.2 shows a block diagram of an asynchronous sequential circuit. It consists of a combinational circuit and delay elements connected to form the feedback loops. There are  $n$  input variables,  $m$  output variables and  $k$  internal states. The delay elements provide a short-term memory for the sequential circuit. The present state and next state variables in asynchronous sequential circuits are called secondary variables and excitation variables respectively.

**Figure 10.2** Block diagram of an asynchronous sequential circuit.

When an input variable changes in value, the secondary variables, i.e.  $y_1, y_2, \dots, y_k$  do not change instantaneously. Certain amount of time is required for the input signal to propagate from

the input terminals through the combinational circuit and the delay elements. The combinational circuit generates  $k$  excitation variables which give the next state of the circuit. The excitation variables are propagated through delay elements to become the new present state for the secondary variables, i.e.  $y_1, y_2, \dots, y_k$ . In the steady state condition excitation and secondary variables are the same, but during transition they are different. In other words, we can say that for a given value of input variables, the system is stable if the circuit reaches a steady-state condition with  $y_i = Y_i$  for  $i = 1, 2, \dots, k$ ; otherwise the circuit is in a continuous transition and is said to be unstable.

To ensure proper operation, it is necessary for asynchronous sequential circuits to attain a stable state before the input is changed to a new value. Because of unequal delays in the wires and gate circuits, it is impossible to have two or more input variables change at exactly the same instant. Therefore, simultaneous changes of two or more input variables are usually avoided. In other words, we can say that only one input variable is allowed to change at any one time and the time between two input changes is kept longer than the time it takes the circuit to reach a stable state.

According to how the input variables are to be considered, there are two types of asynchronous circuits: *fundamental mode circuits* and *pulse mode circuits*.

Fundamental mode circuit assumes that:

1. The input variables change only when the circuit is stable.
2. Only one input variable can change at a given time.
3. Inputs are levels and not pulses.

Pulse mode circuit assumes that:

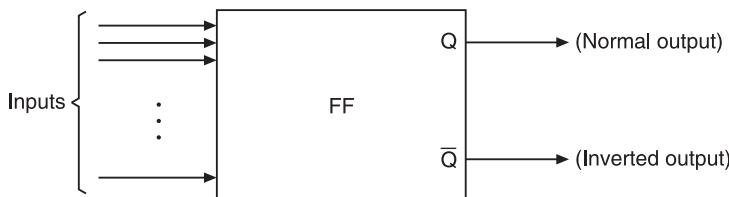
1. The input variables are pulses instead of levels.
2. The width of the pulses is long enough for the circuit to respond to the input.
3. The pulse width must not be so long that it is still present after the new state is reached.

## 10.4 LATCHES AND FLIP-FLOPS

The most important memory element is the flip-flop, which is made up of an assembly of logic gates. Even though a logic gate by itself has no storage capability, several logic gates can be connected together in ways that permit information to be stored. There are several different gate arrangements that are used to construct flip-flops in a wide variety of ways. Each type of flip-flop has special features or characteristics necessary for particular applications. Flip-flops are the basic building blocks of most sequential circuits.

A flip-flop (FF), known more formally as a bistable multivibrator, has two stable states. It can remain in either of the states indefinitely. Its state can be changed by applying the proper triggering signal. It is also called a binary or one-bit memory.

Figure 10.3 shows the general type of symbol used for a flip-flop. The flip-flop has two outputs, labelled  $Q$  and  $\bar{Q}$ . Actually any letter can be used to represent the output, but  $Q$  is the one most often used. The  $Q$  output is the normal output of the flip-flop and  $\bar{Q}$  is the inverted output. The output as well as its complement are available for each flip-flop. The state of the flip-flop always refers to the state of the normal output  $Q$ . The inverted output  $\bar{Q}$  is in the opposite state. A flip-flop is said to be in HIGH state or logic 1 state or SET state when  $Q = 1$ , and in LOW state or logic 0 state or RESET state or CLEAR state when  $Q = 0$ .



**Figure 10.3** General flip-flop symbol.

As the symbol in Figure 10.3 implies, a flip-flop can have one or more inputs. *The input signals which command the flip-flop to change state are called excitations.* These inputs are used to cause the flip-flop to switch back and forth (i.e. ‘flip-flop’) between its possible output states. A flip-flop input has to be pulsed momentarily to cause a change in the flip-flop output, and the output will remain in that new state even after the input pulse has been removed. This is the flip-flop’s memory characteristic.

There are a number of applications of flip-flops. As such, the flip-flop serves as a storage device. It stores a 1 when its Q output is a 1, and stores a 0 when its Q output is a 0. Flip-flops are the fundamental components of shift registers and counters.

The term ‘latch’ is used for certain flip-flops. It refers to non-clocked flip-flops, because these flip-flops ‘latch on’ to a 1 or a 0 immediately upon receiving the input pulse called SET or RESET. They are not dependent on the clock signal for their operation, i.e. a latch is a sequential device that checks all its inputs continuously and changes its outputs accordingly at any time independent of a clock signal. Gated latches (clocked flip-flops) are latches which respond to the inputs and latch on to a 1 or a 0 only when they are enabled, i.e. only when the input ENABLE or gating signal is HIGH. In the absence of ENABLE or gating signal, the latch does not respond to the changes in its inputs (The gating signal may be a clock pulse). On the other hand, a flip-flop is a sequential device that normally samples its inputs and changes its outputs only at times determined by clock pulses.

A latch may be an active-HIGH input latch or an active-LOW input latch. Active-HIGH means that the SET and RESET inputs are normally resting in the LOW state and one of them will be pulsed HIGH whenever we want to change the latch outputs. Active-LOW means that the SET and RESET inputs are normally resting in the HIGH state and one of them will be pulsed LOW whenever we want to change the latch outputs.

#### 10.4.1 The S-R Latch

The simplest type of flip-flop is called an S-R latch. It has two outputs labelled Q and  $\bar{Q}$  and two inputs labelled S and R. The state of the latch corresponds to the level of Q (HIGH or LOW, 1 or 0) and  $\bar{Q}$  is, of course, the complement of that state. It can be constructed using either two cross-coupled NAND gates or two-cross coupled NOR gates. Using two NOR gates, an active-HIGH S-R latch can be constructed and using two NAND gates an active-LOW S-R latch can be constructed. The name of the latch, S-R or SET-RESET, is derived from the names of its inputs.

Figure 10.4a shows the logic symbol of an S-R latch. When the SET input is made HIGH, Q becomes 1 (and  $\bar{Q}$  equals 0). When the RESET input is made HIGH, Q becomes 0 (and  $\bar{Q}$  equals 1). If both the inputs S and R are made LOW, there is no change in the state of the latch. It means that

the latch remains in the same state in which it was, prior to the application of inputs. If both the inputs are made HIGH, the output is unpredictable, i.e. both Q and  $\bar{Q}$  may be HIGH, or both may be LOW or any one of them may be HIGH and the other LOW. This condition is described as not-allowed, unpredictable, invalid or indeterminate. The S-R latch is also called R-S latch or S-C (SET-CLEAR) latch. Resetting is also called clearing because we CLEAR out the 1 in the output by resetting to 0. In more complex flip-flops, called gated latches, the change of state does not take place immediately after the application of the inputs. The change of state takes place only after applying a gate pulse.

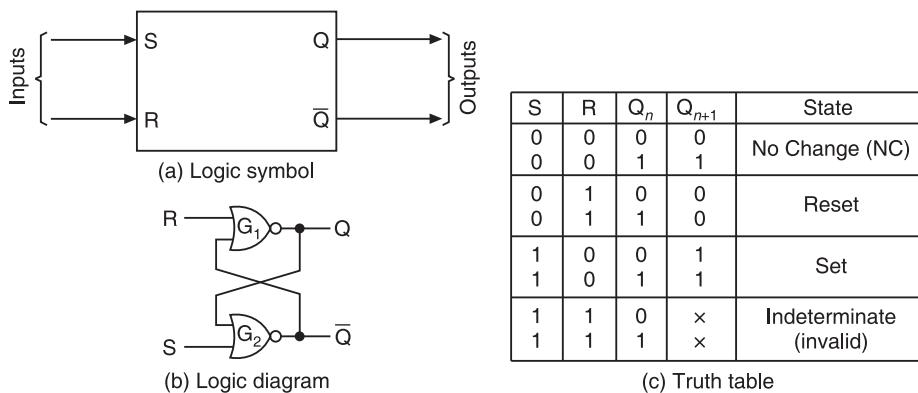


Figure 10.4 Active-HIGH S-R latch.

**The NOR gate S-R latch (active-high S-R latch):** Figure 10.4b shows the logic diagram of an active-HIGH S-R latch composed of two cross-coupled NOR gates. Note that the output of each gate is connected to one of the inputs of the other gate. The latch works as per the truth table of Figure 10.4c.  $Q_n$  represents the state of the flip-flop before applying the inputs (i.e. the present state PS of the flip-flop).  $Q_{n+1}$  represents the state of the flip-flop after the application of the inputs (i.e. the next state NS of the flip-flop).

It is necessary only to pulse a SET or RESET input to change the state of the latch. For example, if the latch is initially RESET, a pulse applied to its SET is the same as making S momentarily a 1, followed by a 0. The 1 sets the latch, after which R and S are once again a 0, the no-change condition. Since a pulse must remain HIGH long enough for NOR gates to change states, the minimum pulse width is the sum of the propagation delays through the gates. One gate must change from LOW to HIGH and the other from HIGH to LOW. Thus,

$$PW_{\min} = t_{PLH} + t_{PHL}$$

where  $PW_{\min}$  is the minimum pulse width required for proper operation of the gate,  $t_{PLH}$  and  $t_{PHL}$  are the propagation delays associated with the gates when the output is changing from LOW to HIGH and HIGH to LOW, respectively.

The analysis of the operation of the active-HIGH NOR latch can be summarized as follows.

1. **SET = 0, RESET = 0:** This is the normal resting state of the NOR latch and it has no effect on the output state. Q and  $\bar{Q}$  will remain in whatever state they were prior to the occurrence of this input condition.

2. **SET = 1, RESET = 0:** This will always set  $Q = 1$ , where it will remain even after SET returns to 0.
3. **SET = 0, RESET = 1:** This will always reset  $Q = 0$ , where it will remain even after RESET returns to 0.
4. **SET = 1, RESET = 1:** This condition tries to SET and RESET the latch at the same time, and it produces  $Q = \bar{Q} = 0$ . If the inputs are returned to zero simultaneously, the resulting output state is erratic and unpredictable. This input condition should not be used. It is forbidden.

The SET and RESET inputs are normally in the LOW state and one of them will be pulsed HIGH, whenever we want to change the latch outputs.

**The NAND gate S-R latch (active-low S-R latch):** Figures 10.5a and b show the logic diagram and truth table of an active-LOW S-R latch. Since the NAND gate is equivalent to an active-LOW OR gate, an active-LOW S-R latch using OR gates may also be represented as shown in Figure 10.5c.

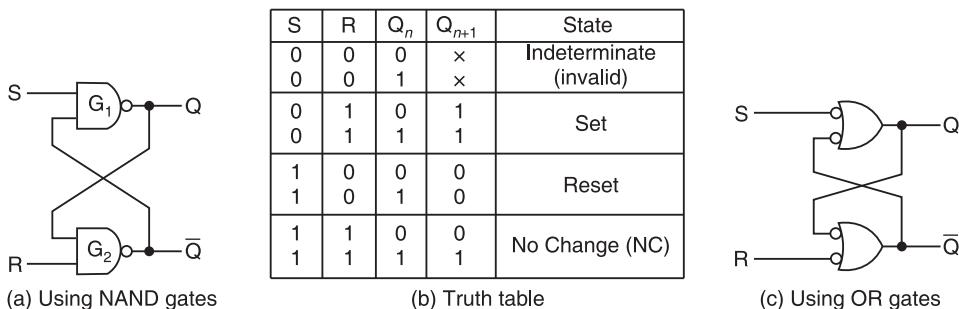


Figure 10.5 An active-LOW S-R latch.

The operation of this latch is the reverse of the operation of the NOR gate latch discussed earlier. That is why it is called an active-LOW S-R latch. If the 0s are replaced by 1s and 1s by 0s in Figure 10.5b, we get the same truth table as that of the NOR gate latch shown in Figure 10.4c.

The SET and RESET inputs are normally resting in the HIGH state and one of them will be pulsed LOW, whenever we want to change the latch outputs.

**The  $\bar{S}\bar{R}$  latch (active-high NAND latch):** An active-LOW NAND latch can be converted into an active-HIGH NAND latch by inserting the inverters at the S and R inputs. Figure 10.6 shows the proof.

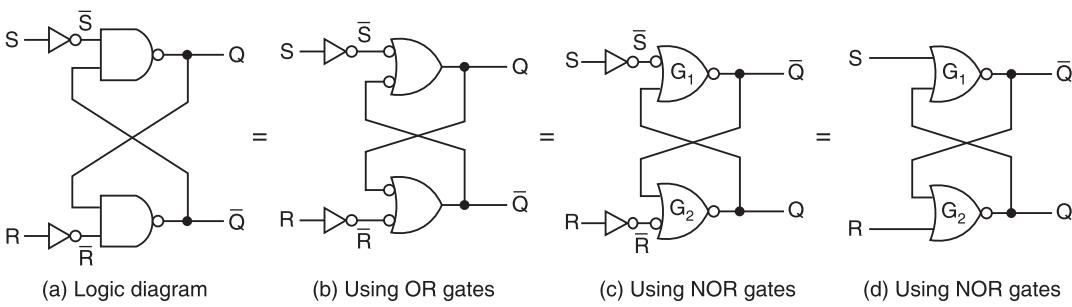


Figure 10.6 An active-HIGH NAND latch.

When power is applied to a circuit, it is not possible to predict the starting state of a flip-flop output, when its SET and RESET inputs are in their inactive states (i.e.  $S = R = 1$  for a NAND latch, and  $S = R = 0$  for a NOR latch). There is just as much chance that the starting state will be a  $Q = 0$  as  $Q = 1$ . It will depend on things like internal propagation delays, parasitic capacitance and external loading.

#### 10.4.2 Gated Latches (Clocked Flip-Flops)

**The gated S-R latch:** In the latches described earlier, the output can change state any time the input conditions are changed. So, they are called *asynchronous* latches. A gated S-R latch requires an ENABLE (EN) input. Its S and R inputs will control the state of the flip-flop only when the ENABLE is HIGH. When the ENABLE is LOW, the inputs become ineffective and no change of state can take place. The ENABLE input may be a clock. So, a gated S-R latch is also called a *clocked S-R latch or synchronous S-R latch*. Since this type of flip-flop responds to the changes in inputs only as long as the clock is HIGH, these types of flip-flops are also called *level triggered flip-flops*. The sequential circuits (machines) controlled by a clock are called synchronous sequential circuits (machines). The logic diagram, the logic symbol and the truth table for a gated S-R latch are shown in Figure 10.7. In this circuit, the invalid state occurs when both S and R are simultaneously HIGH.

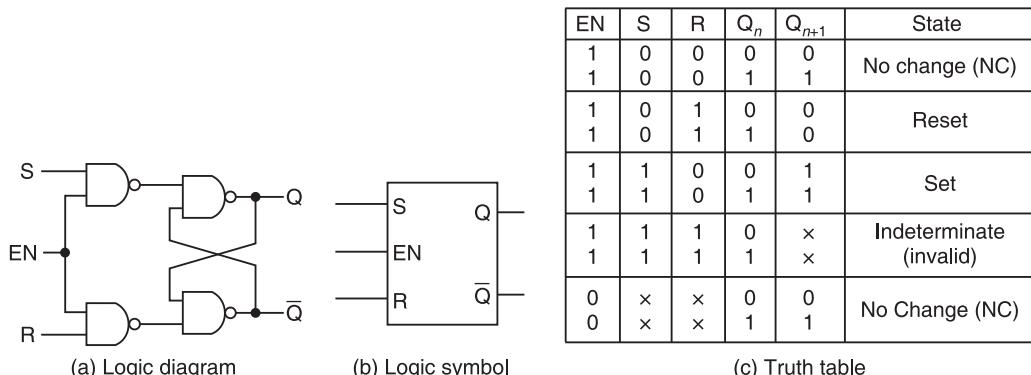


Figure 10.7 A gated S-R latch.

**EXAMPLE 10.1** Determine the output waveform Q if the inputs shown in Figure 10.8a are applied to a gated S-R latch shown in Figure 10.8b, that was initially SET.

#### Solution

The output waveform Q shown in Figure 10.8c is drawn as follows:

Prior to  $t_0$ , Q is HIGH. Even though R goes HIGH prior to  $t_0$ , Q will not change because EN is LOW. Similarly, even though S goes HIGH prior to  $t_1$ , Q will not change because EN is LOW. Any time S is HIGH and R is LOW, a HIGH on the EN sets the latch, and any time S is LOW and R is HIGH, a HIGH on the EN resets the latch.

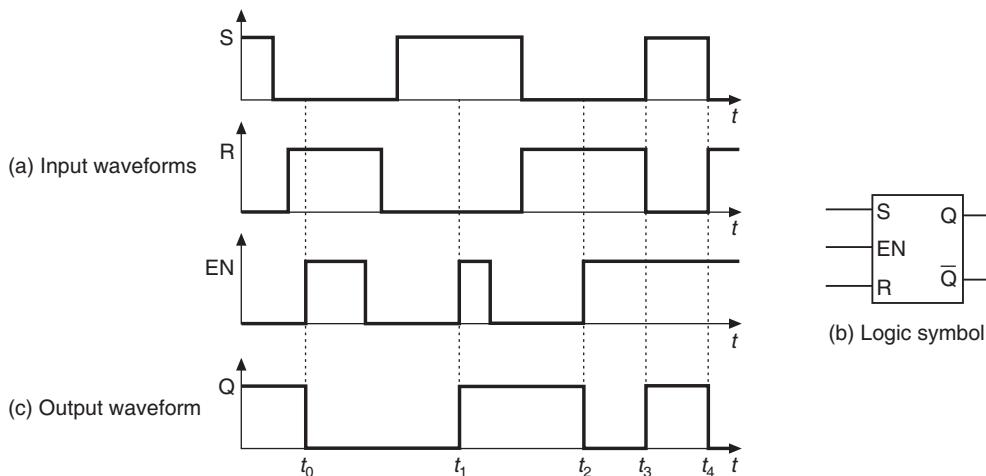


Figure 10.8 Example 10.1: Waveforms—the gated S-R latch.

**The gated D-latch:** In many applications, it is not necessary to have separate S and R inputs to a latch. If the input combinations  $S = R = 0$  and  $S = R = 1$  are never needed, the S and R are always the complement of each other. So, we can construct a latch with a single input (S) and obtain the R input by inverting it. This single input is labelled D (for data) and the device is called a D-latch. So, another type of gated latch is the gated D-latch. It differs from the S-R latch in that it has only one input in addition to EN. When  $D = 1$ , we have  $S = 1$  and  $R = 0$ , causing the latch to SET when ENABLED. When  $D = 0$ , we have  $S = 0$  and  $R = 1$ , causing the latch to RESET when ENABLED. When EN is LOW, the latch is ineffective, and any change in the value of D input does not affect the output at all. When EN is HIGH, a LOW D input makes Q LOW, i.e. resets the flip-flop and a HIGH D input makes Q HIGH, i.e. sets the flip-flop. In other words, we can say that the output Q follows the D input when EN is HIGH. So, this latch is said to be transparent. The logic diagram, the logic symbol and the truth table of a gated D-latch are shown in Figure 10.9.

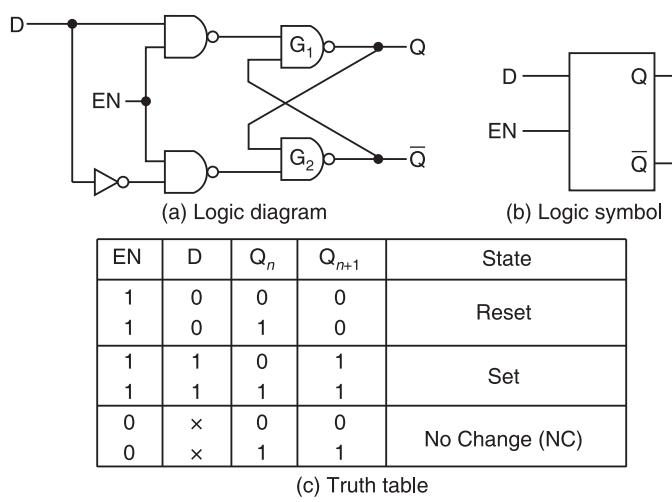
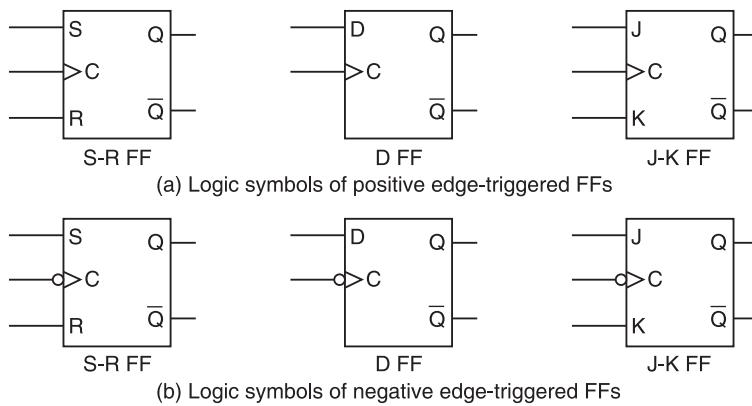


Figure 10.9 A gated D-latch.

### 10.4.3 Edge-Triggered Flip-Flops

Digital systems can operate either synchronously or asynchronously. In asynchronous systems, the outputs of logic circuits can change state any time, when one or more of the inputs change. An asynchronous system is difficult to design and troubleshoot. In synchronous systems, the exact times at which any output can change states are determined by a signal commonly called the *clock*. The flip-flops using the clock signal are called the *clocked flip-flops*. Control signals are effective only if they are applied in synchronization with the clock signal. The clock signal is distributed to all parts of the system and most of the system outputs can change state only when the clock makes a transition. Clocked flip-flops may be positive edge-triggered or negative edge-triggered. Positive edge-triggered flip-flops are those in which ‘state transitions’ take place only at the positive-going (0 to 1, or LOW to HIGH) edge of the clock pulse, and negative edge-triggered flip-flops are those in which ‘state transitions’ take place only at the negative-going (1 to 0, or HIGH to LOW) edge of the clock signal. Positive-edge triggering is indicated by a ‘triangle’ at the clock terminal of the flip-flop. Negative-edge triggering is indicated by a ‘triangle’ with a bubble at the clock terminal of the flip-flop. Thus edge-triggered flip-flops are sensitive to their inputs only at the transition of the clock. There are three basic types of edge-triggered flip-flops: S-R, J-K, and D (Figure 10.10). Of these, D and J-K flip-flops are the most widely used ones and readily available in the IC form than is the S-R type. But the S-R flip-flop is also covered here because it is a good base upon which to build both the D and the J-K flip-flops, having been derived from the S-R flip-flop.

The edge-triggering is also called *dynamic triggering*.

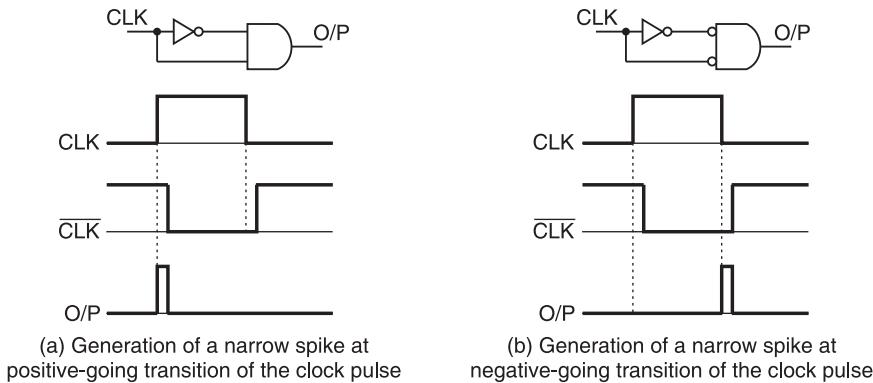


**Figure 10.10** Edge-triggered flip-flops.

**Generation of narrow spikes:** A narrow positive spike is generated at the rising edge of the clock using an inverter and an AND gate as shown in Figure 10.11a. The inverter produces a delay of a few nanoseconds. The AND gate produces an output spike that is HIGH only for a few nanoseconds, when CLK and  $\bar{\text{CLK}}$  are both HIGH. This results in a narrow pulse at the output of the AND gate which occurs at the positive-going transition of the clock signal.

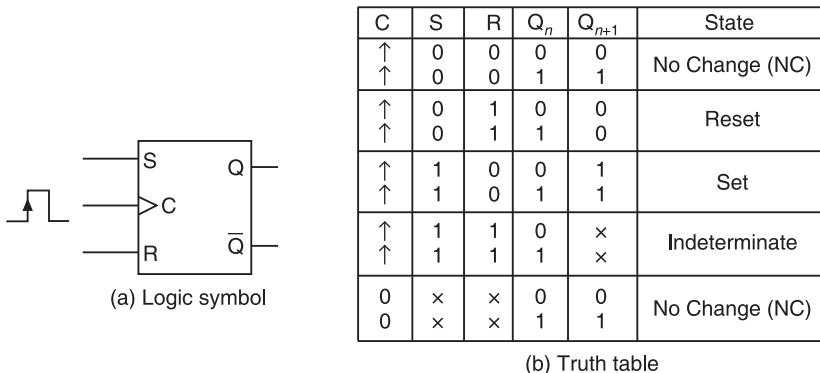
Similarly, a narrow positive spike is generated at the falling edge of the clock by using an inverter and an active-LOW AND gate as shown in Figure 10.11b. The inverter produces a delay of a few nanoseconds. The active-LOW AND gate produces an output spike that is HIGH only for

a few nanoseconds, when CLK and  $\overline{\text{CLK}}$  are both LOW. This results in a narrow pulse at the output of the AND gate which occurs at the negative-going transition of the clock.



**Figure 10.11** Generation of narrow spikes using edge detector.

**The edge-triggered S-R flip-flop:** Figure 10.12 shows the logic symbol and the truth table for a positive edge-triggered S-R flip-flop. The S and R inputs of the S-R flip-flop are called the *synchronous control inputs* because data on these inputs affect the flip-flop's output only on the triggering (positive going) edge of the clock pulse. Without a clock pulse, the S and R inputs cannot affect the output. When S is HIGH and R is LOW, the Q output goes HIGH on the positive-going edge of the clock pulse and the flip-flop is SET. (If it is already in SET state, it remains SET). When S is LOW and R is HIGH, the Q output goes LOW on the positive-going edge of the clock pulse and the flip-flop is RESET, i.e. cleared. (If it is already in RESET state, it remains RESET). When both S and R are LOW, the output does not change from its prior state (If it is in SET state, it remains SET and if it is in RESET state, it remains RESET). When both S and R are HIGH simultaneously, an invalid condition exists. The basic operation described above is illustrated in Figure 10.12b.



**Figure 10.12** Positive edge-triggered S-R flip-flop.

The truth table of a negative edge-triggered S-R flip-flop is the same as that of a positive edge triggered S-R flip-flop except that the arrows point downwards. This flip-flop will trigger only when the clock input goes from 1 to 0.

**EXAMPLE 10.2** The waveforms shown in Figure 10.13a are applied to the positive edge-triggered S-R flip-flop shown in Figure 10.13b. Sketch the output waveforms.

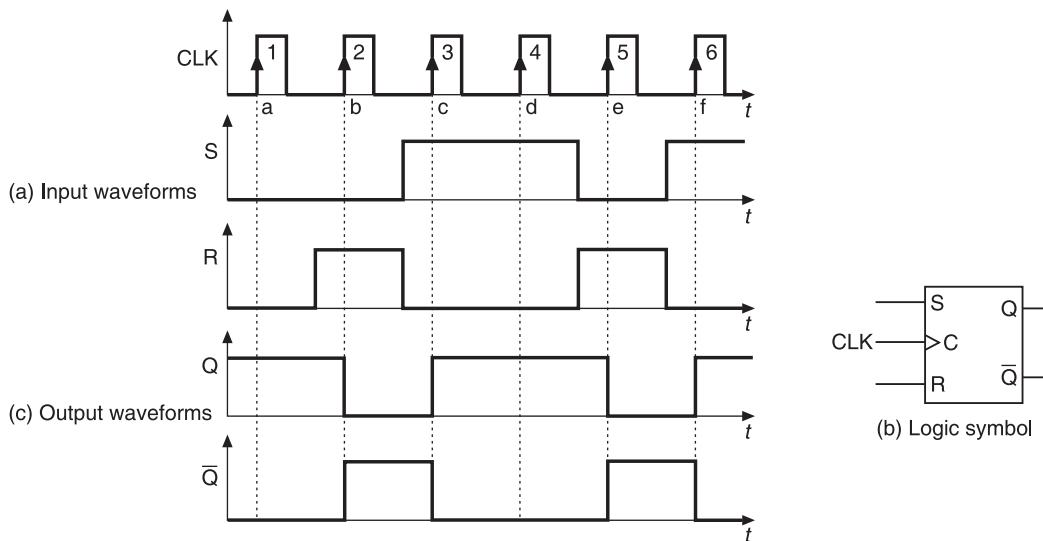


Figure 10.13 Example 10.2: Waveforms—positive edge-triggered S-R flip-flop.

### Solution

The output waveform is drawn as shown in Figure 10.13c after going through the following steps.

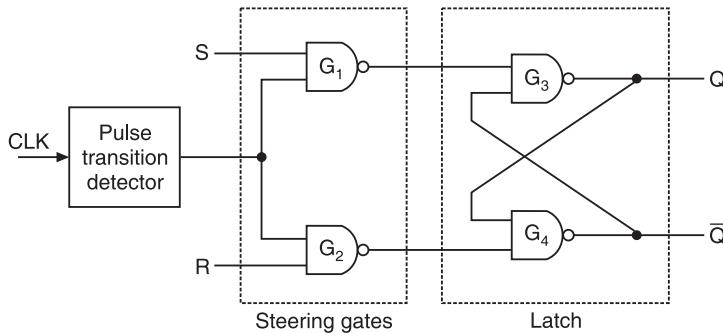
1. Initially,  $S = 0$  and  $R = 0$  and  $Q$  is assumed to be HIGH.
2. At the positive-going transition of the first clock pulse (i.e. at a), both  $S$  and  $R$  are LOW. So, no change of state takes place.  $Q$  remains HIGH and  $\bar{Q}$  remains LOW.
3. At the leading edge of the second clock pulse (i.e. at b),  $S = 0$  and  $R = 1$ . So, the flip-flop resets. Hence,  $Q$  goes LOW and  $\bar{Q}$  goes HIGH.
4. At the positive-going edge of the third clock pulse (i.e. at c),  $S = 1$  and  $R = 0$ . So, the flip-flop sets. Hence,  $Q$  goes HIGH and  $\bar{Q}$  goes LOW.
5. At the rising edge of the fourth clock pulse,  $S = 1$  and  $R = 0$ . Since the flip-flop is already in a SET state, it remains SET. That is,  $Q$  remains HIGH and  $\bar{Q}$  remains LOW.
6. The fifth pulse resets the flip-flop at its positive-going edge because  $S = 0$  and  $R = 1$  is the input condition and  $Q = 1$  at that time.
7. The sixth pulse sets the flip-flop at its rising edge because  $S = 1$  and  $R = 0$  is the input condition and  $Q = 0$  at that time.

*Internal circuitry of the edge-triggered S-R flip-flop:* A detailed analysis of the internal circuitry of a flip-flop is not necessary, since all types are readily available as ICs. A simplified description is, however, presented here. Figure 10.14 shows the simplified circuitry of the edge-triggered S-R flip-flop.

It contains three sections.

1. A basic NAND gate latch formed by NAND gates  $G_3$  and  $G_4$

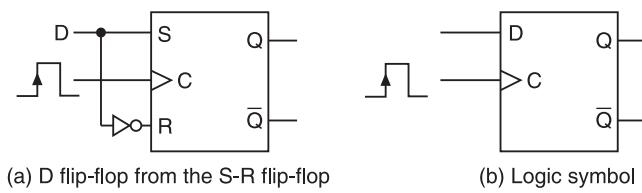
2. A pulse steering circuit formed by NAND gates  $G_1$  and  $G_2$
3. An edge (pulse transition) detector circuit



**Figure 10.14** Simplified circuit diagram of the edge-triggered S-R flip-flop.

The edge detector generates a positive spike at the positive-going or negative-going edge of the clock pulse. The steering gates ‘direct’ or ‘steer’ the narrow spike either to  $G_3$  or to  $G_4$  depending on the state of the S and R inputs.

**The edge-triggered D flip-flop:** The edge-triggered D flip-flop has only one input terminal. The D flip-flop may be obtained from an S-R flip-flop by just putting one inverter between the S and R terminals (Figure 10.15a). Figures 10.15b and c show the logic symbol and the truth table of a positive edge-triggered D flip-flop. Note that, this flip-flop has only one synchronous control input in addition to the clock. This is called the D (data) input. The operation of the D flip-flop is very simple. The output Q will go to the same state that is present on the D input at the positive-going transition of the clock pulse. In other words, the level present at D will be stored in the flip-flop at the instant the positive-going transition occurs. That is, if D is a 1 and the clock is applied, Q goes to a 1 and  $\bar{Q}$  to a 0 at the rising edge of the pulse and thereafter remain so. If D is a 0 and the clock is applied, Q goes to a 0 and  $\bar{Q}$  to a 1 at the rising edge of the clock pulse and thereafter remain so.



**(a)** D flip-flop from the S-R flip-flop      **(b)** Logic symbol

C	D	$Q_n$	$Q_{n+1}$	State
↑	0	0	0	Reset
↑	0	1	0	
↑	1	0	1	Set
↑	1	1	1	
0	x	0	0	No Change (NC)
0	x	1	1	

**(c)** Truth table

**Figure 10.15** The positive edge-triggered D flip-flop.

*Internal circuitry of the edge-triggered D flip-flop:* Figure 10.16 shows the internal circuitry of the edge-triggered D flip-flop.

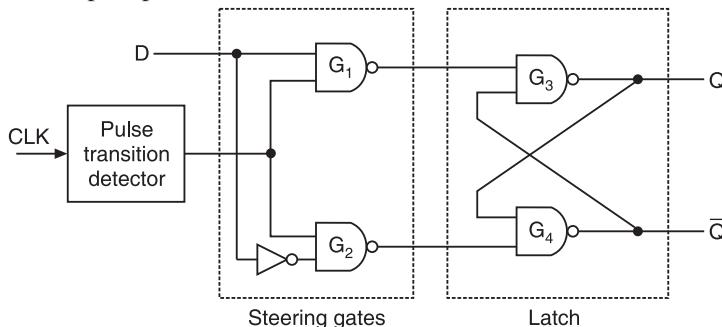


Figure 10.16 Simplified circuit diagram of the edge-triggered D flip-flop.

The negative edge-triggered D flip-flop operates in the same way as the positive edge-triggered D flip-flop except that the change of state takes place at the negative-going edge of the clock pulse. In the truth table of the negative edge triggered flip-flop the arrows point downwards.

**The edge-triggered J-K flip-flop:** The J-K flip-flop is very versatile and also the most widely used. The J and K designations for the synchronous control inputs have no known significance.

The functioning of the J-K flip-flop is identical to that of the S-R flip-flop, except that it has no invalid state like that of the S-R flip-flop. The logic symbol and the truth table for a positive edge-triggered J-K flip-flop are shown in Figure 10.17.

C	J	K	Q <sub>n</sub>	Q <sub>n+1</sub>	State
↑	0	0	0	0	No Change (NC)
↑	0	0	1	1	
↑	0	1	0	0	Reset
↑	0	1	1	0	
↑	1	0	0	1	Set
↑	1	0	1	1	
↑	1	1	0	1	Toggle
↑	1	1	1	0	
0	x	x	0	0	No Change (NC)
0	x	x	1	1	

(a) Logic symbol
(b) Truth table

Figure 10.17 Positive edge-triggered J-K flip-flop.

When  $J = 0$  and  $K = 0$ , no change of state takes place even if a clock pulse is applied.

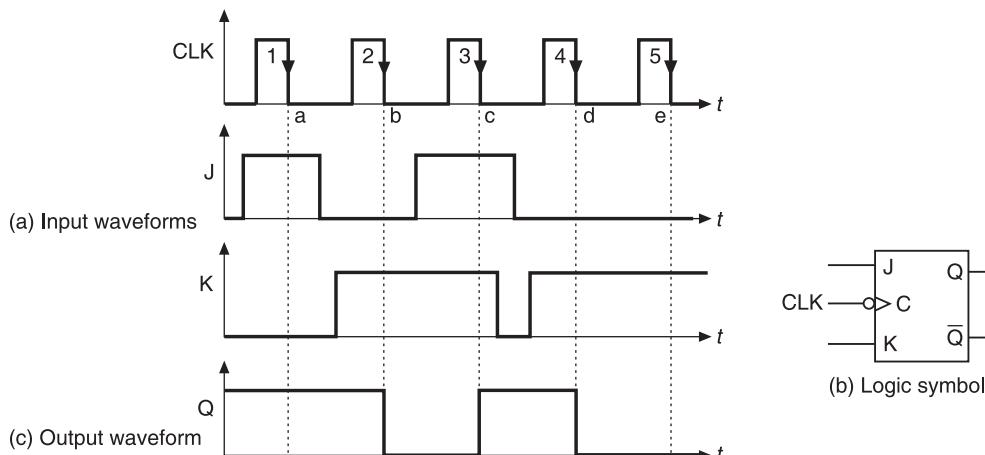
When  $J = 0$  and  $K = 1$ , the flip-flop resets at the positive-going edge of the clock pulse.

When  $J = 1$  and  $K = 0$ , the flip-flop sets at the positive-going edge of the clock pulse.

When  $J = 1$  and  $K = 1$ , the flip-flop toggles, i.e. goes to the opposite state at the positive-going edge of the clock pulse. In this mode, the flip-flop toggles or changes state for each occurrence of the positive-going edge of the clock pulse.

A negative edge-triggered J-K flip-flop operates in the same way as a positive edge-triggered J-K flip-flop except that the change of state takes place at the negative-going edge of the clock pulse. In the truth-table of a negative edge-triggered J-K flip-flop the arrows point downwards.

**EXAMPLE 10.3** The waveforms shown in Figure 10.18a are applied to the edge-triggered J-K flip-flop shown in Figure 10.18b. Draw the output waveform.



**Figure 10.18** Example 10.3: Waveforms—edge-triggered J-K flip-flop.

### Solution

The output waveform shown in Figure 10.18c is drawn as explained below:

1. Initially  $J = 0$ ,  $K = 0$  and  $CLK = 0$ . Assume that the initial state of the flip-flop is a 1, i.e.  $Q = 1$  initially.
2. At the negative-going edge of the first clock pulse (i.e. at a),  $J = 1$  and  $K = 0$ . So,  $Q$  remains as a 1 and, therefore,  $\bar{Q}$  as a 0.
3. At the trailing edge of the second clock pulse (i.e. at b),  $J = 0$  and  $K = 1$ . So, the flip-flop resets. That is,  $Q$  goes to a 0 and  $\bar{Q}$  to a 1.
4. At the falling edge of the third clock pulse (i.e. at c), both  $J$  and  $K$  are a 1. So, the flip-flop toggles. That is,  $Q$  changes from a 0 to a 1 and  $\bar{Q}$  from a 1 to a 0.
5. At the negative-going transition of the fourth clock pulse (i.e. at d),  $J = 0$  and  $K = 1$ . So, the flip-flop RESETS, i.e.  $Q$  goes to a 0 and  $\bar{Q}$  to a 1.
6. At the negative going edge of the fifth clock pulse (i.e. at d),  $J = 0$  and  $K = 1$ . So the flip-flop remains reset, i.e.  $Q$  remains as 0 and  $\bar{Q}$  remains as 1.

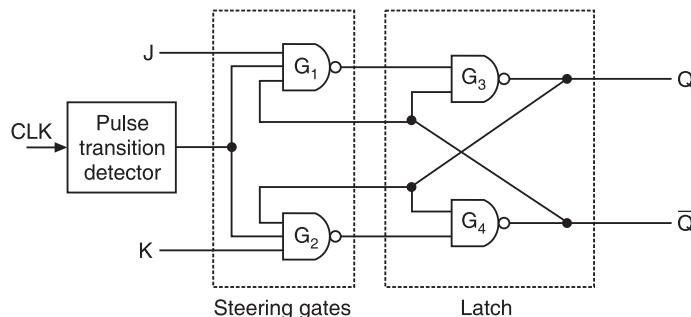
*Internal circuitry of the edge-triggered J-K flip-flop:* A simplified version of the internal circuitry of the edge-triggered J-K flip-flop is shown in Figure 10.19. It contains the same three sections as those of the edge-triggered S-R flip-flop. In fact, the only difference between the two circuits is that, the  $Q$  and  $\bar{Q}$  outputs are fed back to the pulse steering NAND gates. This feedback connection is what gives the J-K flip-flop its toggle operation for  $J = K = 1$  condition.

The toggling operation may be explained as given below:

1. Suppose  $Q = 0$ ,  $\bar{Q} = 1$  and  $J = K = 1$ . When a clock pulse is applied, the narrow positive pulse of the edge detector is inverted by gate  $G_1$ , i.e.  $G_1$  steers the spike inverted (because its other inputs  $J$  and  $\bar{Q}$  are both a 1) to gate  $G_3$ . So, the output of  $G_3$ , i.e.  $Q$  goes HIGH. Since  $Q$  is connected as one input of  $G_2$ , the output of  $G_2$  remains HIGH (because initially  $Q$  was a 0). So,  $G_4$  has both inputs as a 1. Thus, its output  $\bar{Q}$  will be a 0.

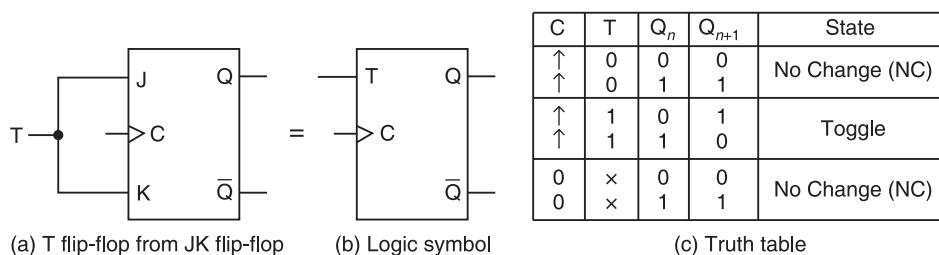
2. Suppose,  $Q = 1$ ,  $\bar{Q} = 0$ , and  $J = K = 1$ . When a clock pulse is applied, the narrow positive spike at  $G_2$  is steered (inverted) to the input of  $G_4$ . So, the output of  $G_4$ , i.e.  $\bar{Q}$  goes HIGH. Since the output of  $G_1$  is HIGH (because initially  $\bar{Q}$  was a 0), both the inputs to  $G_3$  are HIGH. So, the output of  $G_3$ , i.e.  $Q$  goes LOW.

So, we can say that, if a clock pulse is applied when  $J = 1$  and  $K = 1$ , the flip-flop toggles, i.e. it changes its state.



**Figure 10.19** Simplified circuit diagram of the edge-triggered J-K flip-flop.

**The edge-triggered T flip-flop:** A T flip-flop has a single control input, labelled T for toggle. When T is HIGH, the flip-flop toggles on every new clock pulse. When T is LOW, the flip-flop remains in whatever state it was before. Although T flip-flops are not widely available commercially, it is easy to convert a J-K flip-flop to the functional equivalent of a T flip-flop by just connecting J and K together and labelling the common connection as T. Thus, when  $T = 1$ , we have  $J = K = 1$ , and the flip-flop toggles. When  $T = 0$ , we have  $J = K = 0$ , and so there is no change of state. The logic symbol and the truth table of a T flip-flop are shown in Figure 10.20.



**Figure 10.20** Edge-triggered T flip-flop.

#### 10.4.4 Triggering and Characteristic Equations of Flip-Flops

The momentary change in control input of a latch or flip-flop to switch it from one state to the other is called *trigger* and the transition it causes is said to trigger the flip-flop. The process of applying the control signal to change the state of a flip-flop is called *triggering*. There are two types of triggering the flip-flops: *level triggering* and *edge triggering*. In level triggering, the input signals affect the flip-flop only when the clock is at logic 1 level. In edge triggering, the input signals affect the flip-flop only if they are present at the positive going or negative going edge of the clock pulse. Figure 10.21 shows the clock response in the latch and flip-flop.

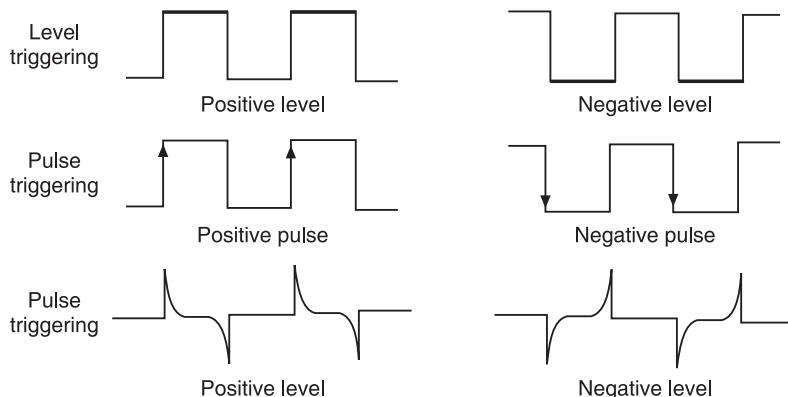


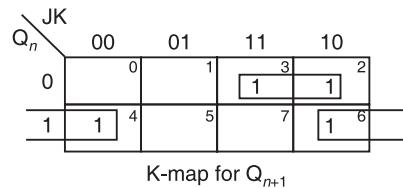
Figure 10.21 Clock response in latch and flip-flop.

A table which lists the present state, the next state and the excitations of a flip-flop is called the excitation table of a flip-flop, i.e. the excitation table is a table which indicates the excitations required to take the flip-flop from the present state to the next state.

The characteristic equation of a flip-flop is the equation expressing the next state of a flip-flop in terms of its present state and present excitations. To obtain the characteristic equation of a flip-flop write the excitation requirements of the flip-flop, draw a K-map for the next state of the flip-flop in terms of its present state and inputs and simplify it as shown in Figures 10.22 (JK FF), 10.23 (SR FF), 10.24 (T FF), and 10.25 (D FF).

(a)	Present state		Inputs		Next state
	$Q_n$		J	K	$Q_{n+1}$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	1
0	1	1	0	0	1
1	0	0	0	0	1
1	0	1	0	0	0
1	1	0	0	0	1
1	1	1	0	0	0

Excitation requirements of JK flip-flop

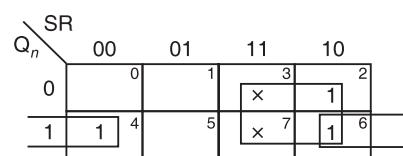


The characteristic equation of a JK flip-flop is

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

(b)	Present state		Inputs		Next state
	$Q_n$		S	R	$Q_{n+1}$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	1
1	0	0	0	0	1
1	0	1	0	0	0
1	1	0	0	0	1

Excitation requirements of SR flip-flop



The characteristic equation of SR flip-flop

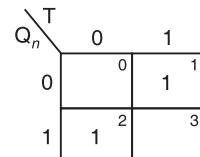
$$Q_{n+1} = S + Q_n \bar{R}$$

Figure 10.23 Excitation requirements and K-map of an S-R flip-flop.

(c)

Present state $Q_n$	Input T	Next state $Q_{n+1}$
0	0	0
0	1	1
1	0	1
1	1	0

Excitation requirements of T flip-flop

K-map for  $Q_{n+1}$  of T flip-flop

The characteristic equation of T flip-flop is

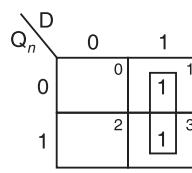
$$Q_{n+1} = \bar{Q}_n T + Q_n \bar{T}$$

Figure 10.24 Excitation requirements and K-map of a T flip-flop.

(d)

Present state $Q_n$	Input D	Next state $Q_{n+1}$
0	0	0
0	1	1
1	0	0
1	1	1

Excitation requirements of D flip-flop

K-map for  $Q_{n+1}$  of D flip-flop

The characteristic equation of D flip-flop is  $Q_{n+1} = D$

Figure 10.25 Excitation requirements and K-map of a D flip-flop.

The characteristic equations of flip-flops are listed in Table 10.3.

Table 10.3 Characteristic equations of flip-flops

Flip-flop	Characteristic equation
D	$Q_{n+1} = D$
J-K	$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$
T	$Q_{n+1} = T\bar{Q}_n + \bar{T}Q_n$
S-R	$Q_{n+1} = S + \bar{R}Q_n$

## 10.5 ASYNCHRONOUS INPUTS

For the clocked flip-flops just discussed, the S-R, D and J-K inputs are called synchronous inputs, because their effect on the flip-flop output is synchronized with the clock input. These synchronous control inputs must be used in conjunction with a clock signal to trigger the flip-flop.

Most IC flip-flops also have one or more asynchronous inputs. These asynchronous inputs affect the flip-flop output independently of the synchronous inputs and the clock input. These asynchronous inputs can be used to SET the flip-flop to the 1 state or RESET the flip-flop to the 0 state at any time regardless of the conditions at the other inputs. In other words, we can say that the asynchronous inputs are the override inputs, which can be used to override all the other inputs in order to place the flip-flop in one state or the other. They are normally labelled PRESET (PRE) or direct SET ( $S_D$ ) or DC SET, and CLEAR (CLR) or direct RESET ( $R_D$ ) or DC CLEAR. An active level on the PRESET input will SET the flip-flop and an active level on the CLEAR input will RESET it. If the asynchronous inputs are active-LOW, the same is indicated by a small bubble at

the input terminals. The inputs in that case are labelled  $\overline{\text{PRE}}$  and  $\overline{\text{CLR}}$  or  $\overline{S_D}$  and  $\overline{R_D}$ . Most IC flip-flops have both DC SET and DC CLEAR inputs. Some have only DC CLEAR. Some have active-HIGH inputs, and some have active-LOW inputs.

It is important to realize that these asynchronous inputs respond to the DC levels. That means, in the case of active-HIGH (LOW) inputs, if a constant 1 (0) is held on the PRE ( $\overline{\text{PRE}}$ ) input, the flip-flop will remain in the  $Q = 1$  state regardless of what is occurring at the other inputs. Similarly, if a constant 1 (0) is held on the CLR ( $\overline{\text{CLR}}$ ) input, the flip-flop will remain in the  $Q = 0$  state regardless of what is occurring at the other inputs. Most often, however, the asynchronous inputs are used to SET or CLEAR the flip-flop to the desired state by the application of a momentary pulse. When DC SET and DC RESET conditions are not used in any application, they must be held at their inactive levels.

The logic symbol and the truth table of a J-K flip-flop with active-LOW PRESET and CLEAR inputs are shown in Figure 10.26. In this case, both PRESET and CLEAR inputs must be kept HIGH for synchronous operation.

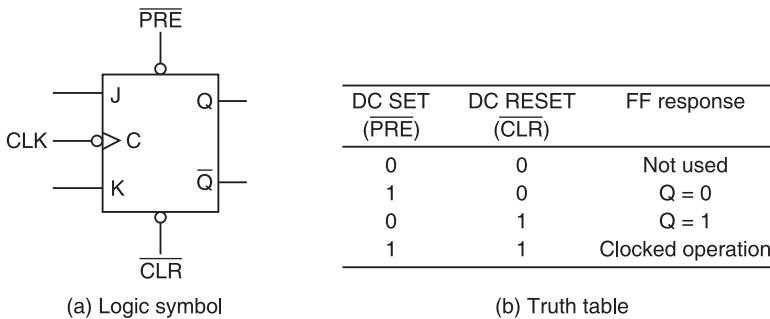


Figure 10.26 J-K flip-flop with active-LOW PRESET and CLEAR inputs.

The operation is discussed as follows:

1.  $\overline{\text{PRE}} = 1$ ,  $\overline{\text{CLR}} = 1$ , i.e. DC SET = 1 and DC CLEAR = 1. The asynchronous inputs are inactive and the flip-flop responds freely to J, K and CLK inputs in the normal way. In other words, the clocked operation can take place.
2.  $\overline{\text{PRE}} = 0$ ,  $\overline{\text{CLR}} = 1$ , i.e. DC SET = 0 and DC CLEAR = 1. The DC SET is activated and Q is immediately SET to a 1, no matter what conditions are present at the J, K and CLK inputs. The CLK input cannot affect the flip-flop while DC SET = 0.
3.  $\overline{\text{PRE}} = 1$ ,  $\overline{\text{CLR}} = 0$ , i.e. DC SET = 1 and DC CLEAR = 0. The DC CLEAR is activated and Q is immediately cleared to a 0 independent of the conditions on the J, K or CLK inputs. The CLK input has no effect when DC CLEAR = 0.
4.  $\overline{\text{PRE}} = 0$ ,  $\overline{\text{CLR}} = 0$ , i.e. DC SET = DC CLEAR = 0. This condition should not be used, since it can result in an invalid state.

Figure 10.27 shows the logic symbol and the truth table of a negative edge triggered J-K flip-flop with active-HIGH PRESET and CLEAR. Figure 10.28 shows the logic diagram for an edge-triggered J-K flip-flop with PRESET ( $\overline{\text{PRE}}$ ) and CLEAR ( $\overline{\text{CLR}}$ ) inputs. As shown in the figure, these inputs are connected directly into the latch portion of the flip-flop so that they override the effect of the synchronous inputs J, K and the CLK.

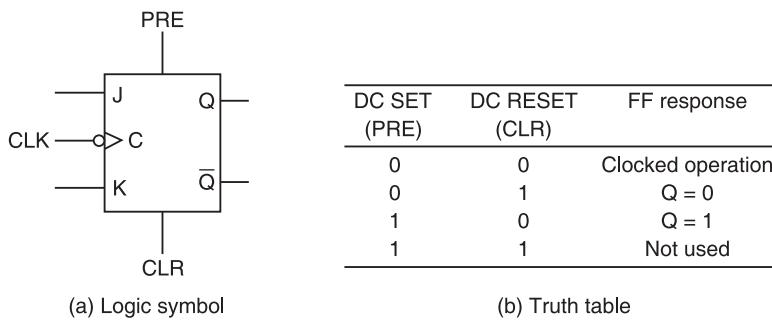


Figure 10.27 J-K flip-flop with active-HIGH PRESET and CLEAR inputs.

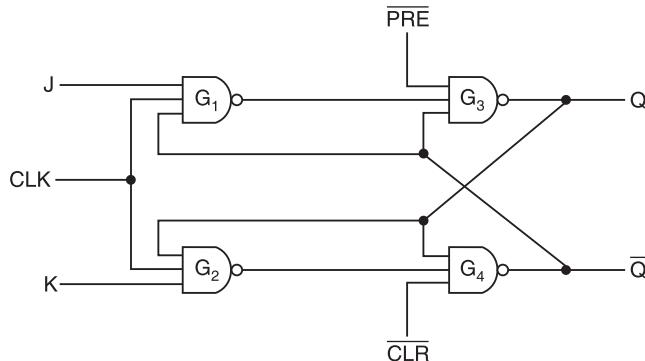


Figure 10.28 Logic diagram of a basic J-K flip-flop with active-LOW PRESET and CLEAR.

**EXAMPLE 10.4** The waveforms shown in Figure 10.29a are applied to the J-K flip-flop shown in Figure 10.29b. Draw the output waveform.

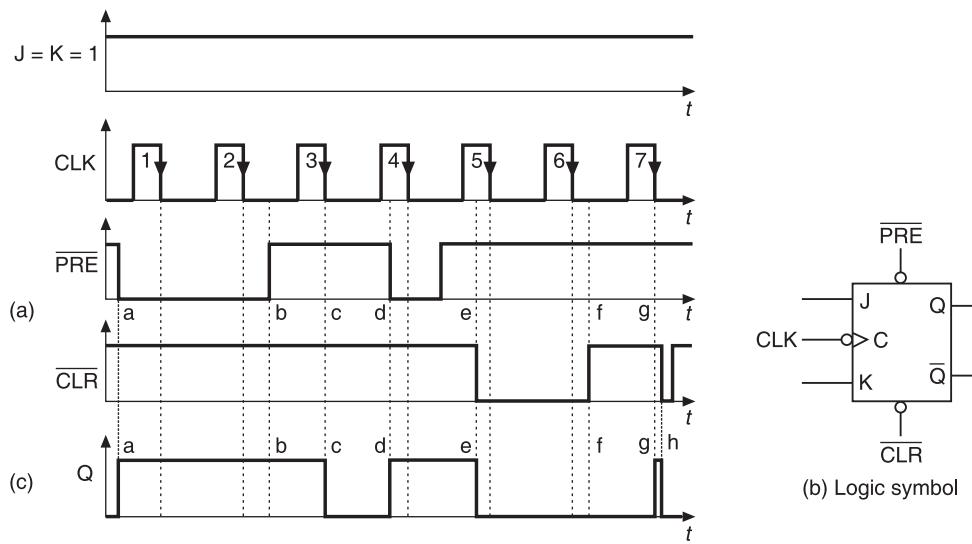


Figure 10.29 Example 10.4: Waveforms—J-K flip-flop.

**Solution**

The output waveform shown in Figure 10.29c is drawn as explained below:

1. Initially  $\overline{\text{PRE}}$  and  $\overline{\text{CLR}}$  are both a 1, and Q is LOW.
2. At the instant a,  $\overline{\text{PRE}}$  goes LOW. So, Q is SET to a 1, and remains SET up to b, because  $\overline{\text{PRE}}$  is kept LOW up to b. From b to c also it remains at a 1, because both  $\overline{\text{PRE}}$  and  $\overline{\text{CLR}}$  are a 1 during this period.
3. Since the flip-flop is in the clocked mode (i.e.  $\overline{\text{PRE}} = 1$  and  $\overline{\text{CLR}} = 1$ ) and since J and K are both a 1, the flip-flop toggles and goes to a 0 at the negative-going edge of the third clock pulse at c.
4. At d,  $\overline{\text{PRE}}$  goes LOW. So, Q is SET to a 1 and remains SET till e.
5. At e,  $\overline{\text{CLR}}$  goes LOW. So, Q is RESET to a 0 and remains RESET till f.
6. After f, Q toggles and goes to a 1 at g at the negative-going edge of the seventh clock pulse.
7. At h,  $\overline{\text{CLR}}$  goes LOW. So, Q also goes LOW.

## 10.6 FLIP-FLOP OPERATING CHARACTERISTICS

Manufacturers of IC flip-flops specify several important characteristics and timing parameters that must be considered before a flip-flop is used in any circuit application. They are typically found in the data sheets for ICs, and they are applicable to all flip-flops regardless of the particular form of the circuit.

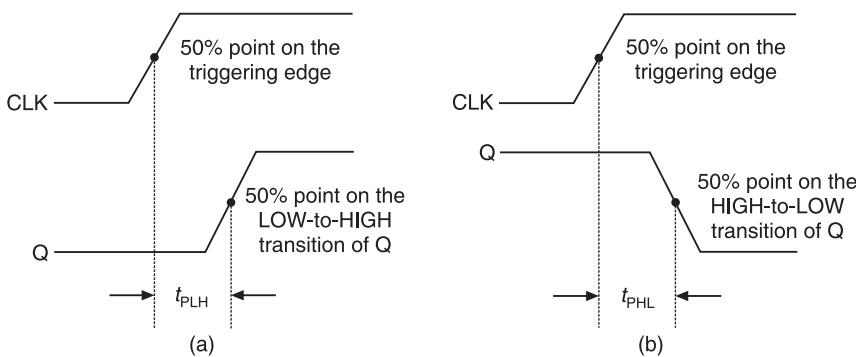
**Propagation delay time:** The output of a flip-flop will not change state immediately after the application of the clock signal or asynchronous inputs. The time interval between the time of application of the triggering edge or asynchronous inputs and the time at which the output actually makes a transition is called the *propagation delay time* of the flip-flop. It is usually in the range of a few ns to 1  $\mu$ s. Several categories of propagation delay are important in the operation of a flip-flop. The propagation delays that occur in response to a positive transition on the clock input are illustrated in Figure 10.30. They are:

1. Propagation delay  $t_{PLH}$  measured from the triggering of the clock pulse to the LOW-to-HIGH transition of the output (shown in Figure 10.30a).
2. Propagation delay  $t_{PHL}$  measured from the triggering of the clock pulse to the HIGH-to-LOW transition of the output (shown in Figure 10.30b).

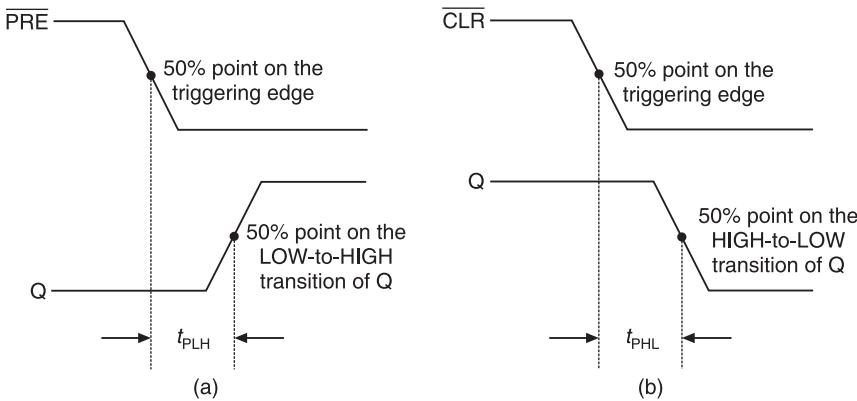
The propagation delays that occur in response to signals on a flip-flop's asynchronous inputs (PRESET and CLEAR) are illustrated in Figure 10.31. They are:

1. Propagation delay  $t_{PLH}$  measured from the PRESET input to the LOW-to-HIGH transition of the output. Figure 10.31a illustrates this delay for active-LOW PRESET.
2. Propagation delay  $t_{PHL}$  measured from the CLEAR input to the HIGH-to-LOW transition of the output. Figure 10.31b illustrates this delay for active-LOW CLEAR.

Note that these delays are measured between the 50% points on the input and output waveforms. The propagation delays  $t_{PLH}$  and  $t_{PHL}$  are usually in the range of a few ns to 1  $\mu$ s. They increase in direct proportion to the number of loads being driven by the Q output.



**Figure 10.30** Propagation delays  $t_{PLH}$  and  $t_{PHL}$  w.r.t. CLK.



**Figure 10.31** Propagation delays  $t_{PLH}$  and  $t_{PHL}$  w.r.t. PRESET and CLEAR.

**Set-up time:** The set-up time ( $t_s$ ) is the minimum time for which the control levels need to be maintained constant on the input terminals of the flip-flop, prior to the arrival of the triggering edge of the clock pulse, in order to enable the flip-flop to respond reliably. Figure 10.32a illustrates the set-up time for a D flip-flop.

**Hold time:** The hold time ( $t_h$ ) is the minimum time for which the control signals need to be maintained constant at the input terminals of the flip-flop, after the arrival of the triggering edge of the clock pulse, in order to enable the flip-flop to respond reliably. Figure 10.32b illustrates the hold time for a D flip-flop.

**Maximum clock frequency:** The maximum clock frequency ( $f_{MAX}$ ) is the highest frequency at which a flip-flop can be reliably triggered. If the clock frequency is above this maximum, the flip-flop would be unable to respond quickly enough and its operation will be unreliable. The  $f_{MAX}$  limit will vary from one flip-flop to another.

**Pulse widths:** The manufacturer usually specifies the minimum pulse widths for the clock and asynchronous inputs. For the clock signal, the minimum HIGH time  $t_w(H)$  and the minimum LOW time  $t_w(L)$  are specified and for asynchronous inputs, i.e. PRESET and CLEAR, the minimum active state time is specified. Failure to meet these minimum time requirements can result in unreliable operation. Figure 10.33 shows pulse widths for CLK and asynchronous inputs.

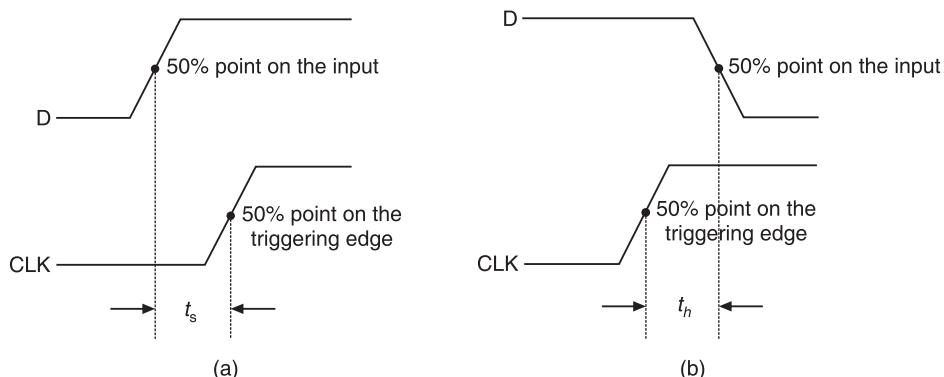


Figure 10.32 Set-up time and hold time for a D flip-flop.

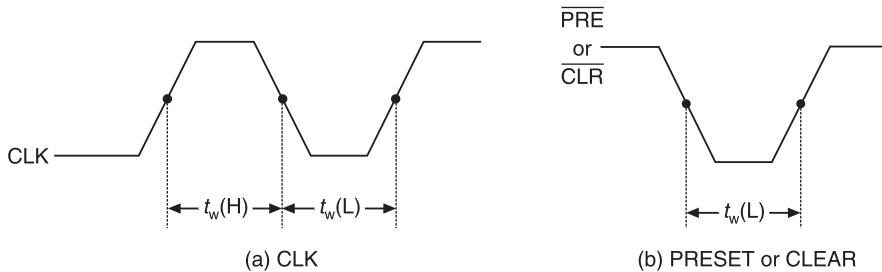


Figure 10.33 Minimum pulse widths.

**Clock transition times:** For reliable triggering, the clock waveform transition times (rise and fall times) should be kept very short. If the clock signal takes too long to make the transitions from one level to the other, the flip-flop may either trigger erratically or not trigger at all.

**Power dissipation:** The power dissipation of a flip-flop is the total power consumption of the device. It is equal to the product of the supply voltage ( $V_{CC}$ ) and the current ( $I_{CC}$ ) drawn from the supply by it.

$$P = V_{CC} \cdot I_{CC}$$

The power dissipation of a flip-flop is usually in mW.

If a digital system has  $N$  flip-flops and if each flip-flop dissipates  $P$  mW of power, the total power requirement

$$P_{TOT} = N \cdot V_{CC} \cdot I_{CC} = (N \cdot P) \text{ mW}$$

## 10.7 CLOCK SKEW AND TIME RACE

One of the most common timing problems in synchronous circuits is clock skew. In many digital circuits, the output of one flip-flop is connected either directly or through logic gates to the input of another flip-flop, and both flip-flops are triggered by the same clock signal. The propagation delay of a flip-flop and/or the delays of the intervening gates make it difficult to predict precisely when the changing state of one flip-flop will be experienced at the input of another.

The clock signal which is applied simultaneously to all the flip-flops in a synchronous system may undergo varying degrees of delay caused by wiring between components, and arrive at the

CLK inputs of different flip-flops at different times. This delay is called *clock skew*. If the clock skew is minimal, a flip-flop may get clocked before it receives a new input (derived from the output of another clocked flip-flop). On the other hand, if the clock pulse is delayed significantly, the inputs to a flip-flop may have changed before the clock pulse arrives. In these situations, we have a kind of a *race* between the two competing signals that are attempting to accomplish opposite effects. This can be termed *time race*. The *winner* in such a race depends largely on unpredictable propagation delays—delays that can vary from one device to another and that can change with environmental conditions. It is clear that reliable system operation is not possible when the responses of a flip-flop depend on the outcome of a race.

### 10.7.1 Potential Timing Problem in Flip-Flop Circuits

A typical situation where this type of potential timing problem occurs is illustrated in Figure 10.34, where the output of the first flip-flop  $Q_1$  is connected to the S input of the second flip-flop and both the flip-flops are clocked by the same signal at their CLK inputs.

The potential timing problem is like this: Since  $Q_1$  will change on the positive-going transition of the clock pulse, the  $S_2$  input of the second flip-flop will be in a changing state as it receives the same positive-going transition. This could lead to an unpredictable response at  $Q_2$ .

Let us assume that, initially  $Q_1 = 1$  and  $Q_2 = 0$ . Let  $FF_1$  has  $S_1 = 0$ ,  $R_1 = 1$  and  $FF_2$  has  $S_2 = 1$ ,  $R_2 = 0$  prior to the positive-going transition of the clock pulse. When the positive-going transition occurs,  $Q_1$  will go to the LOW state, but cannot actually go LOW until after the propagation delay  $t_{PHL}$ . The same positive-going transition will reliably clock  $FF_2$  to the HIGH state, provided that  $t_{PHL}$  is greater than the  $FF_2$ 's hold time requirement  $t_h$ . If this condition is not met, the response of  $FF_2$  will be unpredictable.

Fortunately, all modern edge-triggered flip-flops have hold time requirements that are 5 ns or less; most have  $t_h = 0$  which means that they have no hold time requirements. So, we can say that:

*The flip-flop output will go to a state determined by the logic levels present at its synchronous control inputs just prior to the active clock transition.*

If we apply this rule to Figure 10.34, it says that  $Q_2$  of  $FF_2$  will go to a state determined by  $S_2 = 1$  and  $R_2 = 0$ , a condition that is present just prior to the positive-going transition of the clock pulse. The fact that  $S_2$  is changing in response to the same positive-going transition has no effect on  $Q_2$ .

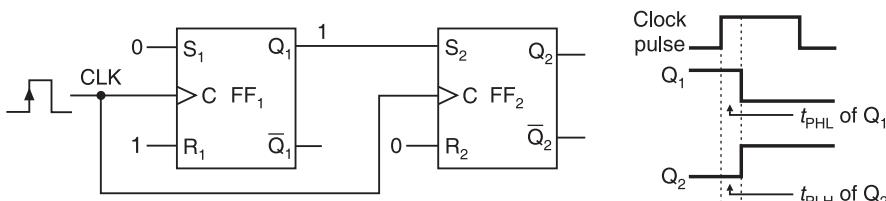


Figure 10.34 Illustration of timing problem.

## 10.8 RACE AROUND CONDITION

For the J-K flip-flop shown in Figure 10.35 consider the assignment of excitations  $J = K = 1$ . If the width of the clock pulse  $t_p$  is too long, the state of the flip-flop will keep on changing from 0 to 1,

1 to 0, 0 to 1 and so on, and at the end of the clock pulse, its state will be uncertain. This phenomenon is called the *race around condition*. The outputs  $Q$  and  $\bar{Q}$  will change on their own if the clock pulse width  $t_p$  is too long compared with the propagation delay  $\tau$  of each NAND gate. Table 10.4 shows how  $Q$  and  $\bar{Q}$  keep on changing with time if the clock pulse width  $t_p$  is greater than  $\tau$ . Assuming that the clock pulse occurs at  $t = 0$ , and  $t_p \gg \tau$ , the following expressions hold before and during  $t_p$  in the J-K flip-flop of Figure 10.35. Note that  $f(t - \tau)$  is  $f(t)$  delayed by  $\tau$ .

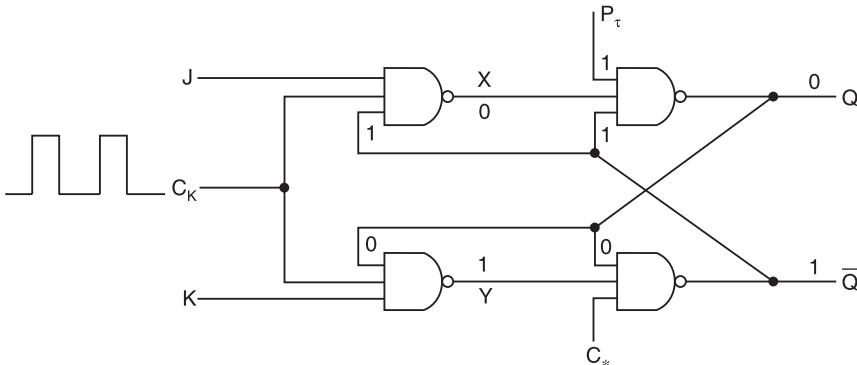


Figure 10.35 J-K flip-flop using NAND gates.

During no pulse

$$\begin{cases} X(t) = 1 \\ Y(t) = 1 \end{cases} \longrightarrow \text{No change in } Q \text{ and } \bar{Q}$$

During the pulse of width  $t_p$  with  $J = K = 1$ ,

$$\begin{aligned} X(t) &= \overline{J \cdot C_K \cdot \overline{Q(t - \tau)}} = \overline{\overline{Q(t - \tau)}}, & Q(t) &= \overline{X(t - \tau) \overline{Q(t - \tau)}} \\ Y(t) &= \overline{K \cdot C_K \cdot Q(t - \tau)} = \overline{Q(t - \tau)}, & \bar{Q}(t) &= \overline{Y(t - \tau) \cdot Q(t - \tau)} \end{aligned}$$

The transitions for  $t_p > \tau$  are shown in Table 10.4. Observe that the change of state takes at least  $2\tau$ . If the flip-flop is initially at  $Q\bar{Q} = 01$ , then it is easily seen that the transition will follow the sequence of logic levels for each  $\tau$  as given below.

$$Q\bar{Q} = 01 \rightarrow 01 \underbrace{\rightarrow 11 \rightarrow 10}_{\text{arrow}} \rightarrow 11 \rightarrow 01 \dots$$

If initially the flip-flop is in  $Q\bar{Q} = 10$ , the transitions will take the following path.

$$Q\bar{Q} = 10 \rightarrow 10 \underbrace{\rightarrow 11 \rightarrow 01}_{\text{arrow}} \rightarrow 11 \rightarrow 10 \dots$$

We thus conclude that the state of the flip-flop keeps on complementing itself for every  $2\tau$ . The clock pulse width should be such as to allow only one change to complement the state and not too long to allow many changes resulting in uncertainty about the final state. This is a stringent requirement which cannot be ensured in practice. This problem is eliminated using master-slave flip-flop or edge triggered flip-flop.

Let  $\tau$  be the propagation delay of the NAND gate. Follow the transition indicated.

$X$  becomes the complement of previous  $\bar{Q}$ .

$Y$  becomes the complement of previous  $Q$ .

$Q$  is the complement of the product of  $X$  and  $\bar{Q}$  of the previous row.

$\bar{Q}$  is the complement of the product of  $Y$  and  $Q$  of the previous row.

**Table 10.4** Flow of signals in race around condition

Time	X	Y	Q	$\bar{Q}$
Initial				
$t < 0$	1	1	0	1
$t \geq 0$				
$t = \tau$	0	1	0	1
$t = 2\tau$	0	1	1	1
$t = 3\tau$	0	0	1	0
$t = 4\tau$	1	0	1	1
$t = 5\tau$	0	0	0	1
$t = 6\tau$	0	1	1	1
$t = 7\tau$	0	0	1	0
$t = 8\tau$	1	0	1	1

Initial state assumed  
Pulse is present for  $t > \tau$

## 10.9 MASTER-SLAVE (PULSE-TRIGGERED) FLIP-FLOPS

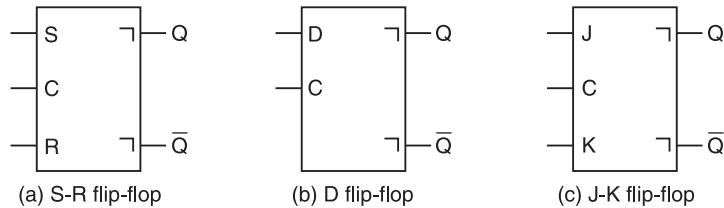
Before the development of edge-triggered flip-flops with little or no hold time requirements, the timing problems such as those shown in Figure 10.34 were often handled by a class of flip-flops called the *master-slave flip-flops*.

The master-slave flip-flop was developed to make the synchronous operation more predictable, that is, to avoid the problems of logic race in clocked flip-flops. This improvement is achieved by introducing a known time delay (equal to the width of one clock pulse) between the time that the flip-flop responds to a clock pulse and the time the response appears at its output. A master-slave flip-flop is also called a *pulse-triggered flip-flop* because the length of the time required for its output to change state equals the width of one clock pulse.

The master-slave or pulse-triggered flip-flop actually contains two flip-flops—a master flip-flop and a slave flip-flop. The control inputs are applied to the master flip-flop and maintained constant for the set-up time  $t_s$  prior to the application of the clock pulse. On the rising edge of the clock pulse, the levels on the control inputs are used to determine the output of the master. On the falling edge of the clock pulse, the state of the master is transferred to the slave, whose outputs are  $Q$  and  $\bar{Q}$ . Thus, the actual outputs of the flip-flop, i.e.  $Q$  and  $\bar{Q}$  change just after the negative-going transition of the clock. These master-slave flip-flops function very much like the negative edge-triggered flip-flops except for one major disadvantage. The control inputs must be held stable while CLK is HIGH, otherwise an unpredictable operation may occur. This problem with the master-slave flip-flop is overcome with an improved master-slave version called the *master-slave with data lock-out*.

There are three basic types of master-slave flip-flops—S-R, D, and J-K. The J-K is by far the most commonly available in IC form. Figure 10.36 shows the logic symbols. The key to identifying

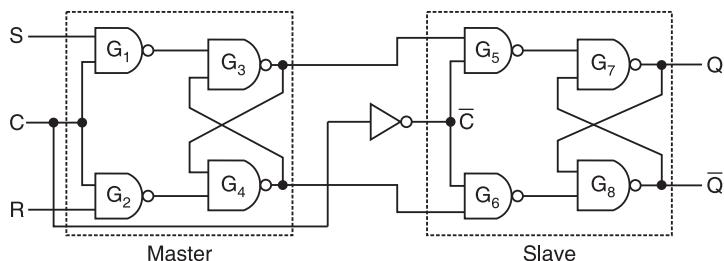
a master-slave flip-flop by its logic symbol is the postponed output symbol  $\lceil$  at the outputs. Note that there is no dynamic input indicator at the clock input.



**Figure 10.36** Logic symbols of master-slave flip-flops.

### 10.9.1 The Master-Slave (Pulse-Triggered) S-R Flip-Flop

Figure 10.37 shows the logic diagram and the truth table of a master-slave, S-R flip-flop. The truth table operation is the same as that for the edge-triggered S-R flip-flop except for the way it is clocked—internally though the master-slave type is quite different. The external control inputs S and R are applied to the master section. The master section is basically a gated S-R latch, and responds to the external S-R inputs applied to it at the positive-going edge of the clock signal. The slave section is the same as the master section except that it is clocked on the inverted clock pulse and thus responds to its control inputs (which are nothing but the outputs of the master flip-flop) at the negative-going edge of the clock pulse. Thus, the master section assumes the state determined by the S and R inputs at the positive-going edge of the clock pulse and the slave section copies the state of the master section at the negative-going edge of the clock pulse. The state of the slave then immediately appears on its Q and  $\bar{Q}$  outputs.



(a) Logic diagram

Inputs	Output			Comments
	S	R	CLK	
0	0			$Q_0$ No change
0	1			0 RESET
1	0			1 SET
1	1			? Invalid

(b) Truth table

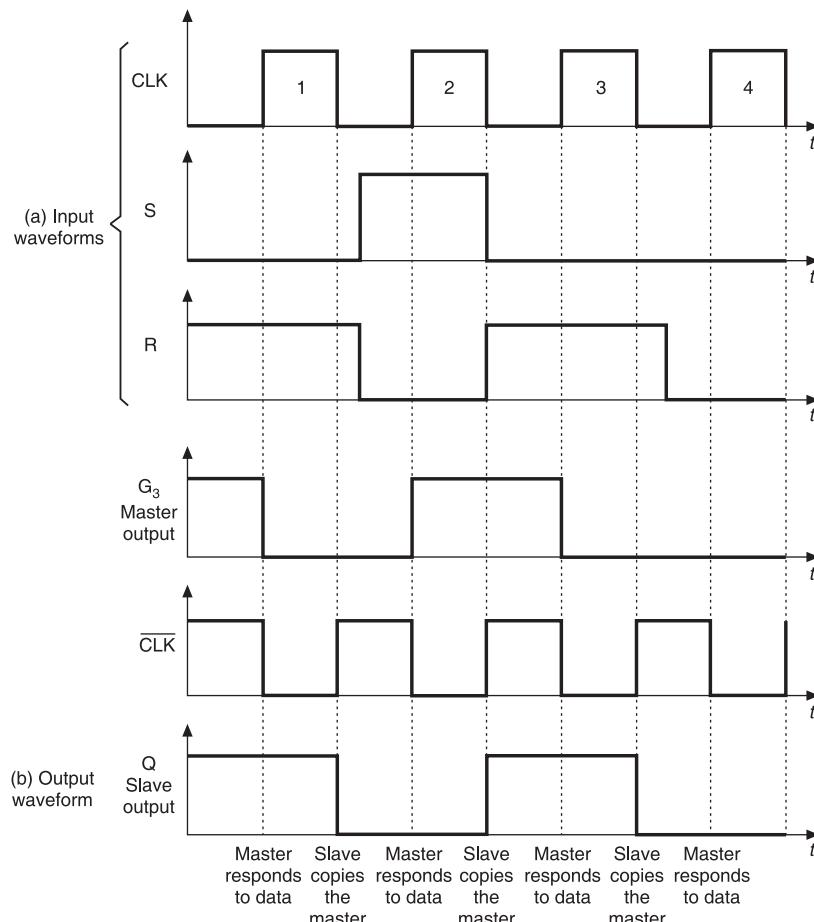
**Figure 10.37** The master-slave S-R flip-flop.

**EXAMPLE 10.5** The waveforms shown in Figure 10.38a are applied to the master-slave S-R flip-flop shown in Figure 10.37a. Draw the output waveform.

**Solution**

Let us assume that, initially the flip-flop is SET, i.e.  $Q = 1$  and the control inputs are  $S = 0$  and  $R = 1$  and the output of master is a 1.

At the positive-going edge of the first clock pulse, the master resets, i.e. the output of  $G_3$  goes LOW. At the negative-going edge of the first clock pulse, the slave copies it. So,  $Q$  goes LOW. The inputs  $S$  and  $R$  now change when the clock is LOW; so it does not affect the operation. At the positive-going edge of the second clock pulse,  $S = 1$  and  $R = 0$  (and  $G_3 = 0$ ,  $Q = 0$ ). So, the master sets, i.e. the output of  $G_3$  goes HIGH. At the negative-going edge of the second clock pulse, the slave copies this action of the master and, therefore,  $Q$  goes HIGH.



**Figure 10.38** Example 10.5: Waveforms—master-slave flip-flop.

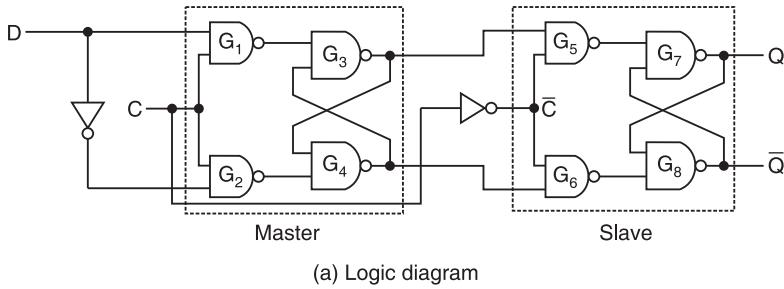
At the positive-going edge of the third clock pulse,  $S = 0$  and  $R = 1$ , so, the master resets, i.e. the output of  $G_3$  goes LOW. At the negative-going edge of the third clock pulse, the slave copies this action of the master and, therefore,  $Q$  goes LOW.

At the positive-going edge of the fourth clock pulse,  $S = 0$  and  $R = 0$  (the output of  $G_3 = 0$ ,  $Q = 0$ ). So, there is no change in the state of the master. Hence, there will not be any change in the state of the slave at the negative-going edge of that clock pulse and  $Q$ , therefore, remains LOW.

The output waveform is shown in Figure 10.38b.

### 10.9.2 The Master-Slave (Pulse-Triggered) D Flip-Flop

The truth table operation of the master-slave D flip-flop shown in Figure 10.39 is the same as that of the negative edge-triggered D flip-flop except for the way it is triggered. The D input is transferred to the master at the positive-going edge of the clock pulse and the same is copied by the slave and, therefore, appears at the Q output of the slave at the negative-going edge of the clock pulse. The truth table of the master-slave D flip-flop is shown in Figure 10.39.



(a) Logic diagram

Inputs	Output	Comments
D	CLK	Q
0	↑↓	0
1	↑↓	1

(b) Truth table

Figure 10.39 The master-slave D flip-flop.

### 10.9.3 The Master-Slave (Pulse-Triggered) J-K Flip-Flop

Figure 10.40 shows the logic diagram and the truth table of a master-slave J-K flip-flop. The truth table operation is the same as that of a negative edge-triggered J-K flip-flop except for the way in which it is triggered. The logic diagram of the master-slave J-K flip-flop is similar to that of the master-slave S-R flip-flop. The difference is that the Q output is connected back to the input of  $G_2$  and the  $\bar{Q}$  output is connected back to the input of  $G_1$  and the external inputs are designated as J and K. The M-S J-K flip-flop is a J-K flip-flop followed by an S-R flip-flop, i.e. the master flip-flop is a J-K flip-flop and slave flip-flop is an S-R flip-flop. The problem of logic race is eliminated because while the clock drives the master, the inverted clock drives the slave.

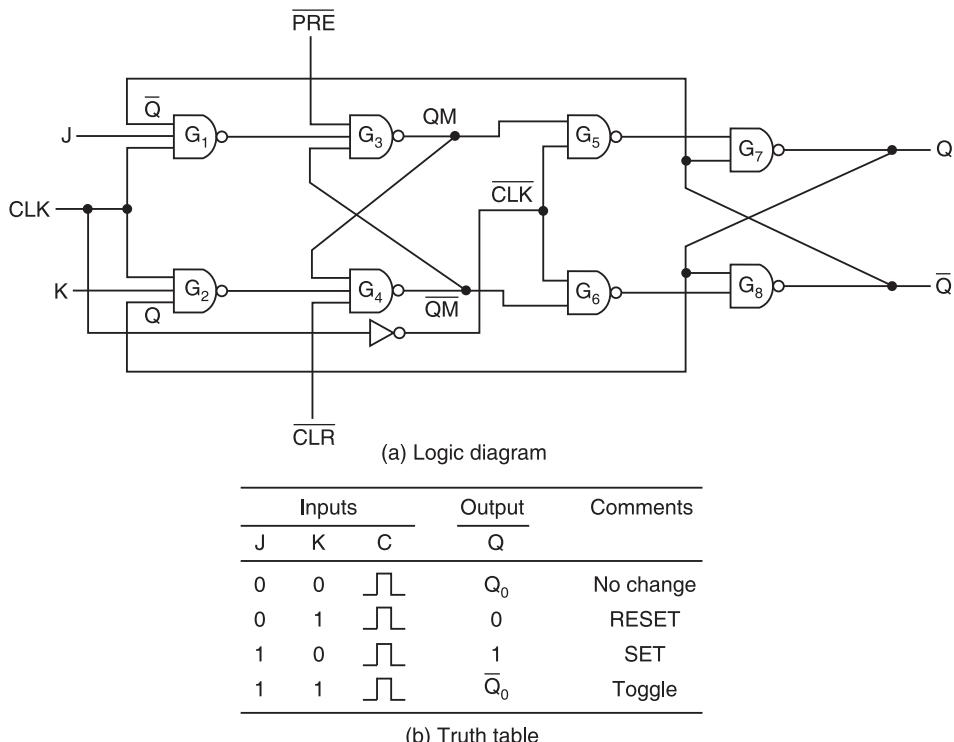


Figure 10.40 The master-slave J-K flip-flop.

**EXAMPLE 10.6** The waveforms shown in Figure 10.41a are applied to the master-slave J-K flip-flop shown in Figure 10.41b. Draw the output waveform.

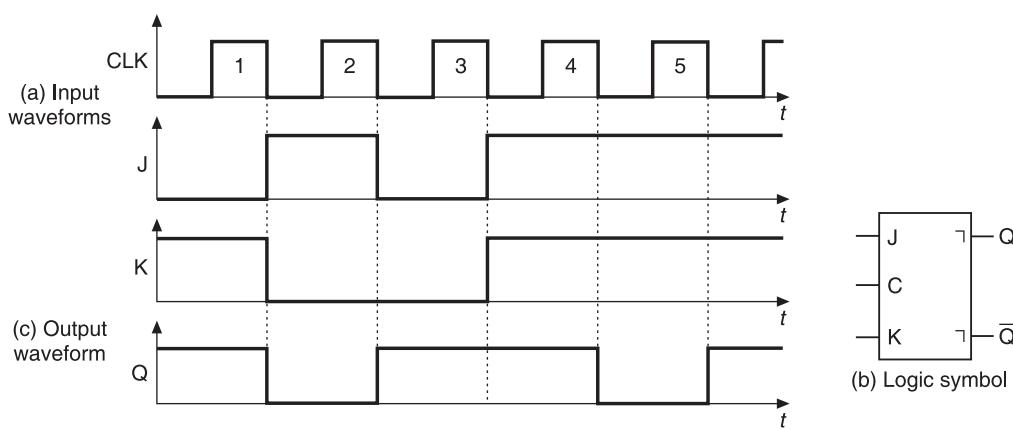


Figure 10.41 Example 10.6: Waveforms—master-slave J-K flip-flop.

### Solution

The output waveform shown in Figure 10.41c is drawn as explained below.

Initially,  $J = 0$  and  $K = 1$  and the flip-flop is assumed to be in SET state, i.e.  $Q = 1$ .

At the positive-going edge of the first clock pulse,  $J = 0$  and  $K = 1$ . So, the flip-flop resets, and  $Q$  goes LOW at the negative-going edge of this clock pulse.

At the positive-going edge of the second clock pulse,  $J = 1$  and  $K = 0$ . So, the flip-flop sets, and  $Q$  goes HIGH at the negative-going edge of this clock pulse.

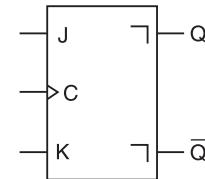
At the positive-going edge of the third clock pulse,  $J = 0$  and  $K = 0$ . So, there will not be any change in the state of the flip-flop at the negative-going edge. Thus, the flip-flop remains SET. Hence  $Q$  remains HIGH. There afterwards, both  $J$  and  $K$  remain HIGH. So, the flip-flop will be in toggle mode. Hence,  $Q$  goes to the opposite state at the negative-going edge of each of the subsequent clock pulses.

#### 10.9.4 The Data Lock-out Flip-Flop

Earlier it was mentioned that a severe limitation of the master-slave flip-flop is that the data inputs must be held constant while the clock is HIGH, because it responds to any changes in the data inputs when the clock is HIGH. This problem is overcome in the data lock-out flip-flop.

The data lock-out flip-flop is similar to the master-slave (pulse-triggered) flip-flop except that it has a dynamic clock input, making it sensitive to the data bits only during clock transitions. After the leading edge of a clock transition, the data inputs are disabled and thus not held constant while the clock pulse is HIGH. In essence, the master portion of this flip-flop is like an edge-triggered device and the slave portion performs like the slave in a master-slave device to produce a postponed output.

Figure 10.42 shows the logic symbol for a data lock-out J-K flip-flop. Note that this symbol has both the dynamic input indicator for the clock, and the postponed output indicators. This type of flip-flop is classified by most manufacturers as a master-slave with a special lock-out feature. The master-slave flip-flop has now become obsolete although we may encounter it in older equipment.



**Figure 10.42** Logic symbol of the master-slave J-K flip-flop with data lock-output.

### 10.10 FLIP-FLOP EXCITATION TABLES

For the design of sequential circuits we should know the excitation tables of flip-flops. The excitation table of a flip-flop can be obtained from its truth table. It indicates the inputs required to be applied to the flip-flop to take it from the present state to the next state. The truth tables and excitation tables of various flip-flops are given below.

**S-R flip-flop:** The truth table and excitation table of an S-R flip-flop are given in Tables 10.5a and b.

**Table 10.5a** S-R truth table

S	R	$Q_{n+1}$
0	0	$Q_n$
0	1	0
1	0	1
1	1	?

**Table 10.5b** S-R excitation table

<b>PS</b>	<b>NS</b>	<b>Required inputs</b>	
		S	R
$Q_n$	$Q_{n+1}$		
0	0	0	$\times$
0	1	1	0
1	0	0	1
1	1	$\times$	0

$0 \rightarrow 0$  transition: If the present state of the FF is 0 and if it has to remain 0 when a clock pulse is applied, the inputs can be either  $S = 0, R = 0$  (no change condition) or  $S = 0, R = 1$  (reset condition). Thus, S has to be 0 but R can be either 0 or 1. So  $SR = 0\times$  for this transition.

$0 \rightarrow 1$  transition: If the present state of the FF is 0 and if it has to go to 1 state when a clock pulse is applied, the inputs have to be  $S = 1$  and  $R = 0$  (set condition). So  $SR = 10$  for this transition.

$1 \rightarrow 0$  transition: If the present state of the FF is 1 and if it has to go to 0 state when a clock pulse is applied, the inputs have to be  $S = 0, R = 1$  (reset condition). So  $SR = 01$  for this transition.

$1 \rightarrow 1$  transition: If the present state of the FF is 1 and if it has to remain 1 when a clock pulse is applied, the inputs can be either  $S = 0, R = 0$  (no change condition) or  $S = 1, R = 0$  (set condition). Thus R has to be 0 but S can be either 0 or 1. So  $SR = \times 0$  for this transition.

**J-K flip-flop:** The truth table and excitation table of a J-K flip-flop are shown in Tables 10.6a and b.

**Table 10.6a** J-K truth table

<b>J</b>	<b>K</b>	<b><math>Q_{n+1}</math></b>
0	0	$Q_n$
0	1	0
1	0	1
1	1	$\bar{Q}_n$

**Table 10.6b** J-K excitation table

<b>PS</b>	<b>NS</b>	<b>Required inputs</b>	
		J	K
$Q_n$	$Q_{n+1}$		
0	0	0	$\times$
0	1	1	$\times$
1	0	$\times$	1
1	1	$\times$	0

**0 → 0 transition:** The present state of the FF is 0 and it has to remain 0 after the clock pulse. This can happen with either  $J = 0, K = 0$  (no change condition) or  $J = 0, K = 1$  (reset condition). Thus, J has to be 0 but K can be either 0 or 1. So  $JK = 0x$  for this transition.

**0 → 1 transition:** The present state of the FF is 0 and it has to go to 1 state after the clock pulse. This can happen with either  $J = 1, K = 0$  (set condition) or  $J = 1, K = 1$  (toggle condition). Thus J has to be 1 but K can be either 0 or 1. So  $JK = 1x$  for this transition.

**1 → 0 transition:** The present state of the FF is 1 and it has to go to 0 state after the clock pulse. This can happen with either  $J = 0, K = 1$  (reset condition) or  $J = 1, K = 1$  (toggle condition). Thus K has to be 1 but J can be either 0 or 1. So  $JK = x1$  for this transition.

**1 → 1 transition:** The present state of the FF is 1 and it has to remain in 1 state after the clock pulse. This can happen with either  $J = 0, K = 0$  (no change condition) or  $J = 1, K = 0$  (set condition). Thus K has to be 0 but J can be either 0 or 1. So  $JK = x0$  for this transition.

**D flip-flop:** The truth table and the excitation table of the D flip-flop are shown in Tables 10.7a and b.

**Table 10.7a** D truth table

D	$Q_{n+1}$
0	0
1	1

**Table 10.7b** D excitation table

PS $Q_n$	NS $Q_{n+1}$	Required input
		D
0	0	0
0	1	1
1	0	0
1	1	1

For a D flip-flop, the next state is always equal to the D input and it is independent of the present state. Therefore, D must be 0 if  $Q_{n+1}$  has to be 0, and 1 if  $Q_{n+1}$  has to be 1 regardless of the value of  $Q_n$ .

**T flip-flop:** The truth table and the excitation table of the T flip-flop are shown in Tables 10.8a and b.

**Table 10.8a** T truth table

T	$Q_{n+1}$
0	$Q_n$
1	$\bar{Q}_n$

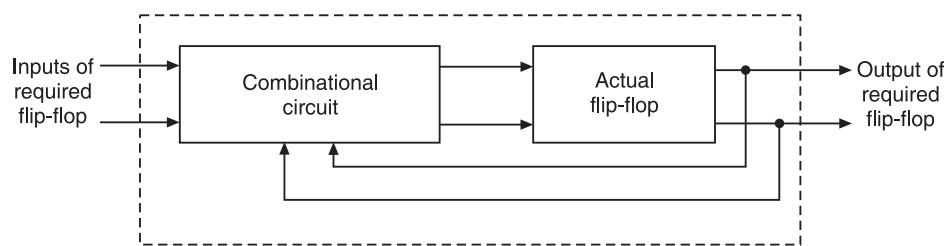
**Table 10.8b** T excitation table

PS $Q_n$	NS $Q_{n+1}$	Required input
		T
0	0	0
0	1	1
1	0	1
1	1	0

For a T flip-flop, when the input  $T = 1$ , the state of the flip-flop is complemented and when  $T = 0$ , the state of the flip-flop remains unchanged. Thus, for  $0 \rightarrow 0$  and  $1 \rightarrow 1$  transitions T must be 0 and for  $0 \rightarrow 1$  and  $1 \rightarrow 0$  transitions T must be 1.

## 10.11 CONVERSION OF FLIP-FLOPS

To convert one type of flip-flop into another type, a combinational circuit is designed such that if the inputs of the required flip-flop (along with the outputs of the actual flip-flop if required) are fed as inputs to the combinational circuit and the output of the combinational circuit is connected to the inputs of the actual flip-flop, then the output of the actual flip-flop is the output of the required flip-flop. In other words, it means that, to convert one type of flip-flop into another type, we have to obtain the expressions for the inputs of the existing flip-flop in terms of the inputs of the required flip-flop and the present state variables of the existing flip-flop and implement them. The arrangement is as shown in Figure 10.43.

**Figure 10.43** Block diagram for conversion of flip-flop.

**S-R flip-flop to J-K flip-flop:** Here the external inputs to the already available S-R flip-flop will be J and K. S and R are the outputs of the combinational circuit, which are also the actual inputs to the S-R flip-flop. We write a truth table with J, K,  $Q_n$ ,  $Q_{n+1}$ , S, and R, where  $Q_n$  is the present state of the flip-flop and  $Q_{n+1}$  is the next state obtained when the particular J and K inputs are applied, i.e.  $Q_n$  denotes the state of the flip-flop before the application of the inputs and  $Q_{n+1}$  refers to the state obtained by the flip-flop after the application of inputs.

J, K and  $Q_n$  can have eight combinations. For each combination of J, K and  $Q_n$ , find the corresponding  $Q_{n+1}$ , i.e. determine to which next state ( $Q_{n+1}$ ) the J-K flip-flop will go from the present state  $Q_n$  if the present inputs J and K are applied. Now complete the table by writing the values of S and R required to get each  $Q_{n+1}$  from the corresponding  $Q_n$ , i.e. write what values of S and R are required to change the state of the flip-flop from  $Q_n$  to  $Q_{n+1}$ .

The conversion table, the K-maps for S and R in terms of J, K and  $Q_n$  and the logic diagram showing the conversion from S-R to J-K are shown in Figure 10.44.

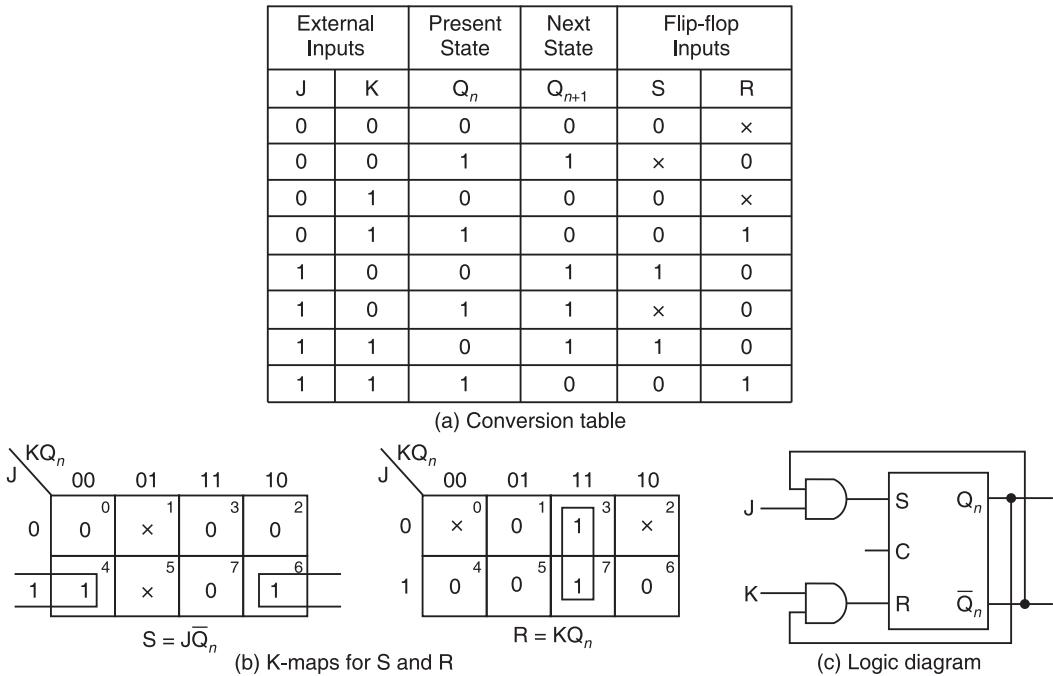


Figure 10.44 Conversion of S-R flip-flop to J-K flip-flop.

**J-K flip-flop to S-R flip-flop:** Here, the external inputs to the already available J-K flip-flop will be S and R. J and K are the outputs of the combinational circuit which are also the actual inputs to the J-K flip-flop. So, we have to get the values of J and K in terms of S, R and  $Q_n$ . Thus, write a table using S, R,  $Q_n$ ,  $Q_{n+1}$ , J, and K. The external inputs S and R and the present output  $Q_n$  can make eight possible combinations. For each combination, find the corresponding next state  $Q_{n+1}$ . In the S-R flip-flop, the combination S = 1 and R = 1 is not permitted. So, the corresponding output is invalid and, therefore, the corresponding J and K are don't cares. Complete the table by writing the values of J and K required to get each  $Q_{n+1}$  from the corresponding  $Q_n$ .

The conversion table, the K-maps for J and K in terms of S, R, and  $Q_n$  and the logic diagram showing the conversion from J-K to S-R are shown in Figure 10.45.

**S-R flip-flop to D flip-flop:** Here S-R flip-flop is available and we want the operation of D flip-flop from it. So D is the external input and the outputs of the combinational circuit are the inputs to the available S-R flip-flop. Express the inputs of the existing flip-flop S and R in terms of the external input D and the present state  $Q_n$ .

The conversion table, the K-maps for S and R in terms of D and  $Q_n$ , and the logic diagram showing the conversion from S-R to D are shown in Figure 10.46.

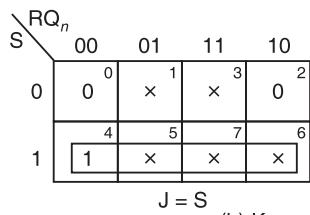
**D flip-flop to S-R flip-flop:** Here D flip-flop is available and we want S-R flip-flop operation from it. So S and R are the external inputs and D is the actual input to the existing flip-flop. S, R

and  $Q_n$  make eight possible combinations, but  $S = R = 1$  is an invalid combination. So, the corresponding entries for  $Q_{n+1}$  and D are don't cares. Express D in terms of S, R and  $Q_n$ .

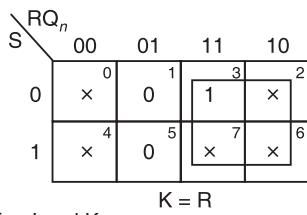
The conversion table, the K-map for D in terms of S, R and  $Q_n$ , and the logic diagram showing the conversion from D to S-R are shown in Figure 10.47.

External Inputs		Present State	Next State	Flip-flop Inputs	
S	R	$Q_n$	$Q_{n+1}$	J	K
0	0	0	0	0	x
0	0	1	1	x	0
0	1	0	0	0	x
0	1	1	0	x	1
1	0	0	1	1	x
1	0	1	1	x	0

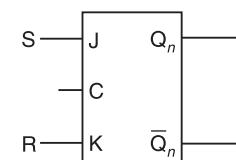
(a) Conversion table



$$J = S$$



$$K = R$$

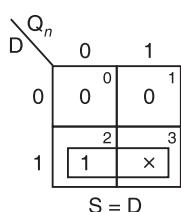


(c) Logic diagram

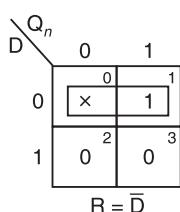
Figure 10.45 Conversion of J-K flip-flop to S-R flip-flop.

External Inputs		Present State	Next State	Flip-flop Inputs	
D		$Q_n$	$Q_{n+1}$	S	R
0		0	0	0	x
0		1	0	0	1
1		0	1	1	0
1		1	1	x	0

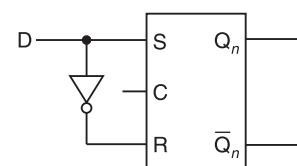
(a) Conversion table



$$S = D$$



$$R = \bar{D}$$

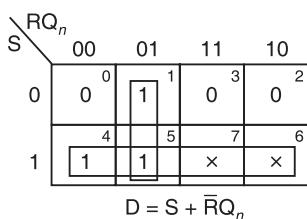


(c) Logic diagram

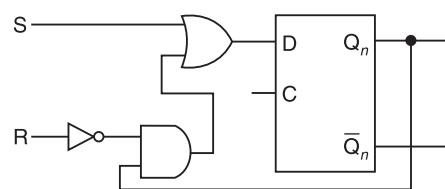
Figure 10.46 Conversion of S-R flip-flop to D flip-flop.

External Inputs		Present State	Next State	Flip-flop Input
S	R	$Q_n$	$Q_{n+1}$	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1

(a) Conversion table



$$(b) K\text{-map for } D$$



(c) Logic diagram

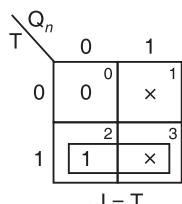
**Figure 10.47** Conversion of D flip-flop to S-R flip-flop.

**J-K flip-flop to T flip-flop:** Here J-K flip-flop is available and we want T flip-flop operation from it. So T is the external input and J and K are the actual inputs to the existing flip-flop. T and  $Q_n$  make four combinations. Express J and K in terms of T and  $Q_n$ .

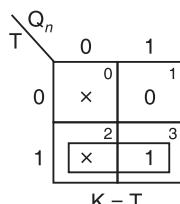
The conversion table, the K-maps for J and K in terms of T and  $Q_n$ , and the logic diagram showing the conversion from J-K to T are shown in Figure 10.48.

External Input	Present State	Next State	Flip-flop Inputs	
T	$Q_n$	$Q_{n+1}$	J	K
0	0	0	0	x
0	1	1	x	0
1	0	1	1	x
1	1	0	x	1

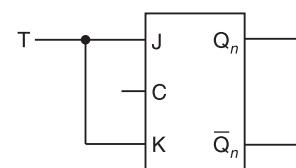
(a) Conversion table



$$J = T$$



$$K = T$$



(c) Logic diagram

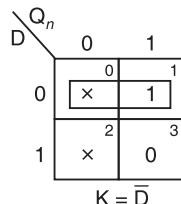
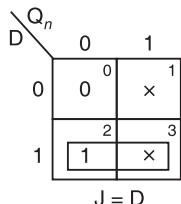
**Figure 10.48** Conversion of J-K flip-flop to T flip-flop.

**J-K flip-flop to D flip-flop:** Here J-K flip-flop is available and we want D flip-flop operation from it. Hence D is the external input and J and K are the actual inputs to the existing flip-flop. D and  $Q_n$  make four combinations. Express the inputs of the existing flip-flop J and K in terms of the external input D and the present state  $Q_n$ .

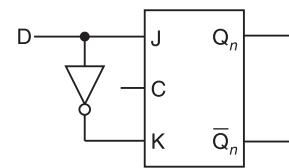
The conversion table, the K-maps for J and K in terms of D and  $Q_n$ , and the logic diagram showing the conversion from J-K to D are shown in Figure 10.49.

External Input	Present State	Next State	Flip-flop Inputs	
D	$Q_n$	$Q_{n+1}$	J	K
0	0	0	0	x
0	1	0	x	1
1	0	1	1	x
1	1	1	x	0

(a) Conversion table



(b) K-maps for J and K



(c) Logic diagram

**Figure 10.49** Conversion of J-K flip-flop to D flip-flop.

**D flip-flop to J-K flip-flop:** Here D flip-flop is available and we want J-K flip-flop operation from it. Hence J and K are the external inputs, i.e. inputs to the combinational circuit and D is the actual input to the existing flip-flop. J, K and  $Q_n$  make eight combinations. Express the input of the existing flip-flop D in terms of external inputs J, K and the present state  $Q_n$ .

The conversion table, the K-map for D in terms of J, K, and  $Q_n$  and the logic diagram showing the conversion from D to J-K are shown in Figure 10.50.

External Inputs		Present State	Next State	Flip-flop Input
J	K	$Q_n$	$Q_{n+1}$	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

(a) Conversion table

**Figure 10.50** Conversion of D flip-flop to J-K flip-flop (Contd.)

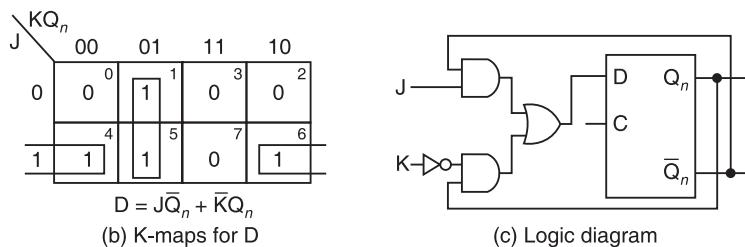


Figure 10.50 Conversion of D flip-flop to J-K flip-flop.

## 10.12 APPLICATIONS OF FLIP-FLOPS

There are a large number of applications of flip-flops. Some of the basic applications are parallel data storage, serial data storage, transfer of data, frequency division, counting, parallel to serial conversion, serial to parallel conversion, synchronizing the effect of asynchronous data, detection of an input sequence, etc. These are discussed in detail as under.

**Parallel data storage:** A group of flip-flops is called a register. To store a data of  $N$  bits,  $N$  flip-flops are required. Since the data is available in parallel form, i.e. all bits are present at a time, these bits may be made available at the D input terminals of the flip-flops, and when a clock pulse is applied to all the flip-flops simultaneously, these bits will be transferred to the Q outputs of the flip-flops and the flip-flops (register) then store the data. The use of flip-flops for parallel data storage is illustrated in Figure 11.1 (see Section 11.2).

**Serial data storage:** To store a data of  $N$  bits available in serial form,  $N$  number of D flip-flops are connected in cascade. The clock signal is connected to all the flip-flops. The serial data is applied to the D input terminal of the first flip-flop. Each clock pulse transfers the D input to its Q output. So, after  $N$  clock pulses the register (group of flip-flops) contains the data and then stores it. The use of flip-flops for serial data storage is illustrated in Figure 11.4 (Section 11.5).

**Transfer of data:** Data stored in flip-flops may be transferred out in a serial fashion, i.e. bit-by-bit from the output of one flip-flop or may be transferred out in parallel form, i.e. all bits at a time from the Q outputs of each of the flip-flops.

**Serial-to-parallel conversion:** To convert the data available in serial form into parallel form, the serial data are first entered and stored in a serial-in parallel-out shift register (a group of flip-flops connected in cascade) and then, since the data is available simultaneously at the outputs of the flip-flops, the data may be taken out parallelly. To convert an  $N$ -bit serial data to parallel form,  $N$  flip-flops are required.  $N$  clock pulses are required to enter the data in serial form and one clock pulse is required to shift the data out in parallel form.

The use of flip-flops for serial-to-parallel conversion is illustrated in Figure 11.7 (Section 11.6).

**Parallel-to-serial conversion:** To convert the data available in parallel form into serial form, the parallel data are first entered in the parallel-in serial-out shift register in parallel form, i.e. all bits at a time and then that data is shifted out of the register serially, i.e. bit-by-bit by the application of clock pulses. To convert an  $N$ -bit parallel data to serial form,  $N$  flip-flops are required. One clock pulse is required to shift the parallel data into the register and  $N$  clock pulses are required to shift the data out of the register serially.

The use of flip-flops for parallel-to-serial conversion is illustrated in Figure 11.8 (Section 11.7).

**Counting:** A number of flip-flops may be connected in a particular fashion to count the pulses electronically. One flip-flop can count up to 2 pulses; two flip-flops can count up to  $2^2 = 4$  pulses. In general,  $N$  flip-flops can count up to  $2^N$  pulses. In a simple counter, all the flip-flops are connected in toggle mode. The clock pulses are applied to the first flip-flop and the clock terminal of each subsequent flip-flop is connected to the Q output of the previous flip-flop. Feedback may be provided if the maximum count required is not  $2^N$ . Flip-flops may be used to count up or down or up/down. Figures 12.1, 12.2, 12.3, 12.4, 12.5, 12.6, etc. illustrate the use of flip-flops for counting.

**Frequency division:** Flip-flops may be used to divide the input signal frequency by any number. A single flip-flop may be used to divide the input frequency by 2. Two flip-flops may be used to divide the input frequency by 4. In general,  $N$  flip-flops may be used to divide the input frequency by  $2^N$ . If  $N$  flip-flops are connected as a ripple counter (a counter in which the external signal is applied to the clock terminal of the first flip-flop and the Q output of each flip-flop is connected to the clock input of next flip-flop) and if the input signal of frequency  $f$  is fed to the first flip-flop, the output of this flip-flop will be of frequency  $f/2$ , the output of the second flip-flop will be of frequency  $f/4$ , and so on.

Figure 12.2a and the waveforms in Figure 12.2b (also Figures 12.7a and 12.7b) illustrate the use of flip-flops for frequency division.

### SHORT QUESTIONS AND ANSWERS

1. Distinguish between combinational and sequential switching circuits.
- A. Basically, switching circuits may be combinational switching circuits or sequential switching circuits. Combinational switching circuits are those whose output levels at any instant of time are dependent only on the levels present at the inputs at that time. Any prior input level conditions have no effect on the present outputs, because combinational logic circuits have no memory. On the other hand, sequential switching circuits are those whose output levels at any instant of time are dependent not only on the levels present at the inputs at that time, but also on the prior input level conditions. It means that sequential circuits have memory. Sequential circuits are thus made of combinational circuits and memory elements.
2. What do you mean by stable state?
- A. The stable state is a state in which the circuit can remain permanently. Only when an external signal is applied, it will change the state.
3. What is a flip-flop?
- A. A flip-flop is the basic memory element used to store one bit of information. It can store a 0 or a 1. The name flip-flop is because this circuit shifts back and forth between its two stable states upon application of proper inputs.. Or we can say a flip-flop is a simple logic circuit.
4. How many flip-flops are required for storing  $n$  bits of information?
- A.  $n$  flip-flops are required for storing  $n$  bits of information. One flip-flop is required for every bit.
5. A flip-flop is known more formally as what? How many stable states does it have?
- A. A flip-flop is known more formally as a bistable multivibrator. It has two stable states.
6. What is the flip-flop's memory characteristic?
- A. A flip-flop input has to be pulsed momentarily to cause a change in the flip-flop output, and the output will remain in that new state even after the input pulse has been removed. This is the flip-flop's memory characteristic.

- 7.** What are the applications of flip-flops?  
**A.** Flip-flops have innumerable applications. They are used for data storage, transfer of data, counting, frequency division, parallel-to-serial and serial-to-parallel data conversion, etc.
- 8.** What do you mean by a latch? Why that name?  
**A.** An unclocked flip-flop is called a latch. This name is because the output of the unclocked flip-flop latches on to a 1 or a 0 immediately after the input is applied.
- 9.** What is an asynchronous latch?  
**A.** Non-gated latches are called asynchronous latches. Here no clock signals are applied and the output latches onto a 0 or a 1 asynchronously any time proper inputs are applied.
- 10.** What is a synchronous latch?  
**A.** Clocked latches are called gated or synchronous latches. Here clock signals are applied and the output latches onto a 0 or a 1 only if the inputs are applied synchronously along with the clock.
- 11.** How do you build a latch using universal gates?  
**A.** A latch may be built by using two cross-coupled NOR gates or NAND gates. By using cross-coupled NOR gates an active-HIGH SR latch can be built and by using cross-coupled NAND gates an active-LOW SR latch can be built.
- 12.** What is an active-HIGH latch? and an active-LOW latch?  
**A.** An active-HIGH latch is one in which the inputs are normally resting in the low state and one of them will be pulsed high whenever we want to change the latch outputs. An active-LOW latch is one in which the inputs are normally resting in the high state and one of them will be pulsed low whenever we want to change the latch outputs.
- 13.** Distinguish between synchronous and asynchronous latches.  
**A.** Synchronous latches are latches which respond to their inputs only when the clock is present. Asynchronous latches are latches which respond to their inputs the moment the inputs are applied. No clock signal is present.
- 14.** What is the normal resting state of SET and CLEAR inputs in a NAND gate SR latch? What is the active state of each input?  
**A.** The normal resting state of SET and CLEAR inputs in a NAND gate SR latch is both the inputs are high. The active state of each input is when the input is low.
- 15.** What is the normal resting state of SET and CLEAR inputs in a NOR gate SR latch? What is the active state of each input?  
**A.** The normal resting state of SET and CLEAR inputs in a NOR gate SR latch is when both the inputs are low. The active state of each input is when the input is high.
- 16.** What is meant by clocked flip-flop?  
**A.** A clocked flip-flop is a flip-flop whose state changes occur (according to the data inputs) only when a clock pulse is present.
- 17.** Name the two types of inputs which a clocked flip-flop has.  
**A.** The two types of inputs which a clocked flip-flop has are (a) control inputs and (b) enable input which may be a clock.
- 18.** Why is a gated D latch called a transparent latch?  
**A.** A gated D latch is called a transparent latch because the Q output follows the D input when ENABLE is high, i.e. when EN is high, a low D input makes Q low, i.e. resets the flip-flop and a high D makes Q high, i.e. sets the flip-flop.
- 19.** What are the two types of flip-flops?  
**A.** The two types of flip-flops are (a) level-triggered flip-flops and (b) edge triggered flip-flops.

- 20.** Distinguish between level-triggered flip-flops and edge-triggered flip-flops.
- A. The level-triggered flip-flops are those which respond to changes in inputs when the clock is high. The edge-triggered flip-flops are those which respond only to inputs present at the transition of the clock pulse.
- 21.** What are the various methods used for triggering flip-flops?
- A. The various methods used for triggering flip-flops are as follows:
- |                              |                              |
|------------------------------|------------------------------|
| (a) Level triggering         | (b) Pulse triggering         |
| (c) Positive edge triggering | (d) Negative edge triggering |
- 22.** What is dynamic triggering?
- A. Dynamic triggering is the other name of edge triggering.
- 23.** Which is the most versatile and most widely used of all the flip-flops?
- A. The JK flip-flop is the most versatile and most widely used of all the flip-flops.
- 24.** Explain the operation of a JK flip-flop.
- A. (a) When  $J = K = 0$ , the outputs are not affected by the clock pulse.  
 (b) When  $J = K = 1$ , the outputs get complemented when a clock pulse is applied.  
 (c)  $J = 1, K = 0$  sets the flip-flop when clock is applied.  
 (d)  $J = 0, K = 1$  resets the flip-flop when clock is applied.
- 25.** List the different types of latches and flip-flops?
- A. The different types of latches are: active-low SR latch, active-high SR latch, D latch—all these are unclocked or non-gated latches. Gated latches called flip-flops are: gated SR latch and gated D latch. The flip-flops are: S-R flip-flop, D flip-flop, J-K flip-flop and T flip-flop which are edge triggered.
- 26.** Which flip-flops are not widely available commercially?
- A. The T flip-flops are not widely available as commercial units.
- 27.** Explain the operation of a SR flip-flop.
- A. (a) When  $S = R = 0$ , the outputs are not affected by the clock pulse.  
 (b) When  $S = R = 1$ , the output is ambiguous. It is invalid.  
 (c)  $S = 1, R = 0$  sets the flip-flop when clock is applied.  
 (d)  $S = 0, R = 1$  resets the flip-flop when clock is applied.
- 28.** How does a J-K flip-flop differ from an S-R flip-flop in its operation? What is its advantage over an S-R flip-flop?
- A. In an S-R flip-flop, the condition both inputs are equal to 1 is invalid, whereas in a J-K flip-flop both inputs are equal to 1 results in toggle mode. The advantage is in ripple counters, the flip-flops are to be in toggle mode.
- 29.** What is the main difference between a gated latch and an edge-triggered flip-flop?
- A. The gated latches are called level-triggered flip-flops. They respond to the inputs when the clock is high. The edge-triggered flip-flops respond to the inputs only at the transition of the clock.
- 30.** Which flip-flop is preferred for counting? Which one is preferred for data transfer?
- A. The J-K flip-flop is preferred for counting and the D flip-flop is preferred for data transfer.
- 31.** What do you mean by toggling?
- A. Toggling means changing the output to the opposite state each time a clock pulse is applied.
- 32.** Can a flip-flop respond to its control and clock inputs while  $\overline{PRE} = 1$  or  $\overline{CLEAR} = 1$ ?
- A. Yes, a flip-flop can respond to its control and clock inputs when  $\overline{PRE} = 1$  or  $\overline{CLEAR} = 1$  as preset and clear inputs have no effect when they are equal to 1.

**33.** What are PRESET and CLEAR inputs?

- A. PRESET and CLEAR inputs are asynchronous inputs. They override all other inputs. They are also called DC SET, and DC RESET or DC CLEAR, or Direct SET ( $S_D$ ) and direct RESET ( $R_D$ ). The PRESET input forces the output Q to 1 and CLEAR input forces the Q output to 0.

**34.** What do (a) a triangle, (b) a bubble and a triangle and (c) no symbol at the clock terminal of a flip-flop indicate?

- A. A triangle at the clock input terminal of a flip-flop indicates that it is a positive edge-triggered flip-flop. A bubble and a triangle at the clock input terminal of a flip-flop indicates that it is a negative edge-triggered flip-flop. No symbol at the clock input terminal of a flip-flop indicates that it is a level-triggered flip-flop.

**35.** Why are asynchronous inputs called overriding inputs?

- A. Asynchronous inputs are called overriding inputs because they override the control inputs, i.e. in the presence of override inputs no other inputs are effective.

**36.** What do you mean by clock skew?

- A. The clock signal which is applied simultaneously to all the flip-flops in a synchronous system may undergo varying degrees of delay caused by wiring between the components and arrive at the CLK inputs of different flip-flops at different times. This delay is called clock skew.

**37.** What do you mean by time race?

- A. If the clock skew is minimal, a flip-flop may get clocked before it receives a new input (derived from the output of another clocked flip-flop). On the other hand, if the clock pulse is delayed significantly, the inputs to a flip-flop may have changed before the clock pulse arrives. In these situations, we have a kind of race between the two competing signals that are attempting to accomplish opposite effects. This is called time race.

**38.** What is meant by race around condition in flip-flops?

- A. In a JK flip-flop when  $J = K = 1$  and clock is applied, the outputs go on complementing every  $\Delta T$  (delay time of flip-flop) as long as clock is present. Therefore, the output at the end of the clock pulse is ambiguous. This condition is known as the race around condition.

**39.** Typically, a manufacturer's data sheet specifies four different propagation delay times associated with a flip-flop? Name and describe them.

- A. The propagation delay times associated with a flip-flop are:

- (i) The propagation delays that occur in response to a positive transition on the clock inputs:
  - (a) Propagation delay  $t_{PLH}$  measured from the triggering of the clock pulse to the LOW-to-HIGH transition of the output.
  - (b) Propagation delay  $t_{PHL}$  measured from the triggering of the clock pulse to the HIGH-to-LOW transition of the output.
- (ii) The propagation delays that occur in response to signals on a flip-flop's synchronous inputs:
  - (a) Propagation delay  $t_{PLH}$  measured from the PRESET input to the LOW-to-HIGH transition of the output.
  - (b) Propagation delay  $t_{PHL}$  measured from the CLEAR input to the HIGH-to-LOW transition of the output.

**40.** Define the following terms as applied to flip-flops.

- |                            |                             |
|----------------------------|-----------------------------|
| (a) Set-up time            | (b) Hold time               |
| (c) Propagation delay time | (d) Maximum clock frequency |
| (e) Power dissipation      |                             |

- A.** (a) *Set-up time:* The set-up time ( $t_s$ ) is the minimum time for which the control levels need to be maintained constant on the input terminals of the flip-flop prior to the arrival of the triggering edge of the clock pulse in order to enable the flip-flop to respond reliably.  
 (b) *Hold time:* The hold time ( $t_h$ ) is the minimum time for which the control signals need to be maintained constant at the input terminals of the flip-flop after the arrival of the triggering edge of the clock pulse, in order to enable the flip-flop to respond reliably.  
 (c) *Propagation delay time:* The time interval between the time of application of the triggering edge or asynchronous inputs and the time at which the output actually makes a transition is called the propagation delay time of the flip-flop.  
 (d) *Maximum clock frequency:* The maximum clock frequency ( $f_{\max}$ ) is the highest frequency at which a flip-flop can be reliably triggered.  
 (e) *Power dissipation:* The power dissipation of a flip-flop is the total power consumption of the device. It is equal to the product of the supply voltage ( $V_{CC}$ ) and the current ( $I_{CC}$ ) drawn from the supply by it.  $P = V_{CC} I_{CC}$ .

**41.** How do you convert one type of flip-flop into another?

- A.** To convert one type of flip-flop into another type, a combinational circuit is designed such that if the inputs of the required flip-flop (along with the outputs of the actual flip-flop if required) are fed as inputs to the combinational circuit and the output of the combinational circuit is connected to the inputs of the actual flip-flop, then the output of the actual flip-flop is the output of the required flip-flop.

**42.** What is a master-slave flip-flop?

- A.** A master-slave flip-flop is a cascade of two flip-flops in which the first one responds to the data inputs when the clock is high, whereas the second one responds to the outputs of the first one when the clock is low. Thus the final outputs change only when clock is low, when the data inputs are not effective. Thus the race around condition gets eliminated in this. The first flip-flop is known as the master and the second as the slave.

**43.** Why are master-slave flip-flops called pulse-triggered flip-flops?

- A.** Master-slave flip-flops are called pulse-triggered flip-flops, because the length of the time required for its output to change state equals the width of one (clock) pulse.

**44.** Differentiate between edge-triggered and master-slave flip-flops.

- A.** In edge-triggered flip-flops only a positive or negative edge is required for triggering, whereas in the case of master-slave flip-flops both a positive and a negative edge are required for triggering.

**45.** What are data lock-out flip-flops?

- A.** Data lock-out flip-flops are nothing but master-slave flip-flops in which the master is an edge-triggered flip-flop.

**46.** What is a trigger?

- A.** A momentary change of signal level and return to the initial level is referred to as a trigger.

**47.** What do you mean by excitations?

- A.** The inputs to the flip-flops are called excitations.

**48.** What is an excitation table?

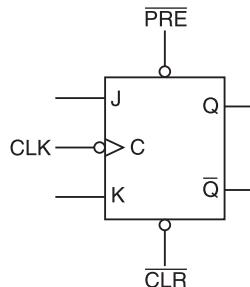
- A.** An excitation table is a table which lists the inputs required to be applied to the flip-flop to take it from the given present state to the required next state.

### REVIEW QUESTIONS

1. Distinguish between combinational and sequential logic circuits.
2. Draw the circuit diagram of a positive edge-triggered JK flip-flop and explain its operation with the help of a truth table. How is the race around condition eliminated?
3. Draw the schematic circuit of an edge-triggered JK flip-flop with ‘active low preset’ and ‘active low clear’ using NAND gates and explain its operation with the help of a truth table.
4. (a) Draw and explain with the help of a truth table the logic diagram of a master-slave D flip-flop using NAND gates with active low preset and clear and with negative edge-triggered clock.  
 (b) Draw a schematic circuit of a D flip-flop with negative edge triggering using NAND gates. Give its truth table and explain its operation.
5. (a) Define the following terms with relation to flip-flop:
 

(i) set-up time	(ii) hold time
(iii) propagation delay time	(iv) preset
(v) clear	

 (b) For the block diagram shown, draw the schematic circuit using NAND gates and explain its operation with the help of a truth table.



6. Draw the circuit diagram of a master-slave J-K flip-flop and explain its operation with the help of a truth-table. How is it different from edge triggering? Explain.
7. Explain the following:
 

(a) Race around condition in flip-flop	(b) J-K master-slave flip-flop
(c) Excitation table for flip-flops	
8. Give the transition table for the following flip-flops:
 

(a) SR flip-flop	(b) JK flip-flop	(c) T flip-flop	(d) D flip-flop
------------------	------------------	-----------------	-----------------

 Briefly state the salient features of each flip-flop.
9. Find the characteristic equation for:
 

(a) SR flip-flop	(b) JK flip-flop	(c) T flip-flop	(d) D flip-flop
------------------	------------------	-----------------	-----------------
10. Convert a J-K flip-flop into:
 

(a) SR flip-flop	(b) T flip-flop	(c) D flip-flop
------------------	-----------------	-----------------
11. Convert a D flip-flop into:
 

(a) SR flip-flop	(b) JK flip-flop	(c) T flip-flop
------------------	------------------	-----------------

12. Convert a T flip-flop into:
  - (a) SR flip-flop
  - (b) JK flip-flop
  - (c) D flip-flop
13. Convert an SR flip-flop into:
  - (a) T flip-flop
  - (b) JK flip-flop
  - (c) D flip-flop
14. Discuss the applications of flip-flops.

**FILL IN THE BLANKS**

1. A \_\_\_\_\_ is the basic memory element.
2. A flip-flop is known more formally as a \_\_\_\_\_.
3. A flip-flop has \_\_\_\_\_ stable states.
4. A \_\_\_\_\_ is called a latch.
5. Non-gated latches are called \_\_\_\_\_ latches and gated latches are called \_\_\_\_\_ latches.
6. A latch is constructed using two cross coupled \_\_\_\_\_ gates or \_\_\_\_\_ gates.
7. A latch may be an \_\_\_\_\_ latch or an \_\_\_\_\_ latch.
8. The NOR gate S-R latch is an \_\_\_\_\_ S-R latch.
9. The NAND gate S-R latch is an \_\_\_\_\_ S-R latch.
10. The clocked D latch is called a \_\_\_\_\_ D latch.
11. Flip-flops may be \_\_\_\_\_ triggered or \_\_\_\_\_ triggered.
12. Edge triggering is also called \_\_\_\_\_ triggering.
13. The \_\_\_\_\_ flip-flop is the most versatile and the most widely used of all the flip-flops.
14. \_\_\_\_\_ flip-flops are not widely available as commercial items.
15. \_\_\_\_\_ and \_\_\_\_\_ are asynchronous inputs.
16. No symbol at the clock input terminal of a flip-flop indicates that it is \_\_\_\_\_.
17. A bubble and a triangle at the clock input terminal of a flip-flop indicate that it is a \_\_\_\_\_ flip-flop.
18. A triangle at the clock input terminal of a flip-flop indicates that it is a \_\_\_\_\_ flip-flop.
19. The undergoing of varying degrees of delay by a clock pulse before it arrives at the flip-flop is called \_\_\_\_\_.
20. In an S-R flip-flop,  $S = 1, R = 1$  \_\_\_\_\_ permitted.
21. For a J-K flip-flop,  $J = 1, K = 1$  is the \_\_\_\_\_ mode.
22. PRESET and CLEAR inputs in a flip-flop are used for making  $Q = \text{_____}$  and \_\_\_\_\_ respectively.
23. Master-slave configuration is used in J-K flip-flops to eliminate \_\_\_\_\_.
24. A momentary change of signal level to the initial level is called \_\_\_\_\_.
25. The master-slave flip-flop is a \_\_\_\_\_ triggered flip-flop.
26. The race around condition is eliminated using \_\_\_\_\_ configuration.
27. An equation expressing the next state  $Q_{n+1}$  in terms of the present state  $Q_n$  and the excitations of the flip-flop is called the \_\_\_\_\_.
28. \_\_\_\_\_ flip-flop is preferred for counting and \_\_\_\_\_ flip-flop is preferred for data transfer.
29. PRESET and CLEAR are \_\_\_\_\_ inputs.
30. Asynchronous inputs are called \_\_\_\_\_ inputs.

31. The name flip-flop is because this circuit shifts \_\_\_\_\_ and \_\_\_\_\_ between its two stable states upon application of proper inputs.
32. The state in which a circuit can remain permanently is called \_\_\_\_\_.
33. The input signals to a flip-flop are called \_\_\_\_\_.

### OBJECTIVE TYPE QUESTIONS

1. A combinational logic circuit
  - (a) must contain flip-flops
  - (b) may contain flip-flops
  - (c) does not contain flip-flops
  - (d) contains latches
2. The output of a logic circuit depends upon the sequence in which the input is applied. The circuit
  - (a) is a combinational logic circuit
  - (b) is a sequential logic circuit
  - (c) may be a combinational or sequential logic circuit
  - (d) is none of the above
3. A sequential circuit does not use clock pulses. It is
  - (a) an asynchronous sequential circuit
  - (b) a synchronous sequential circuit
  - (c) a counter
  - (d) a shift register
4. In a multi-input sequential circuit only one input is allowed to change at a time. This circuit
  - (a) consists of latches and combinational circuit
  - (b) is a clocked sequential circuit
  - (c) is a serial adder circuit
  - (d) may be a synchronous counter or a shift register
5. An asynchronous sequential circuit
  - (a) does not use clock pulses
  - (b) only one input can change at a time
  - (c) consists of combinational circuit and latches
  - (d) meets all the above conditions
6. The basic memory element in a digital circuit
  - (a) consists of a NAND gate
  - (b) consists of a NOR gate
  - (c) is a flip-flop
  - (d) is a shift register
7. A flip-flop has two outputs which are
  - (a) always 0
  - (b) always 1
  - (c) always complementary
  - (d) all of the above states
8. A flip-flop can be made using
  - (a) basic gates such as AND, OR and NOT
  - (b) NAND gates
  - (c) NOR gates
  - (d) any of the above
9. Which of the following flip-flop is used as a latch?
  - (a) J-K flip-flop
  - (b) Master-slave flip-flop
  - (c) T flip-flop
  - (d) D flip-flop



**594 FUNDAMENTALS OF DIGITAL CIRCUITS**

22. The output  $Q_n$  of a J-K flip-flop is 0. It changes to 1 when a clock pulse is applied. The inputs  $J_n$  and  $K_n$  are respectively.  
(a) 1 and X      (b) 0 and X      (c) X and 0      (d) X and 1
23. The output  $Q_n$  of a J-K (or S-R) flip-flop is 0. Its output does not change when a clock pulse is applied. The inputs  $J_n$  and  $K_n$  (or  $S_n R_n$ ) are respectively  
(a) X and 0      (b) X and 1      (c) 1 and X      (d) 0 and X
24. The output  $Q_n$  of a J-K flip-flop is 1. Its output does not change when a clock pulse is applied. The inputs  $J_n$  and  $K_n$  are respectively  
(a) 0X      (b) X0      (c) 10      (d) 01
25. The outputs  $Q$  and  $\bar{Q}$  of a master-slave S-R flip-flop are connected to its R and S inputs respectively. Its output Q when clock pulses are applied will be  
(a) permanently 0      (b) permanently 1  
(c) fixed 0 or 1      (d) complementing with every clock pulse
26. Flip-flops can be used to make  
(a) latches      (b) bounce-elimination switches  
(c) registers      (d) all of the above
27. A master-slave flip-flop is triggered  
(a) when the clock input is at HIGH logic level  
(b) when the clock input makes a transition from LOW to HIGH  
(c) when a pulse is applied at the clock input  
(d) when the clock input is at LOW logic level.
28. A J-K M-S flip-flop comprises which of the following configurations?  
(a) S-R flip-flop followed by an S-R flip-flop      (b) S-R flip-flop followed by a J-K flip flop  
(c) J-K flip flop followed by a J-K flip flop      (d) J-K flip-flop followed by an S-R flip-flop
29. In a J-K M-S flip-flop, race around is eliminated because of which of the following reasons?  
(a) output of slave is fed back to the input of master  
(b) output of master is fed back to the input of slave  
(c) while the clock drives the master, inverted clock drives the slave  
(d) J-K flip-flop is followed by S-R flip-flop
30. The toggle mode for a J-K flip-flop is  
(a)  $J = 0, K = 0$       (b)  $J = 1, K = 0$       (c)  $J = 0, K = 1$       (d)  $J = 1, K = 1$
31. The transparent latch is  
(a) an S-R flip-flop      (b) a D flip-flop      (c) a T flip-flop      (d) a J-K flip-flop
32. In a master-slave J-K flip-flop,  $J = K = 1$ . The state  $Q_{n+1}$  of the flip-flop after the clock pulse will be  
(a) 0      (b) 1      (c)  $Q_n$       (d)  $\bar{Q}_n$
33. The characteristic equation of a J-K flip-flop is  
(a)  $Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$       (b)  $JQ_n + K\bar{Q}_n$       (c)  $\bar{J}Q_n + \bar{K}Q_n$       (d)  $\bar{J}\bar{Q}_n + KQ_n$
34. The characteristic equation of a D flip-flop is  
(a)  $Q_{n+1} = D$       (b)  $Q_{n+1} = Q_n$       (c)  $Q_{n+1} = 1$       (d)  $Q_{n+1} = \bar{Q}_n$

35. The characteristic equation of a T flip-flop is

(a)  $Q_{n+1} = \bar{Q}_n T + Q_n \bar{T}$

(b)  $Q_{n+1} = \bar{Q}_n \bar{T} + Q_n T$

(c)  $Q_{n+1} = Q_n$

(d)  $Q_{n+1} = \bar{Q}_n$

36. The characteristic equation of an S-R flip-flop is

(a)  $Q_{n+1} = Q_n \bar{R} + S$

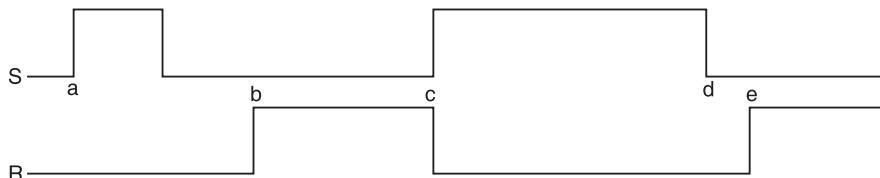
(b)  $Q_{n+1} = \bar{Q}_n R + S$

(c)  $Q_{n+1} = Q_n R + \bar{S}$

(d)  $Q_{n+1} = Q_n$

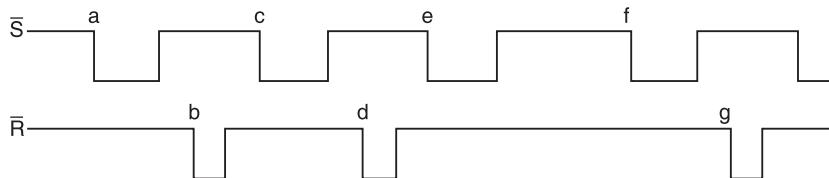
## PROBLEMS

10.1 If the waveforms shown in Figure P10.1 are applied to an active-HIGH S-R latch which is in the RESET state, draw the output waveform.



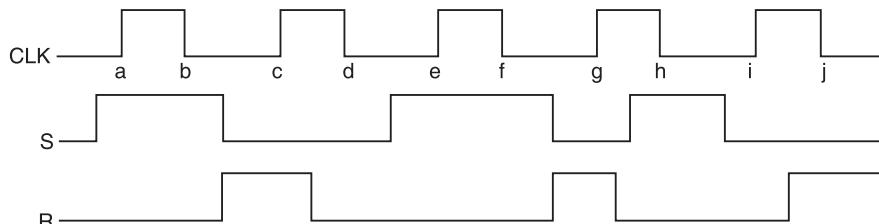
**Figure P10.1**

10.2 If the waveforms shown in Figure P10.2 are applied to an active-LOW S-R latch which is in the RESET state, draw the output waveform.



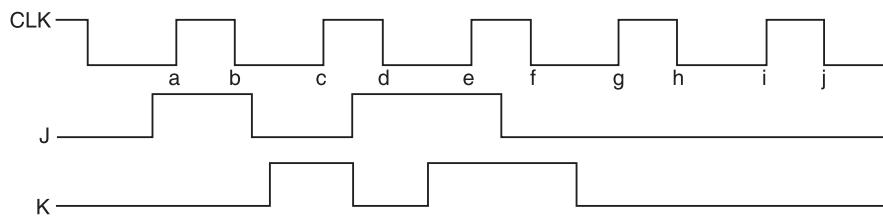
**Figure P10.2**

10.3 The waveforms shown in Figure P10.3 are applied to (a) a positive edge-triggered S-R flip-flop and (b) a negative edge-triggered S-R flip-flop. Draw the output waveform in each case.

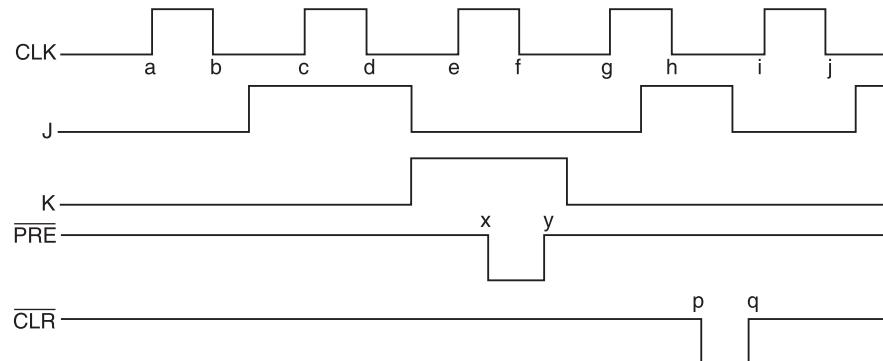


**Figure P10.3**

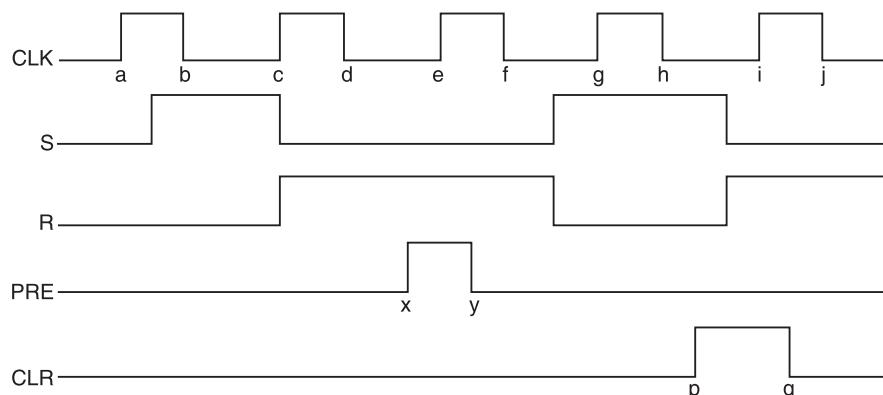
- 10.4** The waveforms shown in Figure P10.4 are applied to (a) a positive-edge-triggered J-K flip-flop, (b) a negative edge-triggered J-K flip-flop, and (c) a master-slave J-K flip-flop. Draw the output waveform in each case.

**Figure P10.4**

- 10.5** The input signals shown in Figure P10.5 are applied to a positive edge-triggered J-K master-slave flip-flop with active-LOW PRESET and CLEAR. Draw the output waveform.

**Figure P10.5**

- 10.6** The waveforms shown in Figure P10.6 are applied to a negative edge-triggered S-R flip-flop with active-HIGH PRESET and CLEAR. Draw the output waveform.

**Figure P10.6**

- 10.7 The following serial data are applied to the flip-flop shown in Figure P10.7. Determine the resulting serial data that appears on the Q output. There is one clock pulse for each bit time. Assume that, Q is initially 0. The rightmost bits are applied first.

$$J_1 = 10110110$$

$$J_2 = 11011001$$

$$K_1 = 10010110$$

$$K_2 = 11011011$$

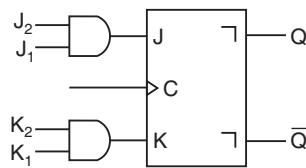


Figure P10.7

## VHDL PROGRAMS

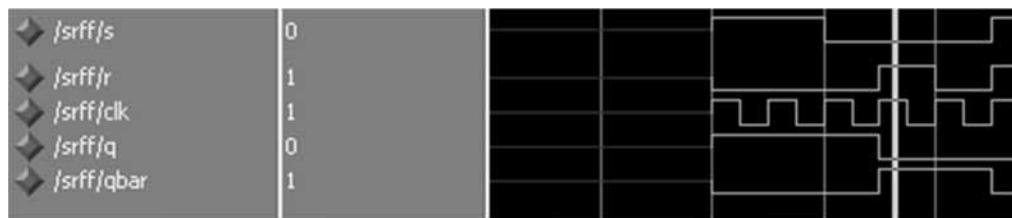
### 1. VHDL PROGRAM FOR S-R FLIP-FLOP

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity srff is
    Port ( s : in std_logic;
           r : in std_logic;
           clk : in std_logic;
           q : inout std_logic;
           qbar : inout std_logic);
end srff;
architecture Behavioral of srff is
begin
    process (s, r, clk, q, qbar)
        variable s1, s2 : std_logic;
    begin
        s1 := clk and r;
        s2 := clk and s;
        q <= s1 nor qbar;
        qbar <= s2 nor q;
    end process;
end Behavioral;

```

### SIMULATION OUTPUT:



### 2. VHDL PROGRAM FOR J-K FLIP-FLOP

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity jkff is
    Port ( j : in std_logic;
           k : in std_logic;
           clk : in std_logic;
           rst : in std_logic;
           q : inout std_logic;
           qbar : inout std_logic);
end jkff;

```

```

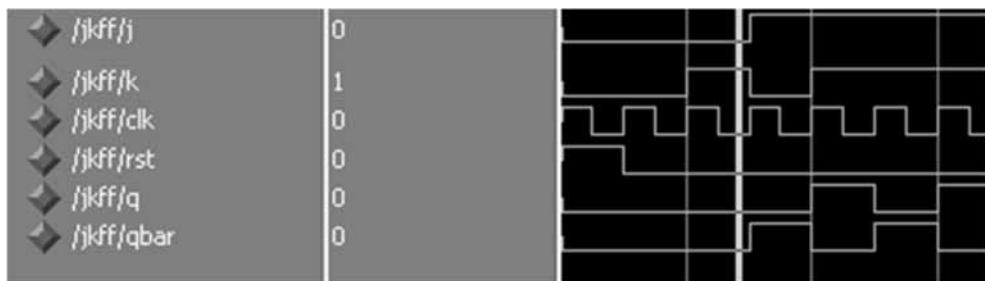
architecture Behavioral of jkff is

begin
    process (j, k, clk, rst, q, qbar)
    begin
        if (rst = '1') then
            q <= '0';
            qbar <= '0';
        else if (clk = '1' and clk'event) then
            if (j = '0' and k = '0' ) then
                q <= q; qbar <= qbar;
            elsif (j = '0' and k = '1' ) then
                q <='0'; qbar <= '1';
            elsif (j = '1' and k = '0' ) then
                q <='1'; qbar <= '0';
            else
                q <= qbar; qbar <= q;
            end if;
        end if;
        end if;
    end process;

end Behavioral;

```

### SIMULATION OUTPUT:



### 3. VHDL PROGRAM FOR T FLIP-FLOP

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity tff is
    Port ( clk : in std_logic;
            rst : in std_logic;
            t : in std_logic;

```

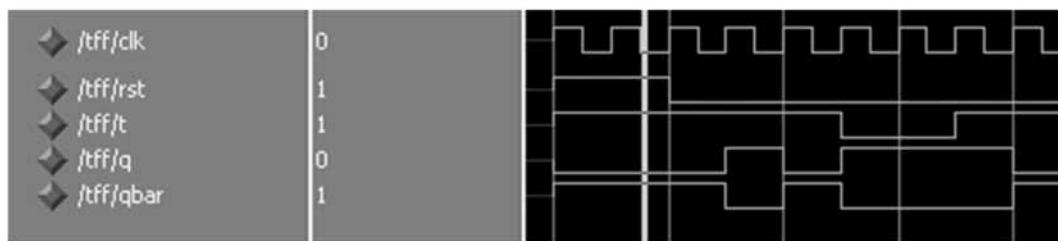
## 600 FUNDAMENTALS OF DIGITAL CIRCUITS

```
        q : inout std_logic;
        qbar : inout std_logic);
end tff;

architecture Behavioral of tff is

begin
process (clk,rst,t)
begin
if(rst='1')then
q<='0';
qbar<='1';
elsif(clk='1' and clk'event)then
if(t='0')then
q<=q;
qbar<=qbar;
else
q<=not(q);
qbar<=not(qbar);
end if;
end if;
end process;
end Behavioral;
```

### SIMULATION OUTPUT:



### 4. VHDL PROGRAM FOR D FLIP-FLOP

```
library ieee;
use ieee.std_logic_1164.all;

entity dff is
port (data  :in std_logic;
      clk   :in std_logic;
      reset :in std_logic;
      q     :out std_logic );
end dff;
```

```

architecture Behavioral of dff is

begin
    process (clk) begin
        if (reset = '1') then
            q <= '0';

        elsif (rising_edge(clk)) then

            q <= data;
        end if;
    end process;

end Behavioral ;

```

#### SIMULATION OUTPUT:



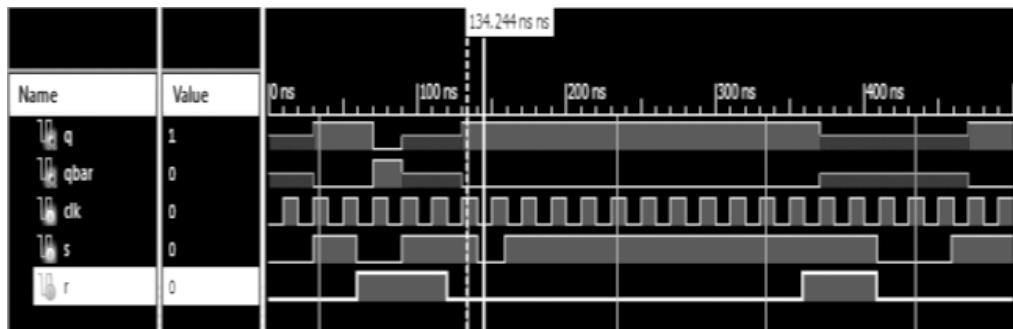
## VERILOG PROGRAMS

### 1. VERILOG PROGRAM FOR S-R FLIP-FLOP

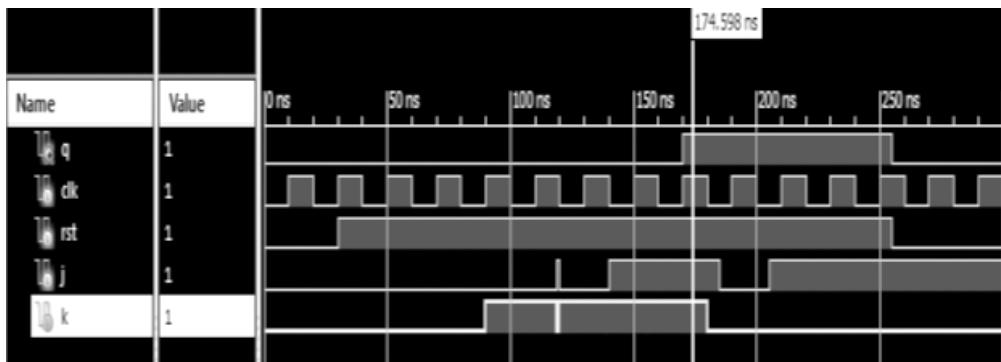
```

module SR_ff(q,qbar,s,r,clk);
output q,qbar;
input clk,s,r;
reg tq;
always @(posedge clk)
begin
if (s == 1'b0 && r == 1'b0)
tq <= tq;
else if (s == 1'b0 && r == 1'b1)
tq <= 1'b0;
else if (s == 1'b1 && r == 1'b0)
tq <= 1'b1;
else if (s == 1'b1 && r == 1'b1)
tq <= 1'bx;
end
assign q = tq;
assign qbar = ~tq;
endmodule

```

**SIMULATION OUTPUT:****2. VERILOG PROGRAM FOR J-K FLIP-FLOP**

```
module jk_ff(q,qbar,clk,rst,j,k);
input clk,rst,j,k;
output q,qbar;
reg q,tq;
always @(posedge clk or negedge rst)
begin
if (!rst)
begin
q <= 1'b0;
tq <= 1'b0;
end
else
begin
if (j == 1'b1 && k == 1'b0)
q <= j;
else if (j == 1'b0 && k == 1'b1)
q <= 1'b0;
else if (j == 1'b1 && k == 1'b1)
begin
tq <= ~tq;
q <= tq;
end
end
end
assign q_bar = ~q;
endmodule
```

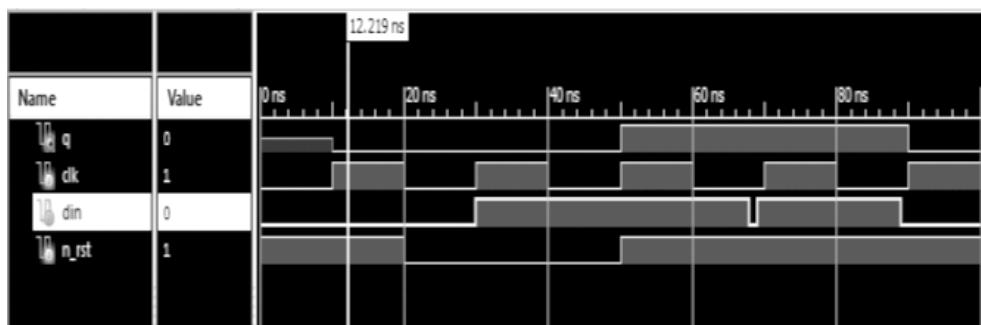
**SIMULATION OUTPUT:****3. VERILOG PROGRAM FOR T FLIP-FLOP**

```
module t_ff(q,qbar,clk,tin,rst);
output q,qbar;
input clk,tin,rst;
reg tq;
always @(posedge clk or negedge rst)
begin
if(!rst)
tq <= 1'b0;
else
begin
if (tin)
tq <= ~tq;
end
end
assign q = tq;
assign qbar = ~q;
endmodule
```

**SIMULATION OUTPUT:**

**4. VERILOG PROGRAM FOR D FLIP-FLOP**

```
module d_ff(q,clk,n_rst,din);
output q;
input clk,din,n_rst;
reg q;
always @(posedge clk or negedge n_rst)
begin
if (!n_rst)
q <= 1'b0;
else
q <= din;
end
endmodule
```

**SIMULATION OUTPUT:**

# 11

## SHIFT REGISTERS

### 11.1 INTRODUCTION

Data may be available in parallel form or in serial form. Multi-bit data is said to be in parallel form when all the bits are available (accessible) simultaneously. The data is said to be in serial form when the data bits appear sequentially (one after the other, in time) at a single terminal. Data may also be transferred in parallel form or in serial form. Parallel data transfer is the simultaneous transmission of all bits of data from one device to another. Serial data transfer is the transmission of one bit of data at a time from one device to another. Serial data must be transmitted under the synchronization of a clock, since the clock provides the means to specify the time at which each new bit is sampled.

As a flip-flop (FF) can store only one bit of data, a 0 or a 1, it is referred to as a single-bit register. When more bits of data are to be stored, a number of FFs are used. A register is a set of FFs used to store binary data. The storage capacity of a register is the number of bits (1s and 0s) of digital data it can retain. Loading a register means setting or resetting the individual FFs, i.e. inputting data into the register so that their states correspond to the bits of data to be stored. Loading may be serial or parallel. In serial loading, data is transferred into the register in serial form, i.e. one bit at a time, whereas in parallel loading, the data is transferred into the register in parallel form meaning that all the FFs are triggered into their new states at the same time. Parallel input requires that the SET and/or RESET controls of every FF be accessible.

A register may output data either in serial form or in parallel form. Serial output means that the data is transferred out of the register, one bit at a time serially. Parallel output means that the entire data stored in the register is available in parallel form, and can be transferred out at the same time.

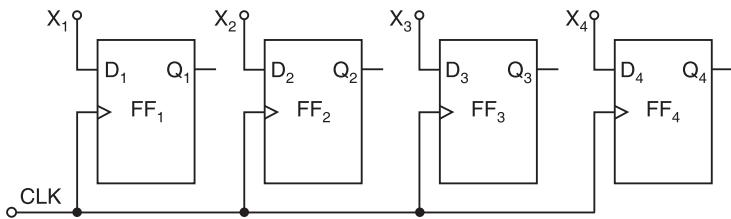
Shift registers are a type of logic circuits closely related to counters. They are used basically for the storage and transfer of digital data. The basic difference between a shift register and a counter is that, a shift register has no specified sequence of states except in certain very specialized applications, whereas a counter has a specified sequence of states.

A shift register is a very important digital building block. It has innumerable applications. Registers are often used to momentarily store binary information appearing at the output of an encoding matrix. A register might be used to accept input data from an alphanumeric keyboard and then present the data at the input of a microprocessor chip. Similarly, shift registers are often used to momentarily store binary data at the output of a decoder. A shift register also forms the basis for some very important arithmetic operations. For example, the operations of complementation, multiplication, and division are frequently implemented by means of a register. A shift register can also be connected to form a number of different types of counters. These counters offer some very distinct advantages.

## 11.2 BUFFER REGISTER

Some registers do nothing more than storing a binary word. The buffer register is the simplest of registers. It simply stores the binary word. The buffer may be a controlled buffer. Most of the buffer registers use D flip-flops.

Figure 11.1 shows a 4-bit buffer register. The binary word to be stored is applied to the data terminals. On the application of clock pulse, the output word becomes the same as the word applied at the input terminals, i.e. the input word is loaded into the register by the application of clock pulse.



**Figure 11.1** Logic diagram of a 4-bit buffer register.

When the positive clock edge arrives, the stored word becomes:

$$Q_4 Q_3 Q_2 Q_1 = X_4 X_3 X_2 X_1$$

or

$$Q = X$$

This circuit is too primitive to be of any use. What it needs is some control over the X bits, i.e. some way of holding them off until we are ready to store them.

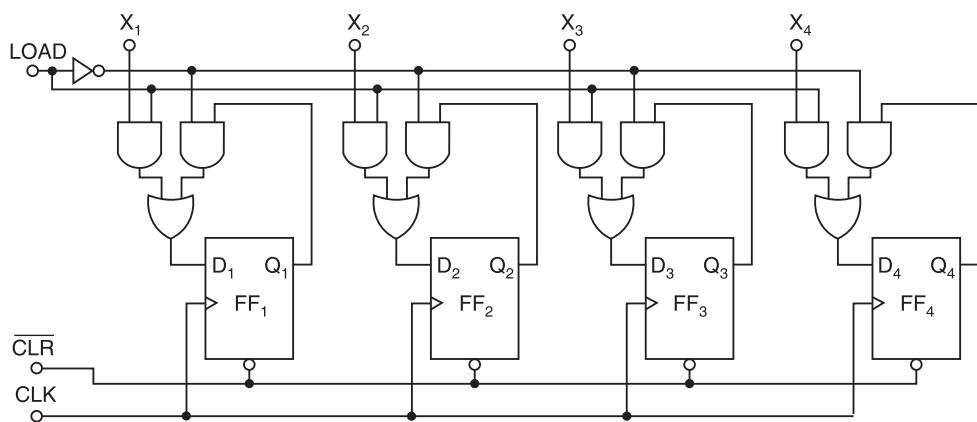
### 11.3 CONTROLLED BUFFER REGISTER

Figure 11.2 shows a controlled buffer register. If  $\overline{\text{CLR}}$  goes LOW, all the FFs are RESET and the output becomes,  $Q = 0000$ .

When  $\overline{\text{CLR}}$  is HIGH, the register is ready for action. LOAD is the control input. When LOAD is HIGH, the data bits  $X$  can reach the D inputs of FFs. At the positive-going edge of the next clock pulse, the register is loaded, i.e.

$$Q_4 Q_3 Q_2 Q_1 = X_4 X_3 X_2 X_1$$

or



**Figure 11.2** Logic diagram of a 4-bit controlled buffer register.

When LOAD is LOW, the X bits cannot reach the FFs. At the same time, the inverted signal LOAD is HIGH. This forces each flip-flop output to feed back to its data input. Therefore, data is circulated or retained as each clock pulse arrives. In other words, the contents of the register remain unchanged in spite of the clock pulses. Longer buffer registers can be built by adding more FFs.

### 11.4 DATA TRANSMISSION IN SHIFT REGISTERS

A number of FFs connected together such that data may be shifted into and shifted out of them is called a *shift register*. Data may be shifted into or out of the register either in serial form or in parallel form. So, there are four basic types of shift registers: serial-in, serial-out; serial-in, parallel-out; parallel-in, serial-out; and parallel-in, parallel-out. The process of data shifting in these registers is illustrated in Figure 11.3. All of these configurations are commercially available as TTL MSI/LSI circuits. Data may be rotated left or right. Data may be shifted from left to right or right to left at will, i.e. in a bidirectional way. Also, data may be shifted in serially (in either way) or in parallel and shifted out serially (in either way) or in parallel.

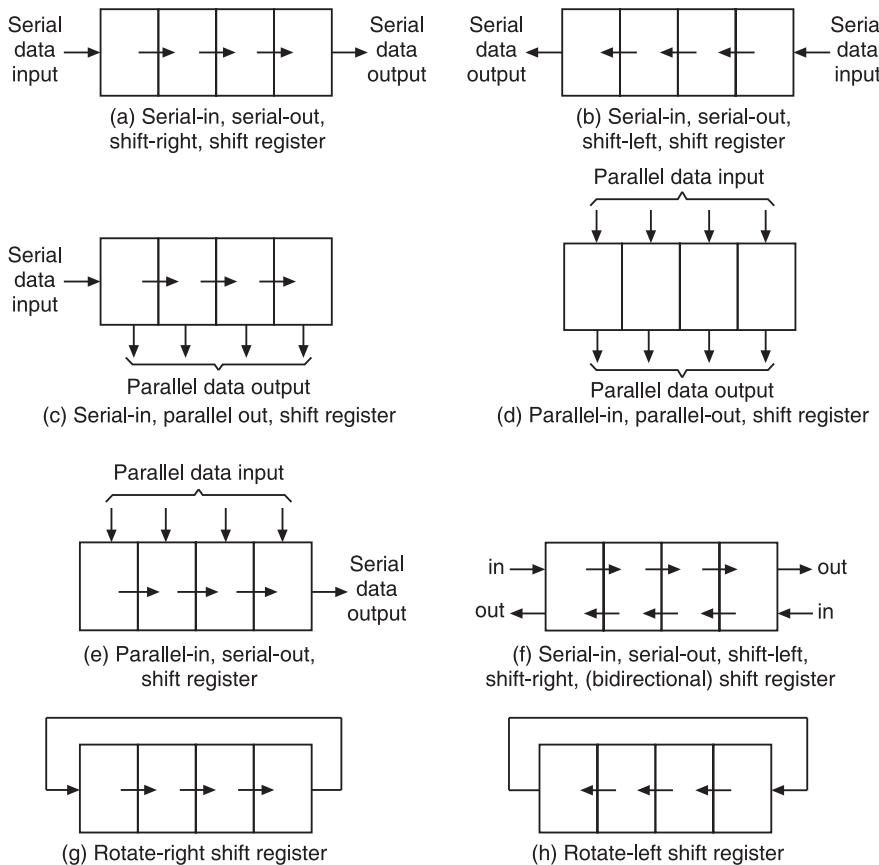


Figure 11.3 Data transfer in registers.

## 11.5 SERIAL-IN, SERIAL-OUT, SHIFT REGISTER

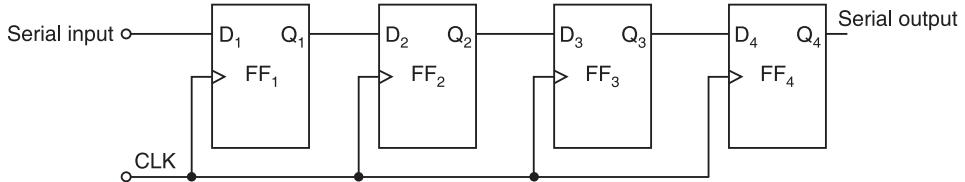
This type of shift register accepts data serially, i.e. one bit at a time, and also outputs data serially.

The logic diagram of a 4-bit serial-in, serial-out, shift-right, shift register is shown in Figure 11.4. With four stages, i.e. four FFs, the register can store upto four bits of data. Serial data is applied at the D input of the first FF. The Q output of the first FF is connected to the D input of the second FF, the Q output of the second FF is connected to the D input of the third FF and the Q output of the third FF is connected to the D input of the fourth FF. The data is outputted from the Q terminal of the last FF.

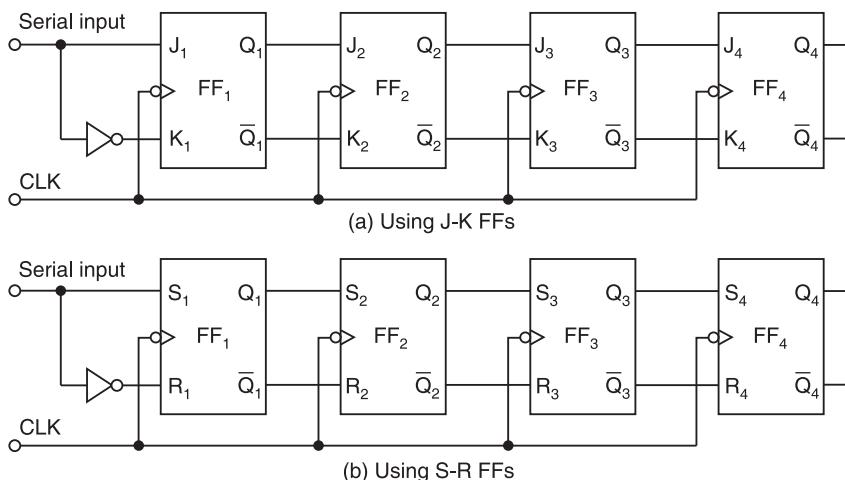
When serial data is transferred into a register, each new bit is clocked into the first FF at the positive-going edge of each clock pulse. The bit that was previously stored by the first FF is transferred to the second FF. The bit that was stored by the second FF is transferred to the third FF, and so on. The bit that was stored by the last FF is shifted out.

A shift register can also be constructed using J-K FFs or S-R FFs as shown in Figures 11.5a and 11.5b, respectively. The data is applied at the J(S) input of the first FF. The complement of this is fed to the K(R) terminal of the first FF. The Q output of the first FF is connected to J(S)

input of the second FF, the Q output of the second FF to J(S) input of the third FF, and so on. Also,  $\bar{Q}_1$  is connected to  $K_2$  ( $R_2$ ),  $\bar{Q}_2$  is connected to  $K_3$  ( $R_3$ ), and so on.

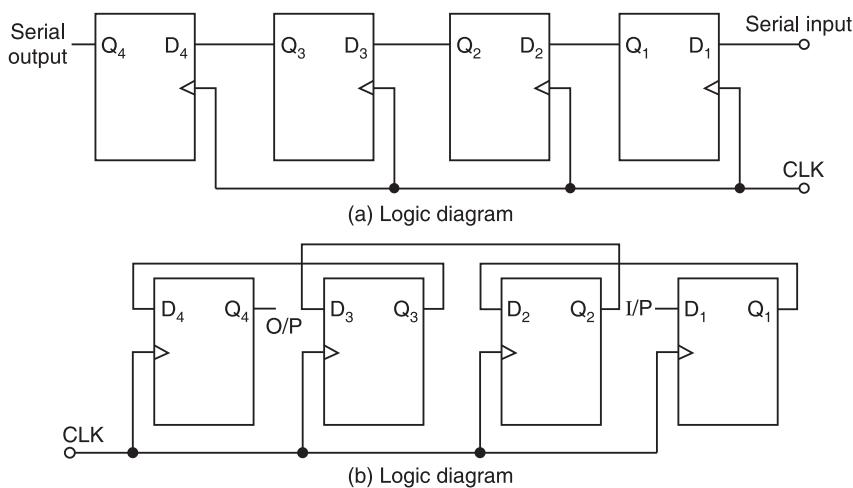


**Figure 11.4** 4-bit serial-in, serial-out, shift-right, shift register.



**Figure 11.5** A 4-bit serial-in, serial-out, shift register.

Figure 11.6 shows the logic diagrams of a 4-bit serial-in, serial-out, shift-left, shift register.

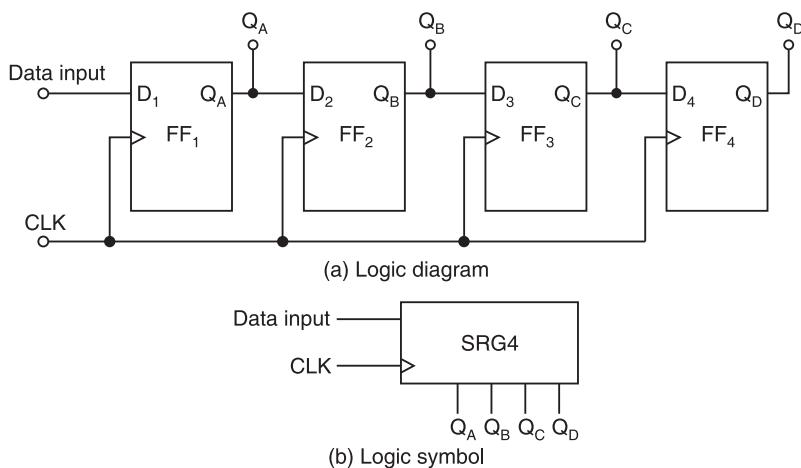


**Figure 11.6** A 4-bit serial-in, serial-out, shift-left, shift register.

## 11.6 SERIAL-IN, PARALLEL-OUT, SHIFT REGISTER

Figure 11.7 shows the logic diagram and the logic symbol of a 4-bit serial-in, parallel-out, shift register. In this type of register, the data bits are entered into the register serially, but the data stored in the register is shifted out in parallel form.

Once the data bits are stored, each bit appears on its respective output line and all bits are available simultaneously, rather than on a bit-by-bit basis as with the serial output. The serial-in, parallel-out, shift register can be used as a serial-in, serial-out, shift register if the output is taken from the Q terminal of the last FF.



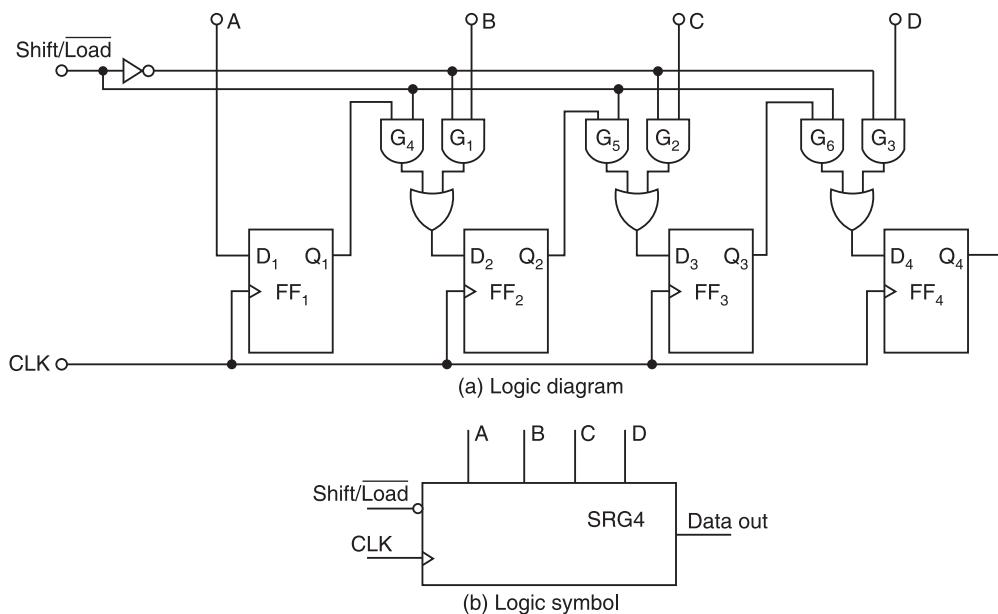
**Figure 11.7** A 4-bit serial-in, parallel-out, shift register.

## 11.7 PARALLEL-IN, SERIAL-OUT, SHIFT REGISTER

For a parallel-in, serial-out, shift register, the data bits are entered simultaneously into their respective stages on parallel lines, rather than on a bit-by-bit basis on one line as with serial data inputs, but the data bits are transferred out of the register serially, i.e. on a bit-by-bit basis over a single line.

Figure 11.8 illustrates a 4-bit parallel-in, serial-out, shift register using D FFs. There are four data lines A, B, C, and D through which the data is entered into the register in parallel form. The signal Shift/LOAD allows (a) the data to be entered in parallel form into the register and (b) the data to be shifted out serially from terminal Q<sub>4</sub>.

When Shift/LOAD line is HIGH, gates G<sub>1</sub>, G<sub>2</sub>, and G<sub>3</sub> are disabled, but gates G<sub>4</sub>, G<sub>5</sub>, and G<sub>6</sub> are enabled allowing the data bits to shift-right from one stage to the next. When Shift/LOAD line is LOW, gates G<sub>4</sub>, G<sub>5</sub>, and G<sub>6</sub> are disabled, whereas gates G<sub>1</sub>, G<sub>2</sub>, and G<sub>3</sub> are enabled allowing the data input to appear at the D inputs of the respective FFs. When a clock pulse is applied, these data bits are shifted to the Q output terminals of the FFs and, therefore, data is inputted in one step. The OR gate allows either the normal shifting operation or the parallel data entry depending on which AND gates are enabled by the level on the Shift/LOAD input.

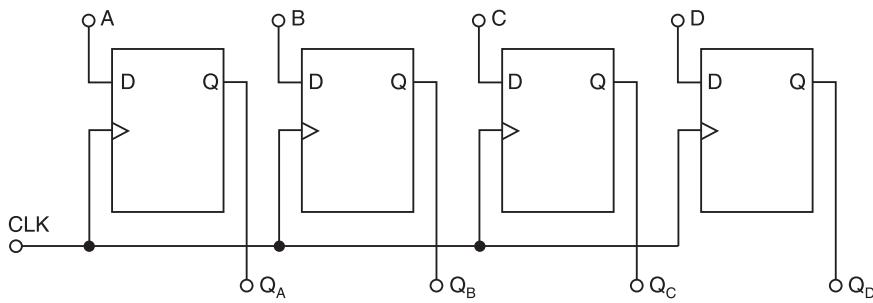


**Figure 11.8** A 4-bit parallel-in, serial-out, shift register.

## 11.8 PARALLEL-IN, PARALLEL-OUT, SHIFT REGISTER

In a parallel-in, parallel-out, shift register, the data is entered into the register in parallel form, and also the data is taken out of the register in parallel form. Immediately following the simultaneous entry of all data bits, the bits appear on the parallel outputs.

Figure 11.9 shows a 4-bit parallel-in, parallel-out, shift register using D FFs. Data is applied to the D input terminals of the FFs. When a clock pulse is applied, at the positive-going edge of that pulse, the D inputs are shifted into the Q outputs of the FFs. The register now stores the data. The stored data is available instantaneously for shifting out in parallel form.



**Figure 11.9** Logic diagram of a 4-bit parallel-in, parallel-out, shift register.

## 11.9 BIDIRECTIONAL SHIFT REGISTER

A bidirectional shift register is one in which the data bits can be shifted from left to right or from right to left.

Figure 11.10 shows the logic diagram of a 4-bit serial-in, serial-out, bidirectional (shift-left, shift-right) shift register. Right/Left is the mode signal. When Right/Left is a 1, the logic circuit works as a shift-right shift register. When Right/Left is a 0, it works as a shift-left shift register. The bidirectional operation is achieved by using the mode signal and two AND gates and one OR gate for each stage as shown in Figure 11.10.

A HIGH on the Right/Left control input enables the AND gates  $G_1$ ,  $G_2$ ,  $G_3$ , and  $G_4$  and disables the AND gates  $G_5$ ,  $G_6$ ,  $G_7$ , and  $G_8$ , and the state of Q output of each FF is passed through the gate to the D input of the following FF. When a clock pulse occurs, the data bits are then effectively shifted one place to the right. A LOW on the Right/Left control input enables the AND gates  $G_5$ ,  $G_6$ ,  $G_7$ , and  $G_8$  and disables the AND gates  $G_1$ ,  $G_2$ ,  $G_3$ , and  $G_4$ , and the Q output of each FF is passed to the D input of the preceding FF. When a clock pulse occurs, the data bits are then effectively shifted one place to the left. Hence, the circuit works as a bidirectional shift register.

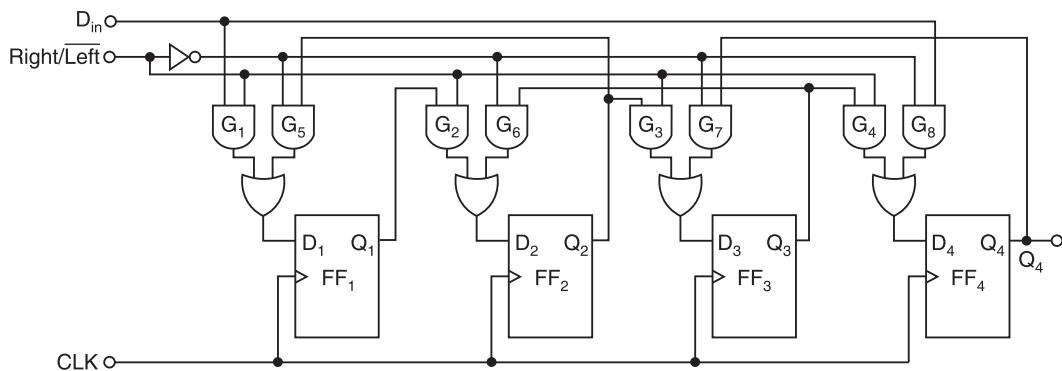


Figure 11.10 Logic diagram of a 4-bit bidirectional shift register.

## 11.10 UNIVERSAL SHIFT REGISTERS

A register capable of shifting in one direction only is a unidirectional shift register. One that can shift in both directions is a bidirectional shift register. If the register has both shifts and parallel load capabilities, it is referred to as a universal shift register. So a universal shift register is a bidirectional register, whose input can be either in serial form or in parallel form and whose output also can be either in serial form or in parallel form.

The most general shift register has the following capabilities:

1. A clear control to clear the register to 0.
2. A clock input to synchronize the operations.
3. A shift-right control to enable the shift-right operation and serial input and output lines associated with the shift-right.
4. A shift-left control to enable the shift-left operation and the serial input and output lines associated with the shift-left.
5. A parallel load control to enable a parallel transfer and the  $n$  input lines associated with the parallel transfer.

6.  $n$  parallel output lines.

7. A control state that leaves the information in the register unchanged in the presence of the clock.

A universal shift register can be realized using multiplexers. Figure 11.11 shows the logic diagram of a 4-bit universal shift register that has all the capabilities listed above. It consists of four D flip-flops and four multiplexers. The four multiplexers have two common selection inputs  $S_1$  and  $S_0$ . Input 0 in each multiplexer is selected when  $S_1S_0 = 00$ , input 1 is selected when  $S_1S_0 = 01$ , and input 2 is selected when  $S_1S_0 = 10$  and input 3 is selected when  $S_1S_0 = 11$ . The selection inputs control the mode of operation of the register according to the function entries in Table 11.1. When  $S_1S_0 = 0$ , the present value of the register is applied to the D inputs of flip-flops. This condition forms a path from the output of each flip-flop into the input of the same flip-flop. The next clock edge transfers into each flip-flop the binary value it held previously, and no change of state occurs. When  $S_1S_0 = 01$ , terminal 1 of the multiplexer inputs have a path to the D inputs of the flip-flops. This causes a shift-right operation, with the serial input transferred into flip-flop  $FF_4$ . When  $S_1S_0 = 10$ ,

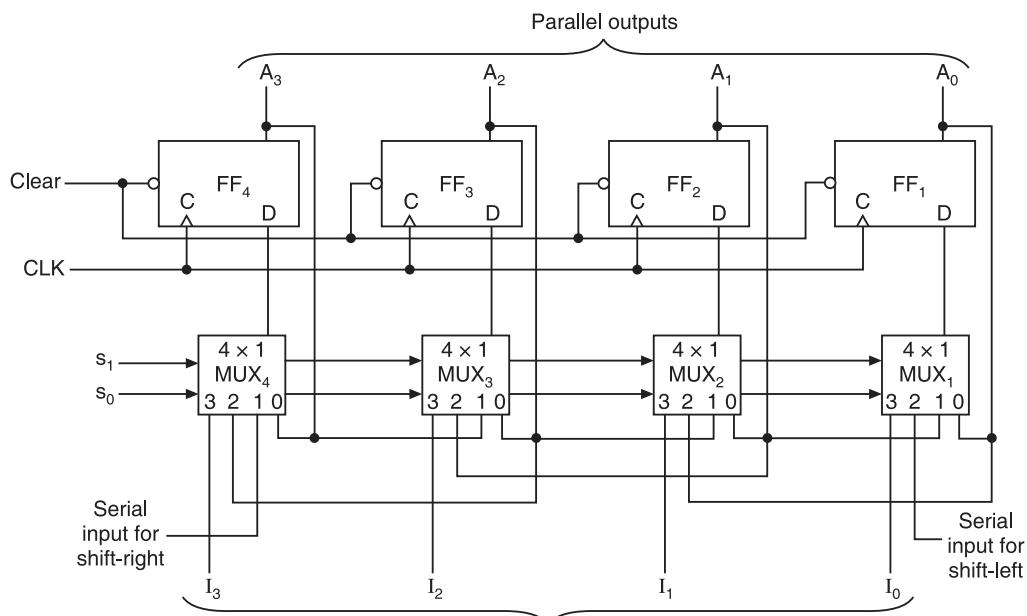


Figure 11.11 4-bit universal shift register.

Table 11.1 Function table for the register of Figure 11.11

Mode control		Register operation
$S_1$	$S_0$	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

a shift-left operation results with the other serial input going into flip-flop  $FF_1$ . Finally when  $S_1S_0 = 11$ , the binary information on the parallel input lines is transferred into the register simultaneously during the next clock edge.

### 11.11 DYNAMIC SHIFT REGISTERS

All the shift registers we have discussed till now are called *static shift registers*, because each one of the memory elements (i.e. FFs) used to build the register can retain the data bit indefinitely. So, once loaded, the contents of each element of the register remain the same. In a dynamic shift register, storage is accomplished by continually shifting the bits from one stage to the next and re-circulating the output of the last stage into the first stage. The data continually circulates through the register under the control of a clock. To obtain output, a serial output terminal must be accessed at a specific clock pulse; otherwise, the sequence of bits will not correspond to the data stored.

For example, if a 32-bit word is circulating through a 32-bit register; serial output must be given at multiples of 32 clock pulses. To store new data in such a register, the re-circulation path between the last stage and the first stage is intercepted and the new data is loaded serially into the first stage.

Since each stage of a dynamic shift register needs to retain a bit only for a time equal to one clock period, it is not necessary that each stage of the shift register be an FF. In particular, dynamic shift registers are constructed using dynamic inverters. The clock pulses cause bits to be transferred from one inverter stage to the next, by transferring the charge stored on the inherent capacitance of the MOS devices. This design requires the use of a clock, having certain minimum frequency to ensure that the capacitance does not fully discharge between the ‘refresh’ cycles. The main advantages of dynamic MOS registers are their small power consumption and simplicity, which permits a very large number of stages to be fabricated on a single IC. Their disadvantage is that all data transfer must be in serial form, which is much slower than parallel data transfer.

Dynamic MOS registers are widely used as memory devices in digital systems that operate on serial data. Because of their small power consumption and the inherent slowness of the serial systems, they are used in applications where power consumption and physical size are more important considerations than speed, such as in pocket calculators. In the context of memory applications, loading a register is called *writing* into it and taking an output from it is called *reading*.

Figure 11.12 shows the logic diagram and the truth table for the 2401 dual 1024-bit dynamic N-MOS shift register. It has a write/re-circulate ( $W/\bar{R}$ ) control that is used to govern whether new serial data is written (loaded) into the register or whether the existing data is re-circulated (stored) by the register. When  $W/\bar{R}$  is HIGH, AND gate 3 is enabled and the serial input is transferred through it to the register. When  $W/\bar{R}$  is LOW, AND gate 4 is enabled and a re-circulation path from output to input is completed. Serial data appears at the output, regardless of the state of  $W/\bar{R}$ . The circuit also has two active-LOW chip-select inputs, labelled  $\bar{CS}_x$  and  $\bar{CS}_y$ . Both  $\bar{CS}_x$  and  $\bar{CS}_y$  must be LOW in order to read or write data. Note, however, that re-circulation is independent of  $\bar{CS}_x$  and  $\bar{CS}_y$  and re-circulation is also independent of  $W/\bar{R}$  if at least one of the chip-selects is HIGH.

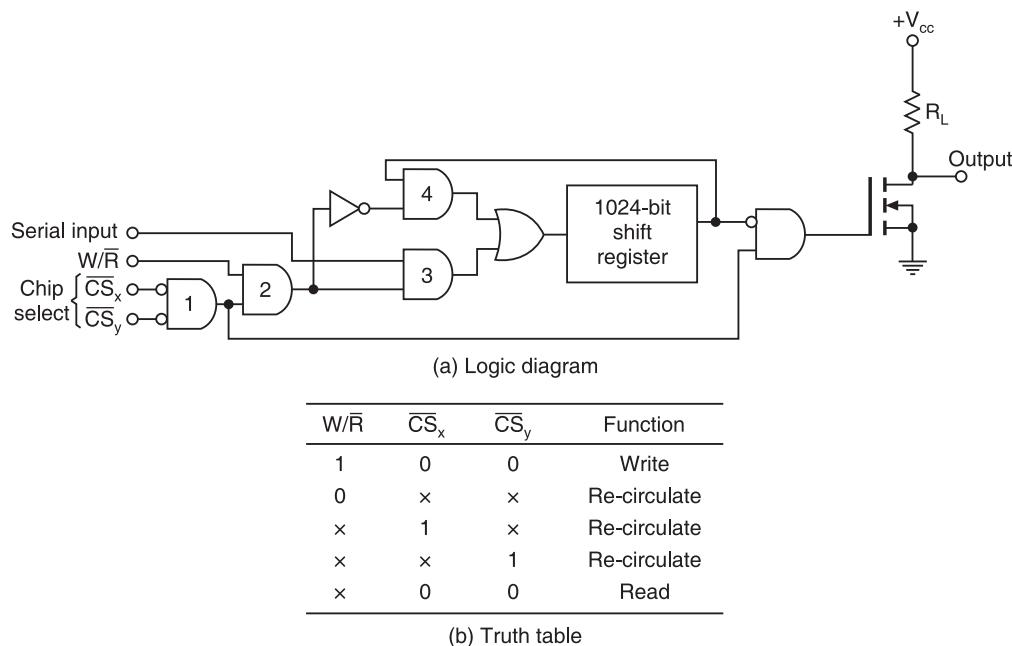


Figure 11.12 One-half of the 2401 dual 1024-bit dynamic NMOS shift register.

## 11.12 APPLICATIONS OF SHIFT REGISTERS

**Time delays:** In many digital systems, it is necessary to delay the transfer of data until such time as operations on other data have been completed, or to synchronize the arrival of data at a subsystem where it is processed with other data. A shift register can be used to delay the arrival of serial data by a specific number of clock pulses, since the number of stages corresponds to the number of clock pulses required to shift each bit completely through the register. The total time delay can be controlled by adjusting the clock frequency and by prescribing the number of stages in the register. In practice, the clock frequency is fixed and the total delay can be adjusted only by controlling the number of stages through which the data is passed. By using a serial-in, parallel-out register and by taking the serial output at any one of the intermediate stages, we have the flexibility to delay the output by any number of clock pulses equal to or less than the number of stages in the register. The arrangement shown in Figure 11.9 can be used to delay the data by 4 clock pulses.

**Serial/Parallel data conversion:** We know that data can be available either in serial form or in parallel form. Transfer of data in parallel form is much faster than that in serial form. Similarly, the processing of data is much faster when all the data bits are available simultaneously. For this reason, digital systems in which speed is an important consideration are designed to operate on data in parallel form. When large data is to be transmitted over long distances, transmitting data on parallel lines is costly and impracticable. It is convenient and economical to transmit data in serial form, since serial data transmission requires only one line. Shift registers are used for converting serial data to parallel form, so that a serial input can be processed by a parallel system and for converting parallel data to serial form, so that parallel data can be transmitted serially.

A serial-in, parallel-out, shift register can be used to perform serial-to-parallel conversion, and a parallel-in, serial-out, shift register can be used to perform parallel-to-serial conversion. A universal shift register can be used to perform both the serial-to-parallel and parallel-to-serial data conversions. A bidirectional shift register can be used to reverse the order of data. The arrangement shown in Figure 11.7 can be used for serial-to-parallel conversion of a 4-bit data. The arrangement shown in Figure 11.8 can be used for parallel-to-serial conversion of a 4-bit data.

**Ring counters:** Ring counters are constructed by modifying the serial-in, serial-out, shift registers. There are two types of ring counters—basic ring counter and Johnson counter. The basic ring counter can be obtained from a serial-in, serial-out, shift register by connecting the Q output of the last FF to the D input of the first FF. The Johnson counter can be obtained from a serial-in, serial-out, shift register by connecting the  $\bar{Q}$  output of the last FF to the D input of the first FF. Ring counter outputs can be used as a sequence of synchronizing pulses. The ring counter is a decimal counter. It is a divide-by- $N$  counter, where  $N$  is the number of stages. The keyboard encoder is an example of the application of a shift register used as a ring counter in conjunction with other devices.

Ring counters are dealt with in detail later in Chapter 12. Figures 12.57, and 12.58 (Section 12.9.1) illustrate the use of the shift register as a ring counter. Figures 12.61 and 12.62 (Section 12.9.2) illustrate the use of the shift register as a twisted ring counter.

**Universal asynchronous receiver transmitter (UART):** Computers and microprocessor-based systems often send and receive data in a parallel format. Frequently these systems must communicate with external devices that send and/or receive serial data. An interfacing device used to accomplish these conversions is the UART.

A UART is a specially designed integrated circuit that contains all the registers and synchronizing circuitry necessary to receive data in serial form and to convert and transmit it in parallel form and vice versa. Figure 11.13 shows the use of UART as an interfacing device.

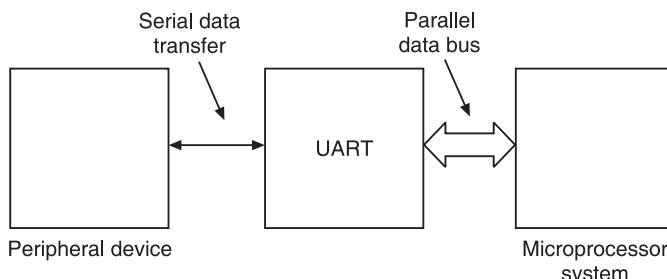


Figure 11.13 UART as an interfacing device.

### SHORT QUESTIONS AND ANSWERS

1. What do you mean by serial data?  
A. Multi-bit data is said to be in serial form when the data bits appear sequentially (one after the other in time) at a single terminal.
2. What do you mean by parallel data?  
A. Multi-bit data is said to be in parallel form when all the bits are available (accessible) simultaneously.

3. What do you mean by serial data transfer?
  - A. Serial data transfer is the transmission of one bit of data at a time from one device to another. Serial data must be transmitted under the synchronization of a clock, since the clock provides the means to specify the time at which each new bit is sampled.
4. What is a single bit register?
  - A. A flip-flop is called a single bit register. It can store only one bit of data a 0 or a 1.
5. What is a register?
  - A. A register is a set of flip-flops used to store binary data.
6. What do you mean by storage capacity of a register?
  - A. The storage capacity of a register is the number of bits (1s and 0s) of digital data it can retain.
7. What do you mean by loading a register?
  - A. Loading a register means setting or resetting the individual flip-flops, i.e. inputting data into the register so that their states correspond to the bits of data to be stored.
8. What are the types of loading the registers?
  - A. There are two types of loading the registers. They are (a) serial loading and (b) parallel loading.
9. What is serial loading?
  - A. Serial loading is one in which the data is transferred into the register in serial form, i.e. one bit at a time.
10. What is parallel loading?
  - A. Parallel loading is one in which the data is transferred into the register in parallel form, meaning that all the flip-flops are triggered into their new states at the same time. Parallel input requires that all the SET and/or RESET controls of every flip-flop be accessible.
11. How does a register output data?
  - A. A register may output data either in serial form or in parallel form.
12. What is serial output?
  - A. Serial output means that the data is transferred out of the register one bit at a time serially.
13. What is parallel output?
  - A. Parallel output means that the entire data stored in the register is available in parallel form and can be transferred out at the same time.
14. What are shift registers?
  - A. A number of flip-flops connected together such that the data may be shifted into and shifted out of them is called a shift register. In other words registers in which shifting of data takes place are called shift registers. They are used basically for the storage and transfer of digital data. Data may be shifted into and out of the register either in serial form or in parallel form.
15. What is the basic difference between a shift register and a counter?
  - A. The basic difference between a shift register and a counter is that, a shift register has no specified sequence of states except in some certain very specialized applications, whereas a counter has a specified sequence of states.
16. What are the applications of shift registers?
  - A. Shift registers are very important digital building blocks. They have innumerable applications. They are used to provide time delays, for serial/parallel data conversion, ring counters, etc. They also form the basis for some very important arithmetic operations.
17. What are buffer registers?
  - A. Buffer registers are registers which are simply used to store data.

## **618 FUNDAMENTALS OF DIGITAL CIRCUITS**

- 18.** How will you use a shift register to multiply or divide a binary number by 2?  
**A.** The binary number is to be stored in the shift register and then shifted towards left or right respectively by one bit position for multiplication or division by 2.
- 19.** What are the basic types of shift registers?  
**A.** There are four basic types of shift registers. They are: (a) serial-in, serial-out, (b) serial-in parallel-out, (c) parallel-in, serial-out and (d) parallel-in, parallel-out shift registers.
- 20.** What is a serial-in, serial-out shift register?  
**A.** A serial-in, serial-out shift register is one which accepts data serially, i.e. one bit at a time and also outputs data serially.
- 21.** What is a serial-in, parallel-out shift register?  
**A.** A serial-in, parallel-out shift register is one in which the data bits are entered into the register serially, i.e. one bit at a time but the data stored in the register is shifted out in parallel form, i.e. simultaneously.
- 22.** What is a parallel-in, serial-out shift register?  
**A.** A parallel-in, serial-out shift register is one in which the data bits are entered simultaneously into their respective stages on parallel lines, but the data bits are transferred out of the register serially, i.e. on a bit-by-bit basis on a single line.
- 23.** What is a parallel-in, parallel-out shift register?  
**A.** A parallel-in, parallel-out shift register is one in which the data is entered into the register in parallel form and also the data is taken out of the register in parallel form.
- 24.** What is a bidirectional shift register?  
**A.** A bidirectional shift register is one in which the data bits can be shifted from left to right as well as from right to left.
- 25.** What is a universal shift register?  
**A.** A universal shift register is a bidirectional register, whose input can be either in serial form or in parallel form and whose output also can be either in serial form or in parallel form.
- 26.** What is a static shift register?  
**A.** A static shift register is one in which each of the memory elements used to build the register can retain the data bit indefinitely. So once loaded, the contents of each element of the register remain the same. So a shift register using flip-flops is called a static shift register. In a static shift register data stored is stationary.
- 27.** What is a dynamic shift register?  
**A.** A dynamic shift register is one in which the storage is accomplished by continually shifting the bits from one stage to the next and recirculating the output of the last stage into the first stage. The data continually circulates through the register under the control of a clock. To obtain output, a serial output terminal must be accessed at a specific clock pulse, otherwise the sequence of bits will not correspond to the data stored.
- 28.** Where are dynamic MOS registers used?  
**A.** Dynamic MOS registers are widely used as memory devices in digital systems that operate on serial data. Because of their small power consumption and the inherent slowness of the serial systems, they are used in applications where power consumption and physical size are more important considerations than speed, such as in pocket calculators.
- 29.** Dynamic shift registers are made up of which devices?  
**A.** Dynamic shift registers are made up of MOS inverters.

**30.** What is UART?

- A. A UART is a specially designed integrated circuit that contains all the registers and synchronizing circuitry necessary to receive data in serial form and to convert and transmit it in parallel form and vice versa. It is an interfacing device.

**31.** What are the advantages and drawbacks of dynamic MOS registers?

- A. The main advantages of dynamic MOS registers are their small power consumption and simplicity, which permits a very large number of stages to be fabricated on a single IC. Their disadvantage is that all data transferred must be in serial form, which is much slower than parallel data transfer.

**REVIEW QUESTIONS**

1. Discuss the applications of shift registers.
2. With neat diagrams explain the working of the following types of shift registers.
 

(a) serial-in, serial-out	(b) serial-in, parallel-out
(c) parallel-in, serial-out	(d) parallel-in, parallel-out
(e) bidirectional	
3. Discuss the applications of shift registers.
4. Design a 4-bit universal shift register and draw the circuit with the given mode of operation table.

$S_1$	$S_0$	Operation
0	0	Parallel
0	1	Shift right
1	0	Shift left
1	1	Inhibit clock

**FILL IN THE BLANKS**

1. Data is said to be in serial form if the bits are available \_\_\_\_\_.
2. Data is said to be in parallel form if the bits are available \_\_\_\_\_.
3. Flip-flop can store \_\_\_\_\_.
4. A \_\_\_\_\_ is called a single bit register.
5. A set of flip-flops used to store binary data is called a \_\_\_\_\_.
6. The registers which are used to simply store the data are called \_\_\_\_\_.
7. A register in which shifting of data takes place is called a \_\_\_\_\_.
8. Transfer of data into and out of shift registers may take place either in \_\_\_\_\_ or in \_\_\_\_\_.
9. In a \_\_\_\_\_ shift register data is fed in and also shifted out serially.

## 620 FUNDAMENTALS OF DIGITAL CIRCUITS

10. In a \_\_\_\_\_ shift register data is fed in serially but taken out parallelly.
11. In a \_\_\_\_\_ shift register data is fed in, in parallel form but shifted out in serial form.
12. In a \_\_\_\_\_ shift register data is both fed in and shifted out in parallel form.
13. In a \_\_\_\_\_ shift register, data can be shifted from left-to-right or right-to-left.
14. In a \_\_\_\_\_ register, data can be shifted from left-to-right or right-to-left and also data can be shifted in or shifted out in serial form or in parallel form.
15. A shift register using flip-flops is called a \_\_\_\_\_ shift register.
16. In a \_\_\_\_\_ shift register, the data stored is stationary.
17. Dynamic shift registers are made up of \_\_\_\_\_.
18. \_\_\_\_\_ registers are used in pocket calculators.
19. Dynamic MOS registers are used in applications where power consumption and space are more important than \_\_\_\_\_.
20. In a \_\_\_\_\_ storage is accomplished by continually shifting data from one stage to the next and re-circulating through the output of the last stage into the first stage.
21. The main advantages of dynamic MOS registers are their \_\_\_\_\_ and \_\_\_\_\_.
22. The main disadvantage of dynamic MOS registers is that \_\_\_\_\_.
23. An interfacing device used to accomplish serial-to-parallel and parallel-to-serial data conversion is called \_\_\_\_\_.
24. \_\_\_\_\_ shift registers are made up of flip-flops which can retain data indefinitely.

### OBJECTIVE TYPE QUESTIONS

1. A shift register using flip-flops is called a
  - (a) dynamic shift register
  - (b) flip-flop shift register
  - (c) static shift register
  - (d) buffer shift register
2. Dynamic shift registers are made up of
  - (a) dynamic flip-flops
  - (b) MOS inverters
  - (c) MOS NAND gates
  - (d) CMOS inverters
3. A universal register
  - (a) accepts serial input
  - (b) accepts parallel input
  - (c) gives serial and parallel outputs
  - (d) is capable of all of the above

## VHDL PROGRAMS

## 1. VHDL PROGRAM FOR SERIAL-IN, SERIAL-OUT SHIFT REGISTER

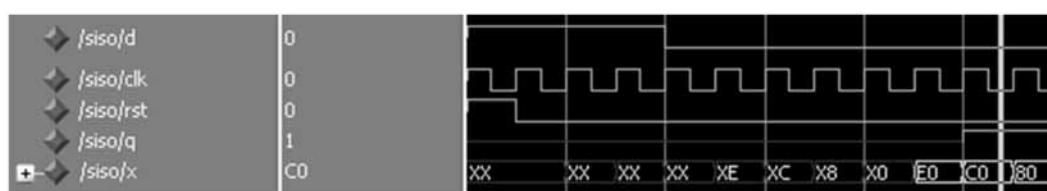
```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity serinout is
    Port ( d : in std_logic;
           clk : in std_logic;
           rst : in std_logic;
           q : out std_logic);
end serinout;

architecture Behavioral of serinout is
signal x : std_logic_vector(7 downto 0);
begin
process (d, clk, rst)
begin
    if (rst = '1') then
        q <= 'X';
    elsif (clk = '1' and clk'event) then
        x(0) <= d;
        x(1) <= x(0);
        x(2) <= x(1);
        x(3) <= x(2);
        x(4) <= x(3);
        x(5) <= x(4);
        x(6) <= x(5);
        x(7) <= x(6);
        q <= x(7);
    end if;
end process;
end Behavioral;

```

## SIMULATION OUTPUT:



**2. VHDL PROGRAM FOR SERIAL-IN, PARALLEL-OUT SHIFT REGISTER**

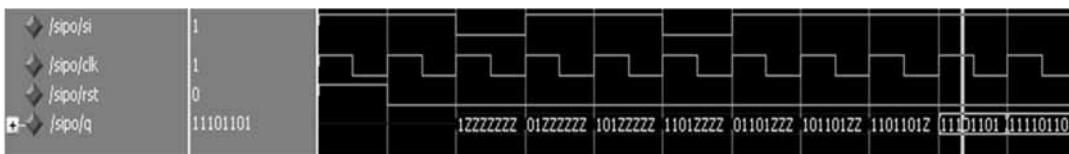
```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sipo is
    Port ( si : in std_logic;
            clk : in std_logic;
            rst : in std_logic;
            q : inout std_logic_vector(0 to 7));
end sipo;

architecture Behavioral of sipo is
begin
process (si,rst,clk)
begin
if(rst='1')then
q<="ZZZZZZZZ";
elsif(clk='1' and clk'event)then
q(0)<=si;
q(1)<=q(0);
q(2)<=q(1);
q(3)<=q(2);
q(4)<=q(3);
q(5)<=q(4);
q(6)<=q(5);
q(7)<=q(6);
end if;
end process;
end Behavioral;

```

**SIMULATION OUTPUT:****3. VHDL PROGRAM FOR PARALLEL-IN, SERIAL-OUT SHIFT REGISTER**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

```

```

entity piso is
    Port ( d : inout std_logic_vector(7 downto 0);
           a : in std_logic_vector(7 downto 0);
           clk : in std_logic;
           rst : in std_logic;
           i : in std_logic;
           q : inout std_logic_vector(7 downto 0));
end piso;

architecture Behavioral of piso is

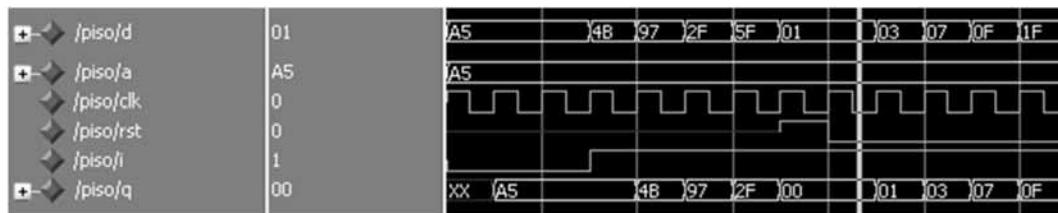
begin
process (d, a, clk, rst, i, q)
variable ibar, s1, s2, s3, s4, s5, s6,s7,s8,s9,s10,s11,s12,s13,
                     s14: std_logic;
begin
    ibar := not i;
    s1 := q(0) and i;
    s2 := ibar and a(1);
    s3 := q(1) and i;
    s4 := ibar and a(2);
    s5 := q(2) and i;
    s6 := ibar and a(3);
    s7 := q(3) and i;
    s8 := ibar and a(4);
    s9 := q(4) and i;
    s10 := ibar and a(5);
    s11 := q(5) and i;
    s12 := ibar and a(6);
    s13 := q(6) and i;
    s14 := ibar and a(7);
    d(0) <= a(0);
    d(1) <= s1 or s2;
    d(2) <= s3 or s4;
    d(3) <= s5 or s6;
    d(4) <= s7 or s8;
    d(5) <= s9 or s10;
    d(6) <= s11 or s12;
    d(7) <= s13 or s14;
    if (rst = '1') then
        q <= "00000000";
    else if (clk = '1' and clk'event) then
        q(0) <= d(0);
        q(1) <= d(1);
end if;
end process;
end Behavioral;

```

```

        q(2) <= d(2);
        q(3) <= d(3);
        q(4) <= d(4);
        q(5) <= d(5);
        q(6) <= d(6);
        q(7) <= d(7);
    end if;
end if;
end process;
end Behavioral;

```

**SIMULATION OUTPUT:****4. VHDL PROGRAM FOR PARALLEL-IN, PARALLEL-OUT SHIFT REGISTER**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity pipo is
    Port ( di : in std_logic_vector(0 to 7);
           clk : in std_logic;
           rst : in std_logic;
           do : out std_logic_vector(0 to 7));
end pipo;

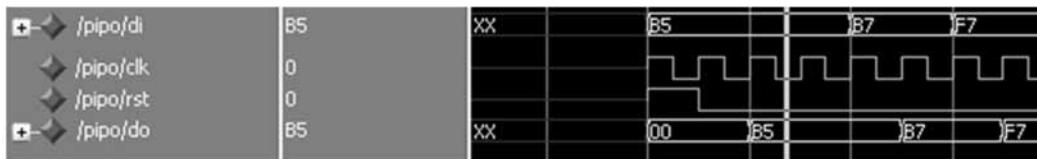
architecture Behavioral of pipo is
begin
    process(clk,rst,di)
begin
    if(rst='1')then
        do<="00000000";
    elsif(clk='1' and clk'event)then
        do(0)<=di(0);
        do(1)<=di(1);
        do(2)<=di(2);
        do(3)<=di(3);
        do(4)<=di(4);
    end if;
end process;
end Behavioral;

```

```

do(5)<=di(5);
do(6)<=di(6);
do(7)<=di(7);
end if;
end process;
end Behavioral;

```

**SIMULATION OUTPUT:****5. VHDL PROGRAM FOR UNIVERSAL SHIFT REGISTER**

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_bit.all;

entity univ_shiftreg is
port(clk, il, ir : in std_logic;
      s: in std_logic_vector(1 downto 0);
      i : in std_logic_vector(3 downto 0);
      q : out std_logic_vector(3 downto 0));
end univ_shiftreg;

architecture Behavioral of univ_shiftreg is
signal qtmp : bit_vector(3 downto 0);
begin
process(clk)
begin

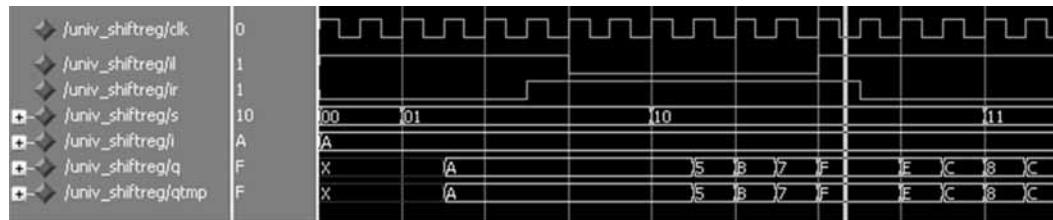
if (clk = '1' and clk'event) then
case s is
when "00" => qtmp <= qtmp;
when "01" => qtmp <= i;
when "10" => qtmp<=qtmp(2 downto 0) & ir;
when "11" => qtmp<=il & qtmp(3 downto 1);
when others => null;
end case;
end if;

```

```

end process;
q <= qtmp;
end Behavioral;

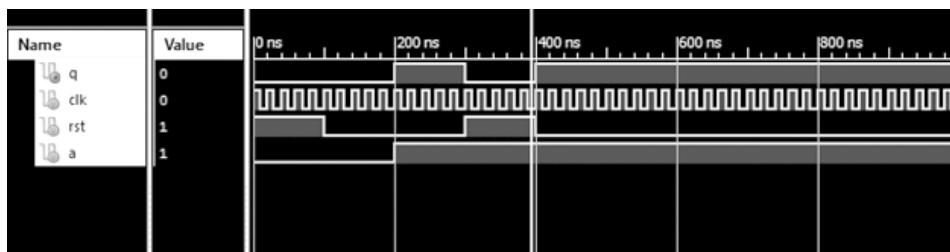
```

**SIMULATION OUTPUT:****VERILOG PROGRAMS****1. VERILOG PROGRAM FOR SERIAL-IN, SERIAL-OUT SHIFT REGISTER**

```

module siso(clk,rst,a,q);
input a;
input clk,rst;
output q;
reg q;
always@(posedge clk,posedge rst)
begin
if(rst==1'b1)
q<=1'b0;
else
q<=a;
end
endmodule

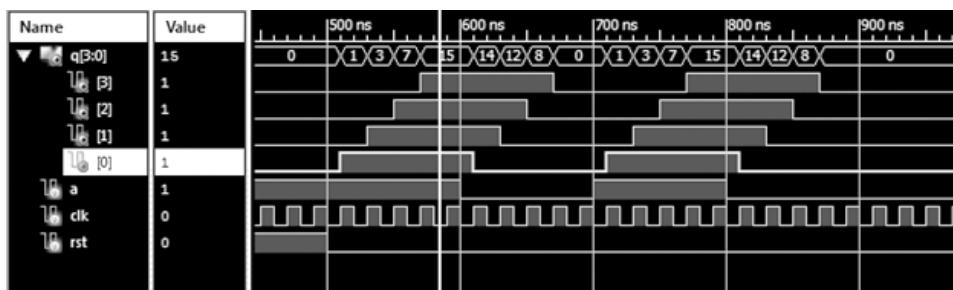
```

**SIMULATION OUTPUT:**

## 2. VERILOG PROGRAM FOR SERIAL-IN, PARALLEL-OUT SHIFT REGISTER

```
module sipo(a,clk,rst,q);
input clk,rst;
input a;
output [3:0]q;
wire [3:0]q;
reg [3:0]temp;
always@(posedge clk,posedge rst)
begin
if(rst==1'b1)
temp<=4'b0000;
else
begin
temp<=temp<<1'b1;
temp[0]<=a;
end
end
assign q=temp;
endmodule
```

### SIMULATION OUTPUT:



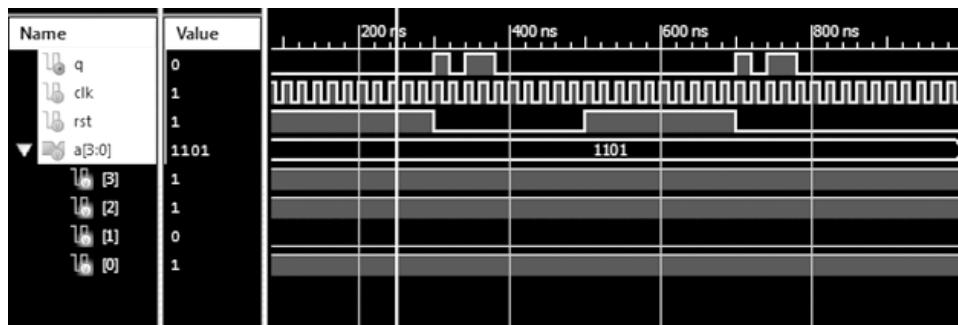
## 3. VERILOG PROGRAM FOR PARALLEL-IN, SERIAL-OUT SHIFT REGISTER

```
module piso(clk,rst,a,q);
input clk,rst;
input [3:0]a;
output q;
reg q;
reg [3:0]temp;
always@(posedge clk,posedge rst)
begin
if(rst==1'b1)
```

```

begin
q<=1'b0;
temp<=a;
end
else
begin
q<=temp[0];
temp <= temp>>1'b1;
end
end
endmodule

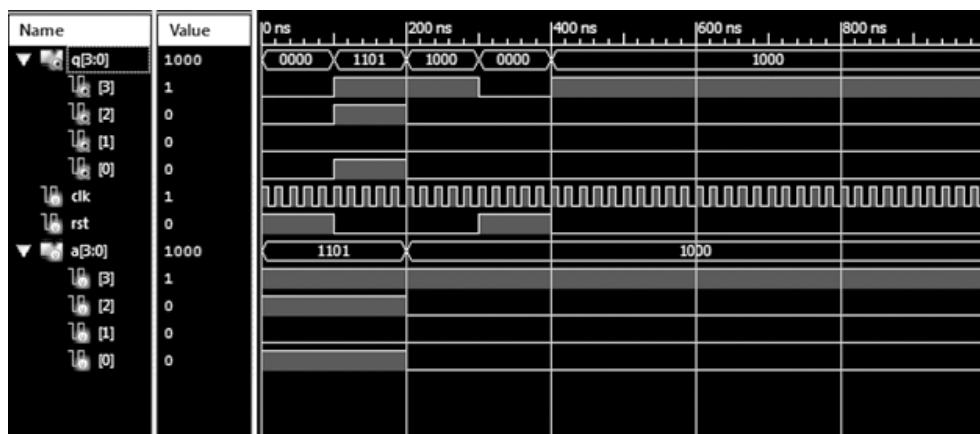
```

**SIMULATION OUTPUT:****4. VERILOG PROGRAM FOR PARALLEL-IN, PARALLEL-OUT SHIFT REGISTER**

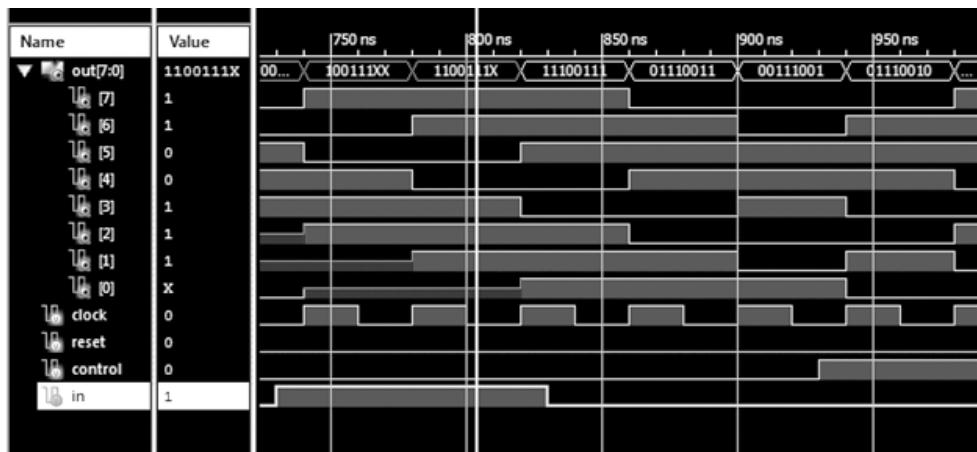
```

module pipo(clk,rst,a,q);
input clk,rst;
input [3:0]a;
output [3:0]q;
reg [3:0]q;
always@(posedge clk,posedge rst)
begin
if (rst==1'b1)
q<=4'b0000;
else
q<=a;
end
endmodule

```

**SIMULATION OUTPUT:****5. VERILOG PROGRAM IN BEHAVIORAL MODELING FOR 8-BIT UNIVERSAL SHIFT REGISTER**

```
module Universal_Shift_Register(
    input clock,
    input reset,
    input control,
    input in,
    output [7:0] out);
    reg [7:0] r_reg, r_next;
    always @ (posedge clock or posedge reset)
    begin
        if(reset)
            r_reg <= 0;
        else
            r_reg <= r_next;
    end
    always @ (*)
    begin
        if(control == 0) //shift right
            r_next = {in, r_reg[7:1]};
        else if(control==1) //shift left
            r_next = {r_reg[6:0], in};
        else
            r_next = r_reg; //default state stays the same
    end
    assign out = r_reg;
endmodule
```

**SIMULATION OUTPUT:**

# 12

## COUNTERS

### 12.1 INTRODUCTION

A digital counter is a set of flip-flops (FFs) whose states change in response to pulses applied at the input to the counter. The FFs are interconnected such that their combined state at any time is the binary equivalent of the total number of pulses that have occurred up to that time. Thus, as its name implies, a counter is used to count pulses. A counter can also be used as a frequency divider to obtain waveforms with frequencies that are specific fractions of the clock frequency. They are also used to perform the timing function as in digital watches, to create time delays, to produce non-sequential binary counts, to generate pulse trains, and to act as frequency counters, etc.

Counters may be *asynchronous* counters or *synchronous* counters. Asynchronous counters are also called *ripple counters*. The ripple counter is the simplest type of counter, the easiest to design and requires the least amount of hardware. In ripple counters, the FFs within the counter are not made to change the states at exactly the same time. This is because the FFs are not triggered simultaneously. The clock does not directly control the time at which every stage changes state. An asynchronous counter uses T FFs to perform a counting function. The actual hardware used is usually J-K FFs connected in toggle mode, i.e. with Js and Ks connected to logic 1. Even D FFs may be used here.

The asynchronous counter has a disadvantage, in so far as the unwanted spikes are concerned. This limitation is overcome in parallel counters. The asynchronous counter is called ripple counter because when the counter, for example, goes from 1111 to 0000, the first stage causes the second to flip, the second causes the third to flip, and the third causes the fourth to flip, and so on. In other

words, the transition of the first stage ripples through to the last stage. In doing so, many intermediate stages are briefly entered. If there is a gate that will AND during any state, a brief spike will be seen at the gate output every time the counter goes from 1111 to 0000. Ripple counters are also called *serial* or *series counters*. Synchronous counters are clocked such that each FF in the counter is triggered at the same time. This is accomplished by connecting the clock line to each stage of the counter. Synchronous counters are faster than asynchronous counters, because the propagation delay involved is less.

Comparison of synchronous and asynchronous counters is given in Table 12.1.

**Table 12.1** Synchronous versus asynchronous counters

Asynchronous counters	Synchronous counters
<ol style="list-style-type: none"> <li>1. In this type of counter FFs are connected in such a way that the output of first FF drives the clock for the second FF, the output of the second the clock of the third and so on.</li> <li>2. All the FFs are not clocked simultaneously.</li> <li>3. Design and implementation is very simple even for more number of states.</li> <li>4. Main drawback of these counters is their low speed as the clock is propagated through a number of FFs before it reaches the last FF.</li> </ol>	<ol style="list-style-type: none"> <li>1. In this type of counter there is no connection between the output of first FF and clock input of next FF and so on.</li> <li>2. All the FFs are clocked simultaneously.</li> <li>3. Design and implementation becomes tedious and complex as the number of states increases.</li> <li>4. Since clock is applied to all the FFs simultaneously the total propagation delay is equal to the propagation delay of only one FF. Hence they are faster.</li> </ol>

A counter may be an *up-counter* or a *down-counter*. An up-counter is a counter which counts in the upward direction, i.e. 0, 1, 2, 3,...,  $N$ . A down-counter is a counter which counts in the downward direction, i.e.  $N$ ,  $N - 1$ ,  $N - 2$ ,  $N - 3$ , ..., 1, 0. Each of the counts of the counter is called the *state* of the counter. The number of states through which the counter passes before returning to the starting state is called the *modulus* of the counter. Hence, the modulus of a counter is equal to the total number of distinct states (counts) including zero that a counter can store. In other words, the number of input pulses that causes the counter to reset to its initial count is called the modulus of the counter. Since a 2-bit counter has 4 states, it is called a mod-4 counter. It divides the input clock signal frequency by 4, therefore, it is also called a divide-by-4 counter. It requires two FFs. Similarly, a 3-bit counter uses 3 FFs and has  $2^3 = 8$  states. It divides the input clock frequency by  $2^3$ , i.e. 8. In general, an  $n$ -bit counter will have  $n$  FFs and  $2^n$  states, and divides the input frequency by  $2^n$ . Hence, it is a divide-by- $2^n$  counter.

A counter may have a shortened modulus. This type of counter does not utilize all the possible states. Some of the states are unutilized, i.e. invalid. The number of FFs required to construct a mod- $N$  counter equals the smallest  $n$  for which  $N \leq 2^n$ . A mod- $N$  counter divides the input frequency by  $N$ , hence, it is called a divide-by- $N$  counter. In an asynchronous counter, the invalid states are bypassed by providing a suitable feedback. In a synchronous counter, the invalid states are taken care of by treating the corresponding excitations as don't cares. The least significant bit (LSB) of any counter is that bit which changes most often. In ripple counters, the LSB is the Q output of the FF to which the external clock is applied.

A counter which goes through all the possible states before restarting is called the *full modulus counter*. A counter in which the maximum number of states can be changed is called the *variable modulus counter*. The final state of the counter sequence is called the *terminal count*.

**Lock-out:** In shortened-modulus counters, there may occur the problem of *lock-out*. Sometimes when the counter is switched on, or any time during counting, because of noise spikes, the counter may find itself in some unused (invalid) state. Subsequent clock pulses may cause the counter to move from one unused state to another unused state and the counter may never come to a valid state. So, the counter becomes useless. A counter whose unused states have this feature is said to suffer from the problem of lock-out. To ensure that, at 'start-up' the counter is in its initial state, external logic circuitry is provided which properly resets each FF. The logic circuitry for presetting the counter to its initial state can be provided either by obtaining an expression for reset/preset for the FFs or by modifying the design such that the counter goes from each invalid state to the initial state after the clock pulse. So, no don't cares are permitted in this design.

**Combination of modulo counters:** A single FF is a mod-2 counter. We can have a counter of any modulus by choosing an appropriate number of FFs and providing proper feedback. Counters of different mods can be combined to get another mod counter. For example, a mod-2 counter and a mod-5 counter can be combined to get a mod-10 counter; a mod-5 counter and a mod-4 counter can be combined to get a mod-20 counter, and so on. The connection between the individual counters may be a ripple connection, or the counters may be operated in synchronism with one another independently of whether the individual counters are ripple or synchronous. Further, we are at liberty to choose the order of the individual counters in a chain of counters. Such permutations will not change the modulus of the composite counter but may well make a substantive difference in the code in which the counter state is to be read.

## 12.2 ASYNCHRONOUS COUNTERS

### 12.2.1 Two-bit Ripple Up-counter Using Negative Edge-triggered Flip-Flops

The 2-bit up-counter counts in the order 0, 1, 2, 3, 0, 1, ..., i.e. 00, 01, 10, 11, 00, 01,..., etc. Figure 12.1 shows a 2-bit ripple up-counter, using negative edge-triggered J-K FFs, and its timing diagram. The counter is initially reset to 00. When the first clock pulse is applied, FF<sub>1</sub> toggles at the negative-going edge of this pulse, therefore, Q<sub>1</sub> goes from LOW to HIGH. This becomes a positive-going signal at the clock input of FF<sub>2</sub>. So, FF<sub>2</sub> is not affected, and hence, the state of the counter after one clock pulse is Q<sub>1</sub> = 1 and Q<sub>2</sub> = 0, i.e. 01. At the negative-going edge of the second clock pulse, FF<sub>1</sub> toggles. So, Q<sub>1</sub> changes from HIGH to LOW and this negative-going signal applied to CLK of FF<sub>2</sub> activates FF<sub>2</sub>, and hence, Q<sub>2</sub> goes from LOW to HIGH. Therefore, Q<sub>1</sub> = 0 and Q<sub>2</sub> = 1, i.e. 10 is the state of the counter after the second clock pulse. At the negative-going edge of the third clock pulse, FF<sub>1</sub> toggles. So Q<sub>1</sub> changes from a 0 to a 1. This becomes a positive-going signal to FF<sub>2</sub>, hence, FF<sub>2</sub> is not affected. Therefore, Q<sub>2</sub> = 1 and Q<sub>1</sub> = 1, i.e. 11 is the state of the counter after the third clock pulse. At the negative-going edge of the fourth clock pulse, FF<sub>1</sub> toggles. So, Q<sub>1</sub> goes from a 1 to a 0. This negative-going signal at Q<sub>1</sub> toggles FF<sub>2</sub>, hence, Q<sub>2</sub> also changes from a 1 to a 0. Therefore, Q<sub>2</sub> = 0 and Q<sub>1</sub> = 0, i.e. 00 is the state of the counter after the fourth clock pulse. For subsequent clock pulses, the counter goes through the same sequence of states. So, it acts as a mod-4 counter with Q<sub>1</sub> as the LSB and Q<sub>2</sub> as the MSB. The counting sequence is thus 00, 01, 10, 11, 00, 01,..., etc.

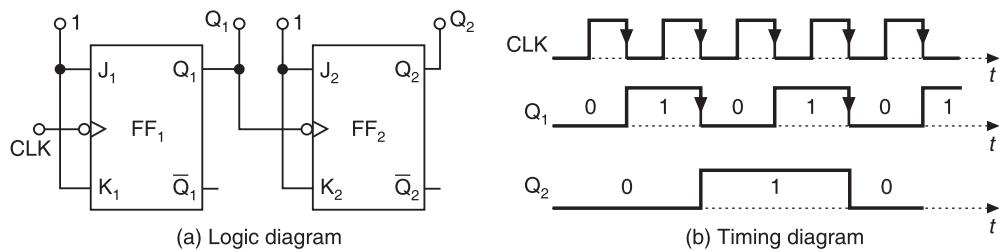


Figure 12.1 Asynchronous 2-bit up-counter using negative edge-triggered flip-flops.

### 12.2.2 Two-bit Ripple Down-counter Using Negative Edge-triggered Flip-Flops

A 2-bit down-counter counts in the order  $0, 3, 2, 1, 0, 3, \dots$ , i.e.  $00, 11, 10, 01, 00, 11, \dots$ , etc. Figure 12.2 shows a 2-bit ripple down-counter, using negative-edge triggered J-K FFs, and its timing diagram.

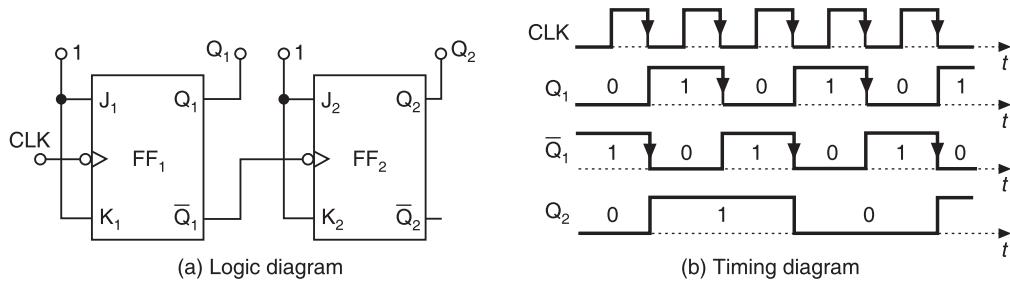


Figure 12.2 Asynchronous 2-bit down-counter using negative edge-triggered flip-flops.

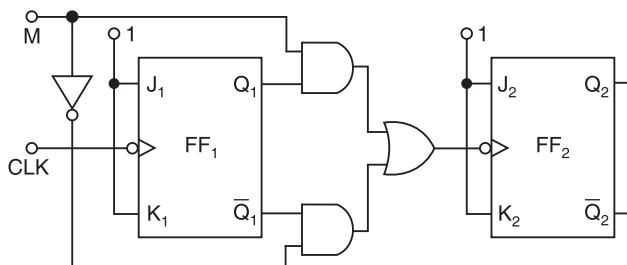
For down counting,  $\bar{Q}_1$  of  $FF_1$  is connected to the clock of  $FF_2$ . Let initially all the FFs be reset, i.e. let the count be 00. At the negative-going edge of the first clock pulse,  $FF_1$  toggles, so,  $Q_1$  goes from a 0 to a 1 and  $\bar{Q}_1$  goes from a 1 to a 0. This negative-going signal at  $\bar{Q}_1$  applied to the clock input of  $FF_2$ , toggles  $FF_2$  and, therefore,  $Q_2$  goes from a 0 to a 1. So, after one clock pulse  $Q_2 = 1$  and  $Q_1 = 1$ , i.e. the state of the counter is 11. At the negative-going edge of the second clock pulse,  $Q_1$  changes from a 1 to a 0 and  $\bar{Q}_1$  from a 0 to a 1. This positive-going signal at  $\bar{Q}_1$  does not affect  $FF_2$  and, therefore,  $Q_2$  remains at a 1. Hence, the state of the counter after the second clock pulse is 10. At the negative-going edge of the third clock pulse,  $FF_1$  toggles. So,  $Q_1$  goes from a 0 to a 1 and  $\bar{Q}_1$  from a 1 to a 0. This negative-going signal at  $\bar{Q}_1$  toggles  $FF_2$  and, so,  $Q_2$  changes from a 1 to a 0. Hence, the state of the counter after the third clock pulse is 01. At the negative-going edge of the fourth clock pulse,  $FF_1$  toggles. So,  $Q_1$  goes from a 1 to a 0 and  $\bar{Q}_1$  from a 0 to a 1. This positive-going signal at  $\bar{Q}_1$  does not affect  $FF_2$ . So,  $Q_2$  remains at a 0. Hence, the state of the counter after the fourth clock pulse is 00. For subsequent clock pulses the counter goes through the same sequence of states, i.e. the counter counts in the order 00, 11, 10, 01, 00, and 11 ...

### 12.2.3 Two-bit Ripple Up-down Counter Using Negative Edge-triggered Flip-Flops

As the name indicates an up-down counter is a counter which can count both in upward and downward directions (Figure 12.3). An up-down counter is also called a forward/backward counter

or a bidirectional counter. So, a control signal or a mode signal M is required to choose the direction of count. When M = 1 for up counting,  $Q_1$  is transmitted to clock of FF<sub>2</sub> and when M = 0 for down counting,  $\bar{Q}_1$  is transmitted to clock of FF<sub>2</sub>. This is achieved by using two AND gates and one OR gate as shown in Figure 12.3. The external clock signal is applied to FF<sub>1</sub>.

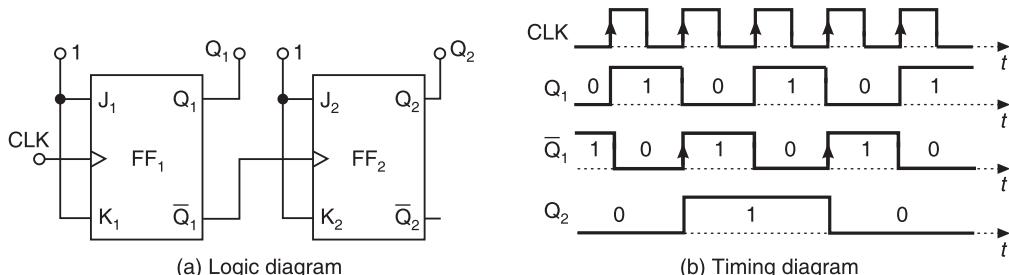
$$\text{Clock signal to FF}_2 = (Q_1 \cdot \text{Up}) + (\bar{Q}_1 \cdot \text{Down}) = Q_1 M + \bar{Q}_1 \bar{M}$$



**Figure 12.3** Asynchronous 2-bit up-down counter using negative edge-triggered flip-flops.

#### 12.2.4 Two-bit Ripple Up-counter Using Positive Edge-triggered Flip-Flops

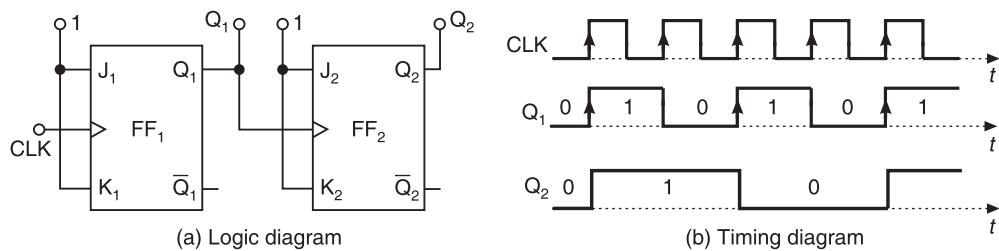
A 2-bit ripple up-counter, using positive edge-triggered J-K FFs, and its timing diagram are shown in Figure 12.4. The  $\bar{Q}_1$  output of the first FF is connected to the clock of FF<sub>2</sub>. The external clock signal is applied to the first flip-flop FF<sub>1</sub>. The FF<sub>1</sub> toggles at the positive-going edge of each clock pulse and FF<sub>2</sub> toggles whenever  $\bar{Q}_1$  changes from a 0 to a 1. State transitions occur at the positive-going edges of the clock pulses. The counting sequence is 00, 01, 10, 11, 00, 01, ..., etc.



**Figure 12.4** Asynchronous 2-bit up-counter using positive-edge triggered J-K flip-flops.

#### 12.2.5 Two-bit Ripple Down-counter Using Positive Edge-triggered Flip-Flops

A 2-bit ripple down-counter, using positive edge-triggered J-K FFs, and its timing diagram are shown in Figure 12.5. The Q<sub>1</sub> output of the first FF is connected to the clock of FF<sub>2</sub>. The external clock signal is applied to FF<sub>1</sub>. The FF<sub>1</sub> toggles at the positive-going edge of each clock pulse. The FF<sub>2</sub> toggles whenever Q<sub>1</sub> changes from a 0 to a 1. The counting sequence is 00, 11, 10, 01, 00, 11..., etc.

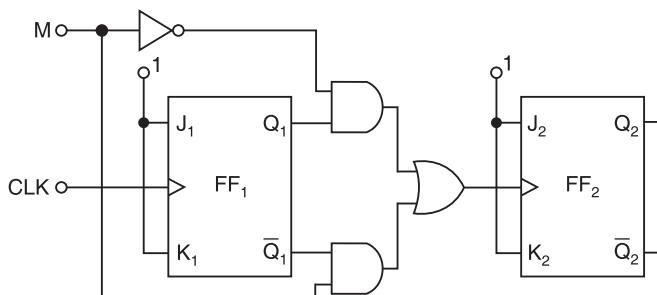


**Figure 12.5** Asynchronous 2-bit down-counter using positive edge-triggered J-K flip-flops.

### 12.2.6 Two-bit Ripple Up/Down Counter Using Positive Edge-triggered Flip-Flops

Figure 12.6 shows a 2-bit ripple up/down counter using positive edge-triggered J-K FFs. When  $M = 1$  for up counting,  $\bar{Q}_1$  is transmitted to the clock of  $FF_2$  and when  $M = 0$  for down counting,  $Q_1$  is transmitted to the clock of  $FF_2$ . This is achieved by using two AND gates and one OR gate as shown in Figure 12.6.

$$\text{Clock signal to } FF_2 = (\bar{Q}_1 \cdot \text{Up}) + (Q_1 \cdot \text{Down}) = \bar{Q}_1 M + Q_1 \bar{M}$$



**Figure 12.6** Logic diagram of a two-bit ripple up/down counter using positive edge-triggered flip-flops.

## 12.3 DESIGN OF ASYNCHRONOUS COUNTERS

To design an asynchronous counter, first write the counting sequence, then tabulate the values of reset signal R for various states of the counter and obtain the minimal expression for R or  $\bar{R}$  using K-map or any other method. Provide a feedback such that R or  $\bar{R}$  resets all the FFs after the desired count.

### 12.3.1 Design of a Mod-6 Asynchronous Counter Using T FFs

A mod-6 counter has six stable states 000, 001, 010, 011, 100, and 101. When the sixth clock pulse is applied, the counter temporarily goes to 110 state, but immediately resets to 000 because of the feedback provided. It is a ‘divide-by-6 counter’, in the sense that it divides the input clock frequency by 6. It requires three FFs, because the smallest value of  $n$  satisfying the condition  $N \leq 2^n$  is  $n = 3$ ; three FFs can have eight possible states, out of which only six are utilized and the remaining two states 110 and 111, are invalid. If initially the counter is in 000 state, then

after the first clock pulse it goes to 001, after the second clock pulse, it goes to 010, and so on. After the sixth clock pulse, it goes to 000.

For the design, write a truth table (Figure 12.7c) with the present state outputs  $Q_3$ ,  $Q_2$  and  $Q_1$  as the variables, and reset R as the output and obtain an expression for R in terms of  $Q_3$ ,  $Q_2$  and  $Q_1$ . That decides the feedback to be provided. From the truth table,  $R = Q_3 Q_2$ . For active-LOW reset,  $\bar{R}$  is used. The reset pulse is of very short duration, of the order of nanoseconds and it is equal to the propagation delay time of the NAND gate used. The expression for R can also be determined as follows.

$$R = 0 \text{ for } 000 \text{ to } 101, R = 1 \text{ for } 110, \text{ and } R = X \text{ for } 111$$

Therefore,

$$R = Q_3 Q_2 \bar{Q}_1 + Q_3 Q_2 Q_1 = Q_3 Q_2$$

The logic diagram, the timing diagram, and the table for R of a mod-6 counter are all shown in Figure 12.7. From the timing diagram it is seen that a glitch appears in the waveform of  $Q_2$ .

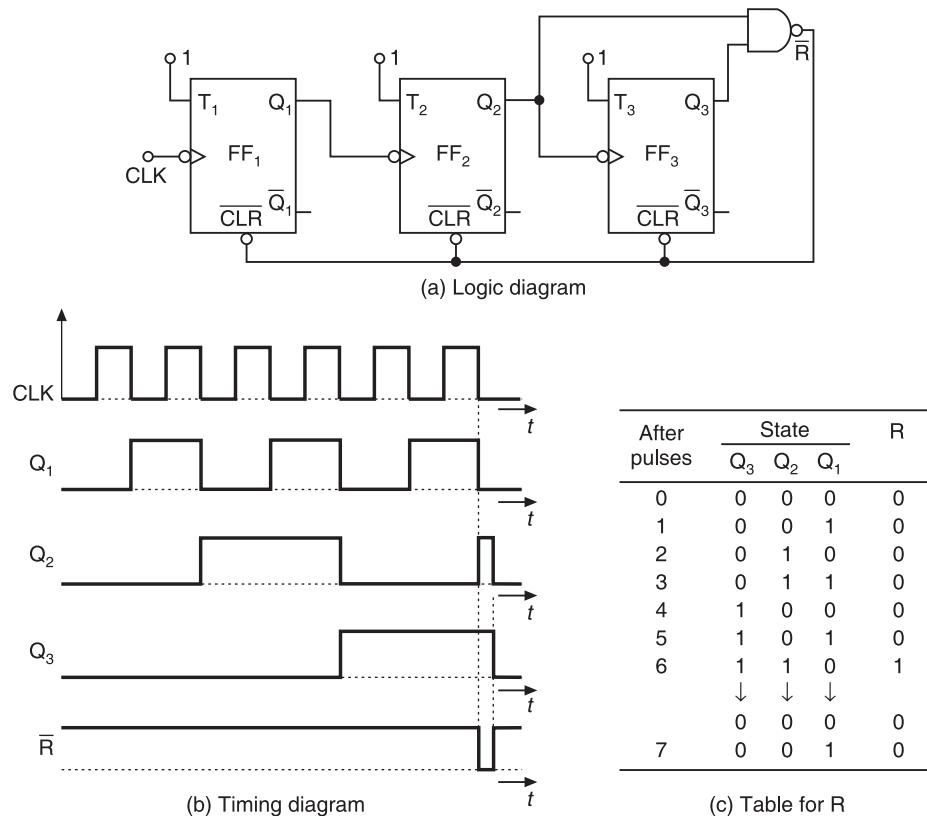


Figure 12.7 Asynchronous mod-6 counter using T flip-flops.

### 12.3.2 Design of a Mod-10 Asynchronous Counter Using T FFs

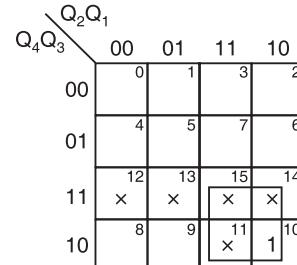
A mod-10 counter is a decade counter. It is also called a BCD counter or a divide-by-10 counter. It requires four FFs (the smallest value of  $n$  satisfying the condition  $10 \leq 2^n$ , is  $n = 4$ ). So, there are

16 possible states, out of which ten are valid and the remaining six are invalid. The counter has ten stable states, 0000 through 1001, i.e. it counts from 0 to 9. The initial state is 0000 and after nine clock pulses it goes to 1001. When the tenth clock pulse is applied, the counter goes to state 1010 temporarily, but because of the feedback provided, it resets to initial state 0000. So, there will be a glitch in the waveform of  $Q_2$ . The state 1010 is a temporary state for which the reset signal  $R = 1$ ,  $R = 0$  for 0000 to 1001, and  $R = X$  (don't care) for 1011 to 1111.

The count table and the K-map for reset are shown in Figures 12.8a and 12.8b, respectively. From the K-map,  $R = Q_4 Q_2$ . So, feedback is provided from second and fourth FFs. For active-HIGH reset,  $\overline{Q_4} \overline{Q_2}$  is applied to the CLEAR terminal. For active-LOW reset,  $\overline{Q_4} \overline{Q_2}$  is connected to CLR of all the FFs. The logic diagram of the decade counter is shown in Figure 12.8c.

After pulses	Count			
	$Q_4$	$Q_3$	$Q_2$	$Q_1$
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0

(a) Count table



(b) K-Map

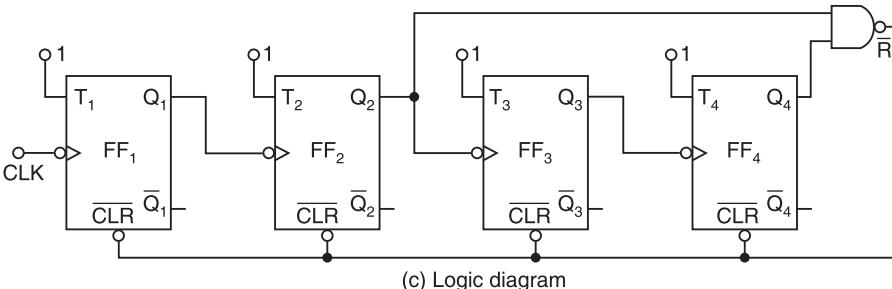
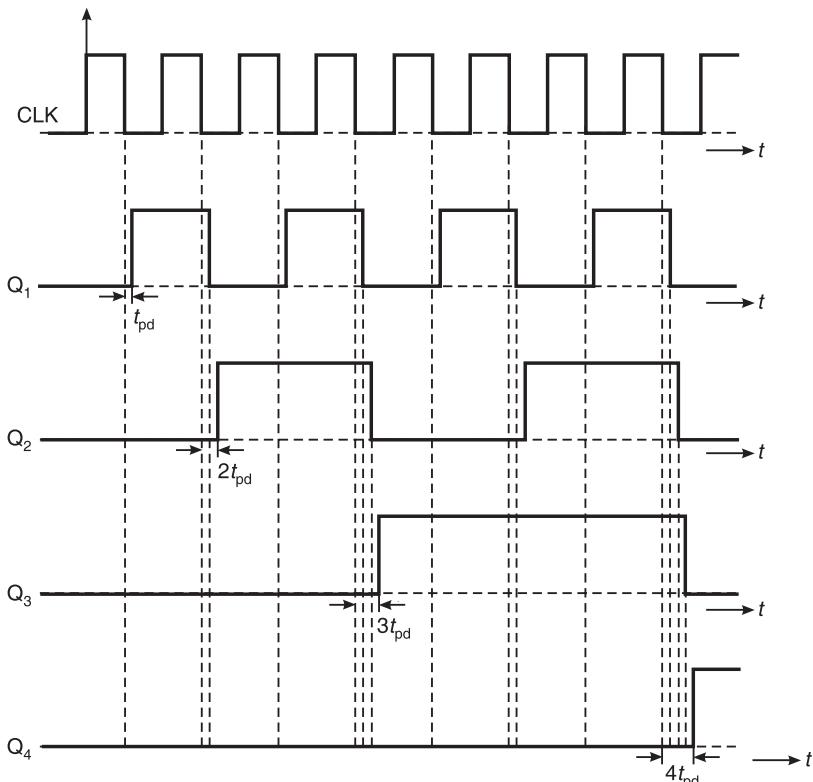


Figure 12.8 Asynchronous mod-10 counter using T flip-flops.

## 12.4 EFFECTS OF PROPAGATION DELAY IN RIPPLE COUNTERS

In ripple counters, each FF is toggled by the changing state of the preceding FF. So, no FF can change state until after the propagation delay of the FF that precedes it, i.e. until all preceding FFs complete their transitions. This delay accumulates as we proceed through additional stages. This will have a detrimental effect on the operation of the ripple counter. For a decade counter or a 4-bit counter, all the FFs change states when the count goes from 0111 to 1000. So, after the eighth clock pulse is applied,  $Q_1$  will not change state instantaneously, but goes from a 1 to a 0 after a

propagation delay of  $t_{pd}$ .  $Q_2$  changes state after another propagation delay time of  $2t_{pd}$ .  $Q_3$  changes state after  $3t_{pd}$  and  $Q_4$  changes state after  $4t_{pd}$ . The timing diagram considering propagation delays is shown in Figure 12.9.



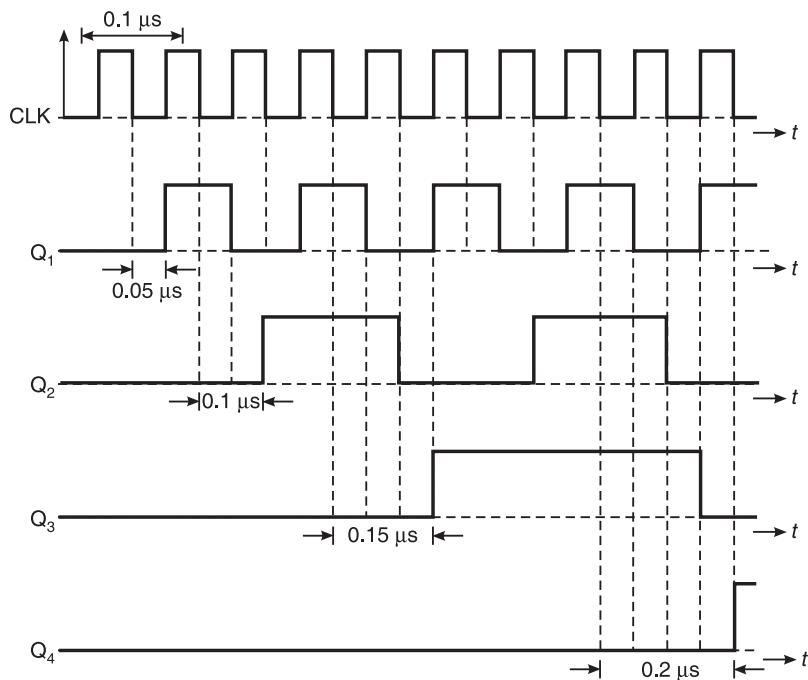
**Figure 12.9** Timing diagram considering propagation delay (no skipping of states).

If the propagation delay is large and clock frequency high, i.e. clock period is low, then there is a possibility that the first FF responds to the new clock pulse before the previous clock pulse has effected transition of the fourth FF. When the last FF finally responds, the counter may read 1001 having skipped 1000. Thus the count goes directly from 0111 to 1001. If the clock frequency is so high that it is possible for the clock pulse to change the state of the first stage, before the state changes caused by the previous clock pulse have rippled through to the last stage, then a count will be skipped. Thus it is obvious that propagation delays in the FFs of a ripple counter impose a limit on the frequency at which the counter can be clocked.

If  $T_C$  is the period of the clock pulse,  $n$  the number of stages and  $t_{pd}$  the propagation delay in each stage, then the clock frequency  $f_C$  is constrained by

$$\frac{1}{T_C} = f_C < \frac{1}{nt_{pd}}$$

Suppose  $T_C = 0.1 \mu\text{s}$ , i.e.  $f_C = 1/(0.1 \mu\text{s}) = 10 \text{ MHz}$  and suppose  $t_{pd} = 0.05 \mu\text{s}$ , the timing diagram would then be as shown in Figure 12.10. Here skipping of states occur. The count 1000 is not reached at all as shown in Figure 12.10.

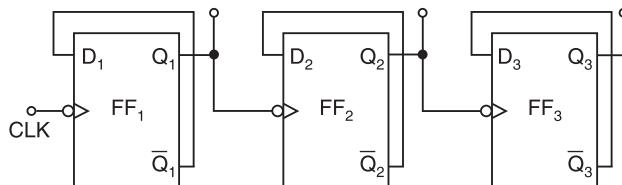


**Figure 12.10** Timing diagram of a ripple counter showing skipping of states.

**EXAMPLE 12.1** Implement a 3-bit ripple counter using D FFs.

**Solution**

For ripple counters, the FFs used must be in toggle mode. The D FFs may be used in toggle mode by connecting the  $\bar{Q}$  of each FF to its D terminal. The 3-bit ripple counter using D FFs is shown in Figure 12.11.



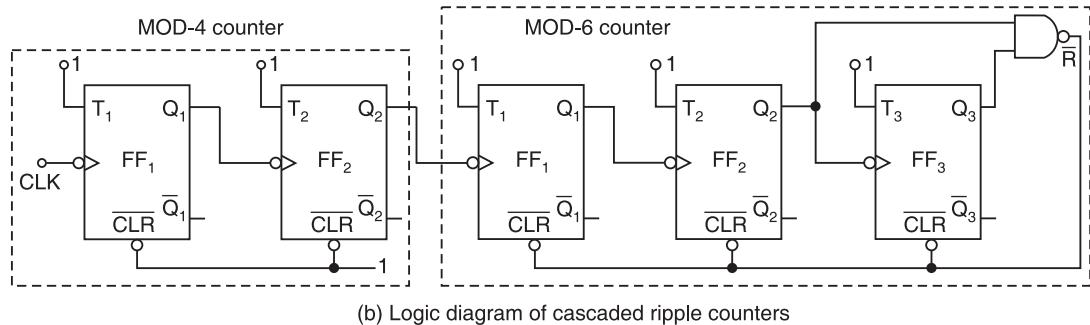
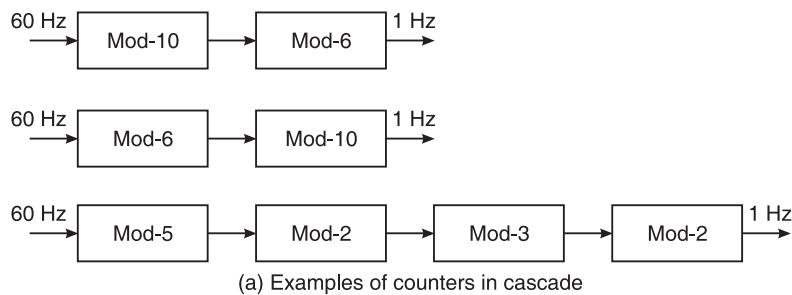
**Figure 12.11** Example 12.1: Logic diagram of a 3-bit asynchronous counter using D flip-flops.

#### 12.4.1 Cascading of Ripple Counters

Ripple counters can be connected in cascade to increase the modulus of the counter. A mod-M and a mod-N counter in cascade give a mod-MN counter. While cascading, the most significant stage of the first counter is connected to the toggling stage of the second counter. Some examples of counters in cascade are shown in Figure 12.12(a).

The order of cascading does not affect the frequency division, however, the duty cycle of the most significant output may depend on the order in which the counters are cascaded. Mod counters

are often constructed by cascading lower modulus counters because of the availability of certain standard modulus counters in IC form. Figure 12.12(b) shows the logic diagram of cascaded ripple counters.



**Figure 12.12** Ripple counters.

**EXAMPLE 12.2** A binary ripple counter is required to count up to  $16,383_{10}$ . How many FFs are required? If the clock frequency is 8.192 MHz, what is the frequency at the output of the MSB?

### Solution

The number of FFs  $n$  is to be selected such that the number of states  $N \leq 2^n$ . With  $n$  FFs, the largest count possible is  $2^n - 1$ . Therefore,

$$2^n - 1 = 16,383$$

or

$$n = \log_2 16,384 = 14$$

So, the number of FFs required is 14.

Frequency at the output of last stage is

$$f_{14} = \frac{f_C}{2^{14}} = \frac{8.192 \text{ MHz}}{16,384} = 500 \text{ Hz}$$

**EXAMPLE 12.3** For what minimum value of propagation delay in each FF will a 10-bit ripple counter skip a count when it is clocked at 10 MHz?

### Solution

For a state change to ripple through all  $n$  stages,  $T_C = nt_{pd}$ . Therefore, the clock frequency is constrained by

$$\frac{1}{T_C} = f_C = \frac{1}{nt_{pd}}$$

Therefore,

$$t_{pd} = \frac{1}{nf_C}$$

or

$$t_{pd}(\text{min}) = \frac{1}{10 \times 10 \times 10^6} = 10 \text{ ns}$$

## 12.5 SYNCHRONOUS COUNTERS

Asynchronous counters are serial counters. They are slow because each FF can change state only if all the preceding FFs have changed their state. The propagation delay thus gets accumulated, and so causes problems. If the clock frequency is very high, the asynchronous counter may skip some of the states and, therefore, malfunction. This problem is overcome in synchronous or parallel counters. Synchronous counters are counters in which all the FFs are triggered simultaneously (in parallel) by the clock-input pulses. Whether a FF toggles or not depends on the FF's inputs (J, K, or D, or T, or S, R). Since all the FFs change state simultaneously in synchronization with the clock pulse, the propagation delays of FFs do not add together (as in ripple counters) to produce the overall delay. In fact, the propagation delay of a synchronous counter is equal to the propagation delay of just one FF plus the propagation delay of the gates involved. So, the synchronous counters can operate at much higher frequencies than those that can be used in asynchronous counters.

Synchronous counters have the advantages of high speed and less severe decoding problems, but the disadvantage of having more circuitry than that of asynchronous counters. Many synchronous (parallel) counters that are available as ICs are designed to be presettable, i.e. they can be preset to any desired starting count either asynchronously or synchronously. This presetting operation is also referred to as *loading* the counter.

### 12.5.1 Design of Synchronous Counters

For a systematic design of synchronous counters, the following procedure is used.

*Step 1. Number of flip-flops:* Based on the description of the problem, determine the required number  $n$  of the FFs—the smallest value of  $n$  is such that the number of states  $N \leq 2^n$ —and the desired counting sequence.

*Step 2. State diagram:* Draw the state diagram showing all the possible states. A state diagram, which can also be called the transition diagram, is a graphical means of depicting the sequence of states through which the counter progresses. In case the counter goes to a particular state from the invalid states on the next clock pulse, the same can also be included in the state diagram.

*Step 3. Choice of flip-flops and excitation table:* Select the type of flip-flops to be used and write the excitation table. An excitation table is a table that lists the present state (PS), the next state (NS) and the required excitations.

*Step 4. Minimal expressions for excitations:* Obtain the minimal expressions for the excitations of the FFs using the K-maps drawn for the excitations of the flip-flops in terms of the present states and inputs.

*Step 5. Logic diagram:* Draw a logic diagram based on the minimal expressions.

If the synchronous counter is a shortened-modulus counter it may suffer from the problem of lock-out. That is, the counter may not self-start. A self-starting counter is one that will eventually enter its proper sequence of states regardless of its initial state. The counter can be made self-starting by so designing it that it goes to a particular state whenever it enters an invalid state. The same procedure can be used for counters of any number of bits and any arbitrary sequence. The only restriction on the sequence is that, it cannot contain the same state more than once within one complete cycle before repeating itself.

In the case of an up-down counter the state diagram shows the relationship between the present state, the input, and the next state of the counter. In the case of an up-counter or down-counter the state diagram shows only the relationship between the present state and the next state because only one type of input is given.

The excitation tables of various flip-flops used in the counters are shown in Table 12.2.

**Table 12.2** Excitation tables

PS		NS		Required inputs	
Q <sub>n</sub>	Q <sub>n+1</sub>	S	R		
0	0	0	×		
0	1	1	0		
1	0	0	1		
1	1	×	0		

(a) S-R FF excitation table

PS		NS		Required inputs	
Q <sub>n</sub>	Q <sub>n+1</sub>	J	K		
0	0	0	×		
0	1	1	×		
1	0	×	1		
1	1	×	0		

(b) J-K FF excitation table

PS		NS		Required input	
Q <sub>n</sub>	Q <sub>n+1</sub>	D			
0	0	0			
0	1	1			
1	0	0			
1	1	1			

(c) D FF excitation table

PS		NS		Required input	
Q <sub>n</sub>	Q <sub>n+1</sub>	T			
0	0	0			
0	1	1			
1	0	1			
1	1	0			

(d) T FF excitation table

### 12.5.2 Design of a Synchronous 3-bit Up-down Counter Using J-K FFs

*Step 1. Determine the number of flip-flops required:* A 3-bit counter requires three FFs. It has 8 states (000, 001, 010, 011, 100, 101, 110, 111) and all the states are valid. Hence no don't cares. For selecting up and down modes, a control or mode signal M is required. Let us say it counts up when the mode signal M = 1 and counts down when M = 0. The clock signal is applied to all the FFs simultaneously.

*Step 2. Draw the state diagram:* The state diagram of the 3-bit up-down counter is drawn as shown in Figure 12.13a.

*Step 3. Select the type of flip-flops and draw the excitation table:* JK flip-flops are selected and the excitation table of a 3-bit up-down counter using JK flip-flops is drawn as shown in Figure 12.13b.

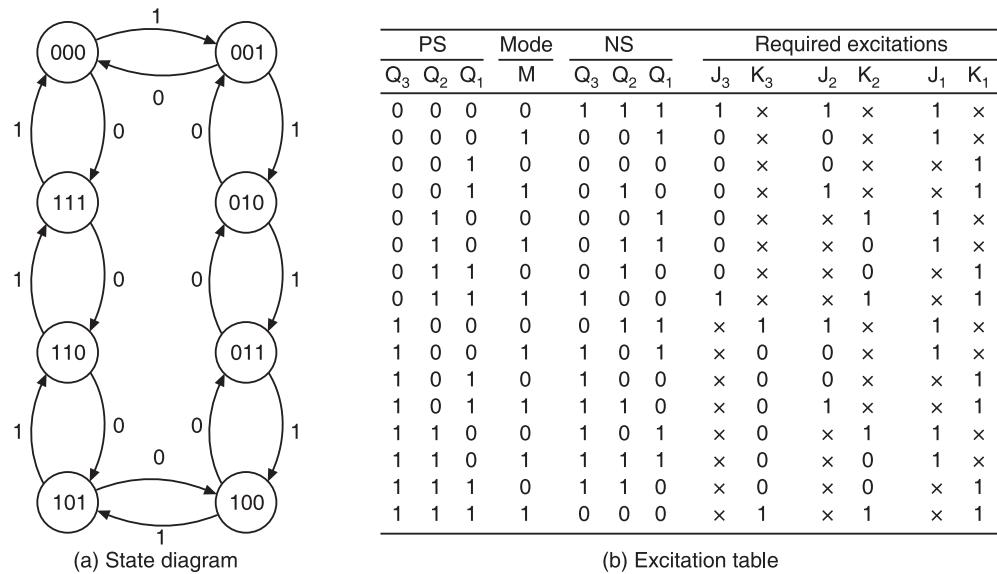


Figure 12.13 Synchronous 3-bit up-down counter.

*Step 4. Obtain the minimal expressions:* From the excitation table we can conclude that  $J_1 = 1$  and  $K_1 = 1$ , because all the entries for  $J_1$  and  $K_1$  are either X or 1. The K-maps for  $J_3$ ,  $K_3$ ,  $J_2$  and  $K_2$  based on the excitation table and the minimal expressions obtained from them are shown in Figure 12.14.

*Step 5. Draw the logic diagram:* A logic diagram using those minimal expressions can be drawn as shown in Figure 12.15.

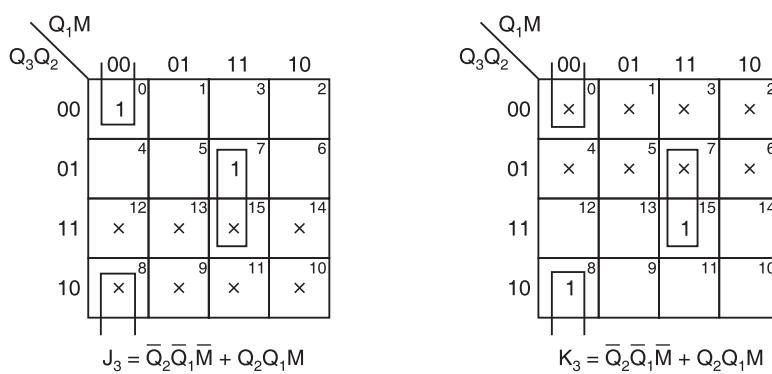
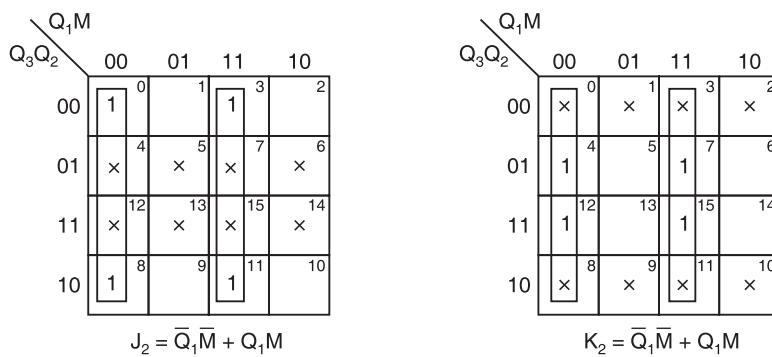
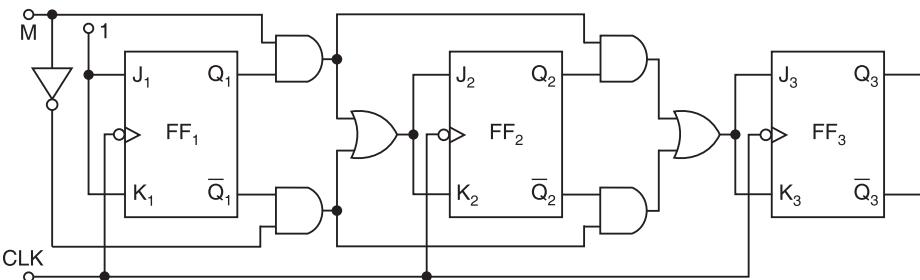


Figure 12.14 K-maps for excitations of synchronous 3-bit up-down counter (Contd.)...



**Figure 12.14** K-maps for excitations of synchronous 3-bit up-down counter.



**Figure 12.15** Logic diagram of the synchronous 3-bit up-down counter using J-K FFs.

**Second method:** A 3-bit up-down counter can also be realized by designing the up-counter and the down-counter separately and then combining them using a mode signal and additional gates.

### 12.5.3 Design of Synchronous 3-bit Up-counter

*Step 1. Determine the number of flip-flops required:* A 3-bit up-counter requires 3 flip-flops. The counting sequence is 000, 001, 010, 011, 100, 101, 110, 111, 000 ...

*Step 2. Draw the state diagram:* The state diagram of the 3-bit up-counter is drawn as shown in Figure 12.16a.

*Step 3. Select the type of flip-flops and draw the excitation table:* JK flip-flops are selected and the excitation table of a 3-bit up-counter using J-K flip-flops is drawn as shown in Figure 12.16b.

*Step 4. Obtain the minimal expressions:* From the excitation table it is seen that,  $J_1 = K_1 = 1$ , because all the entries for  $J_1$  and  $K_1$  are either a 1 or an X. The K-maps for excitations based on the excitation table and the minimal expressions for excitations  $J_3$ ,  $K_3$ ,  $J_2$ , and  $K_2$  in terms of the present outputs  $Q_3$ ,  $Q_2$ , and  $Q_1$  obtained by minimizing the K-maps are shown in Figure 12.17.

Also observing the up counting sequence, we can conclude that  $Q_1$  changes state for every clock pulse. So  $FF_1$  has to be in toggle mode. Therefore  $J_1 = K_1 = 1$ .  $Q_2$  changes state whenever  $Q_1$  is 1, i.e.  $FF_2$  toggles whenever  $Q_1$  is 1. Therefore  $J_2 = K_2 = Q_1$ .  $Q_3$  changes state whenever  $Q_2$  is 1 and  $Q_1 = 1$ ; that means,  $FF_3$  toggles whenever  $Q_1 Q_2 = 1$ . Therefore  $J_3 = K_3 = Q_1 Q_2$ .

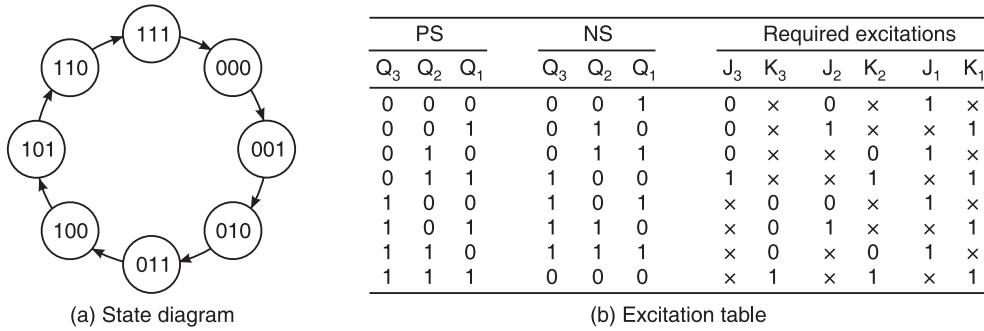


Figure 12.16 A 3-bit up-counter.

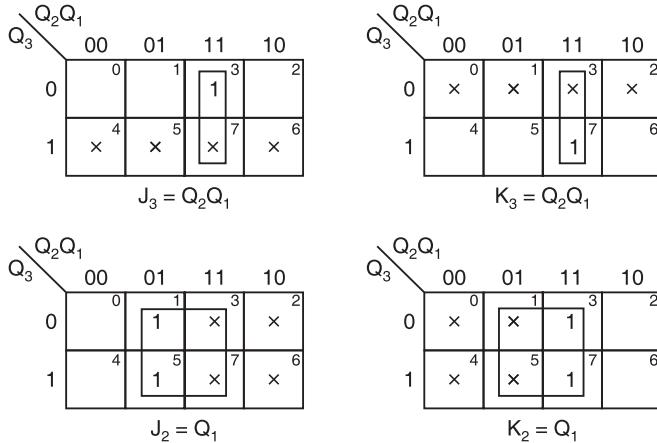


Figure 12.17 Karnaugh maps for a 3-bit up-counter.

#### 12.5.4 Design of Synchronous 3-bit Down-counter

*Step 1. Determine the number of flip-flops required:* A 3-bit down-counter requires 3 flip-flops. The counting sequence is 000, 111, 110, 101, 100, 011, 010, 001, 000, ...

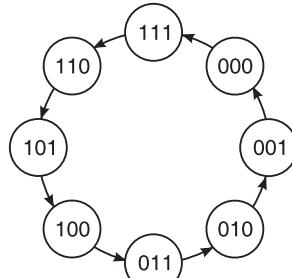
*Step 2. Draw the state diagram:* The state diagram of the 3-bit down-counter is drawn as shown in Figure 12.18a.

*Step 3. Select the type of flip-flops and draw the excitation table:* JK flip-flops are selected and the excitation table of a 3-bit down-counter using J-K FFs is drawn as shown in Figure 12.18b.

*Step 4. Obtain the minimal expressions:* From the excitation table it is seen that,  $J_1 = K_1 = 1$ , because all the entries for  $J_1$  and  $K_1$  are either a 1 or an X. The K-maps for excitations based on the excitation table and the minimal expressions for excitations  $J_3$ ,  $K_3$ ,  $J_2$ , and  $K_2$  in terms of the present outputs  $Q_3$ ,  $Q_2$ , and  $Q_1$  obtained by minimizing the K-maps are shown in Figure 12.19.

Also observing the down counting sequence, we can conclude that  $Q_1$  changes state for every clock pulse. So  $FF_1$  has to be in toggle mode. Therefore  $J_1 = K_1 = 1$ .  $Q_2$  changes state whenever  $\bar{Q}_1$  is 1, i.e.  $\bar{Q}_1 = 1$ , i.e.  $FF_2$  toggles whenever  $\bar{Q}_1$  is 1. Therefore  $J_2 = K_2 = \bar{Q}_1$ .  $Q_3$  changes state

whenever  $Q_2 = 0$  and  $Q_1 = 0$ , i.e.  $\bar{Q}_1 = 1$  and  $\bar{Q}_2 = 1$ ; that means,  $FF_3$  toggles whenever  $\bar{Q}_1\bar{Q}_2 = 1$ . Therefore  $J_3 = K_3 = \bar{Q}_1\bar{Q}_2$ .

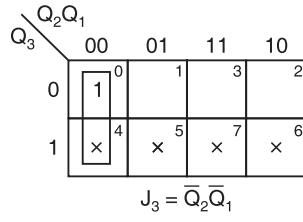


(a) State diagram

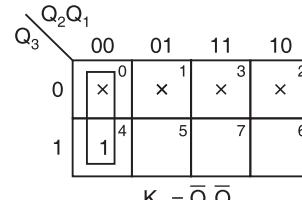
PS			NS			Required excitations					
$Q_3$	$Q_2$	$Q_1$	$Q_3$	$Q_2$	$Q_1$	$J_3$	$K_3$	$J_2$	$K_2$	$J_1$	$K_1$
0	0	0	1	1	1	1	x	1	x	1	x
1	1	1	1	1	0	x	0	x	0	x	1
1	1	0	1	0	1	x	0	x	1	1	x
1	0	1	1	0	0	x	0	0	x	x	1
1	0	0	0	1	1	x	1	1	x	1	x
0	1	1	0	1	0	0	x	x	0	x	1
0	1	0	0	0	1	0	x	x	1	1	x
0	0	1	0	0	0	0	x	0	x	x	1

(b) Excitation table

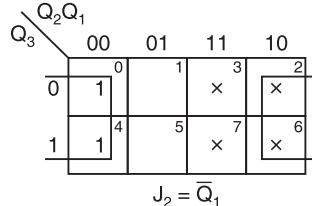
Figure 12.18 A 3-bit down-counter.



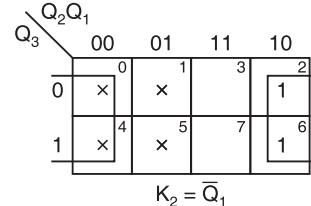
$$J_3 = \bar{Q}_2\bar{Q}_1$$



$$K_3 = \bar{Q}_2\bar{Q}_1$$



$$J_2 = \bar{Q}_1$$



$$K_2 = \bar{Q}_1$$

Figure 12.19 K-maps for a three-bit down counter using J-K FFs.

From the design equations of the up-counter and the down-counter we see that, in both cases  $J_1 = 1$  and  $K_1 = 1$ . Therefore, for the up/down counter too,  $J_1 = 1$  and  $K_1 = 1$ . For the up-counter  $J_2 = K_2 = Q_1$  and for the down-counter,  $J_2 = K_2 = \bar{Q}_1$ . So, for the up/down counter,  $J_2 = K_2 = (Q_1 \cdot \text{Up}) + (\bar{Q}_1 \cdot \text{Down})$ .

Similarly, for the up-counter,  $J_3 = K_3 = Q_2 Q_1$ , and for the down-counter,  $J_3 = K_3 = \bar{Q}_2 \bar{Q}_1$ . So, for an up/down counter

$$J_3 = K_3 = (Q_1 \cdot Q_2 \cdot \text{Up}) + (\bar{Q}_1 \cdot \bar{Q}_2 \cdot \text{Down})$$

By using a control signal  $M$ , and taking  $M = 1$  for the up-mode and  $M = 0$  for the down-mode and combining the above equations, the design equations of an up/down counter are

$$J_1 = K_1 = 1$$

$$J_2 = K_2 = (Q_1 \cdot \text{Up}) + (\bar{Q}_1 \cdot \text{Down}) = Q_1 M + \bar{Q}_1 \bar{M}$$

$$J_3 = K_3 = (Q_1 \cdot Q_2 \cdot \text{Up}) + (\bar{Q}_1 \cdot \bar{Q}_2 \cdot \text{Down}) = Q_1 Q_2 M + \bar{Q}_1 \bar{Q}_2 \bar{M}$$

We see that these equations are the same as the equations obtained by the direct design of the up/down counter. So, the circuit will be the same as that shown in Figure 12.15.

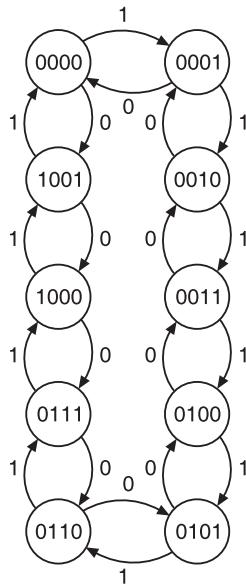
### 12.5.5 Design of a Synchronous Modulo-10 up/down Counter Using T FFs

*Step 1. The number of flip-flops:* A modulo-10 counter has 10 states and so it requires 4 FFs. ( $10 \leq 2^4$ ).

4-FFs can have 16 states. So out of 16, six states (1010 through 1111) are invalid. The entries for excitations corresponding to invalid states are don't cares. For selecting up and down modes a control or mode signal is required. Let us say it counts up when the mode signal  $M = 1$  and counts down when  $M = 0$ . The clock signal is applied to all the FFs simultaneously.

*Step 2. The state diagram:* The state diagram of the mod-10 up/down counter is drawn as shown in Figure 12.20a.

*Step 3. The type of flip-flops and the excitation table:* T flip-flops are selected and the excitation table of the modulo-10 up/down counter using T FFs is drawn as shown in Figure 12.16b.



(a) State diagram

PS				Mode	NS				Required excitations			
$Q_4$	$Q_3$	$Q_2$	$Q_1$	$M$	$Q_4$	$Q_3$	$Q_2$	$Q_1$	$T_4$	$T_3$	$T_2$	$T_1$
0	0	0	0	0	1	0	0	1	1	0	0	1
0	0	0	0	1	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	0	1	0	0	0	1
0	0	1	0	0	0	0	0	1	0	0	1	1
0	0	1	0	1	0	0	1	1	0	0	0	1
0	0	1	1	0	0	0	1	0	0	0	0	1
0	1	0	0	0	0	0	1	1	0	1	1	1
0	1	0	0	1	0	1	0	1	0	0	0	1
0	1	0	1	1	0	1	1	0	0	0	0	1
0	1	1	0	0	0	1	0	1	0	0	0	1
0	1	1	0	1	0	1	1	1	0	0	0	1
0	1	1	1	0	0	1	1	0	0	0	0	1
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	0	1	0	0	0	1
1	0	0	1	0	0	1	0	0	0	0	0	1
1	0	0	1	1	1	0	0	0	1	0	0	1

(b) Excitation table

Figure 12.20 Synchronous mod-10 up-down counter.

*Step 4. The minimal expressions:* From the excitation table we observe that  $T_1 = 1$  because all the entries for  $T_1$  are 1. The K-maps for  $T_4$  and  $T_3$  based on the excitation table and the minimal expressions obtained from them are shown in Figure 12.21. Also drawing and minimizing the K-maps for  $T_2$ , we get

$$T_2 = Q_4 \bar{Q}_1 \bar{M} + \bar{Q}_4 Q_1 M + Q_2 \bar{Q}_1 \bar{M} + Q_3 \bar{Q}_1 \bar{M}$$

*Step 5. The logic diagram:* The logic diagram based on those minimal expressions is drawn as shown in Figure 12.22.

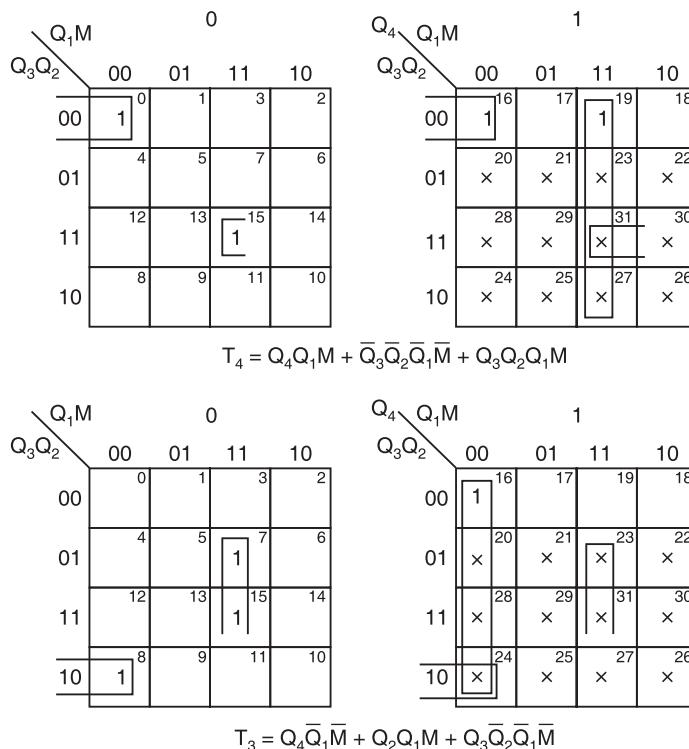


Figure 12.21 K-maps for excitations  $T_4$  and  $T_3$  of a mod-10 up-down counter.

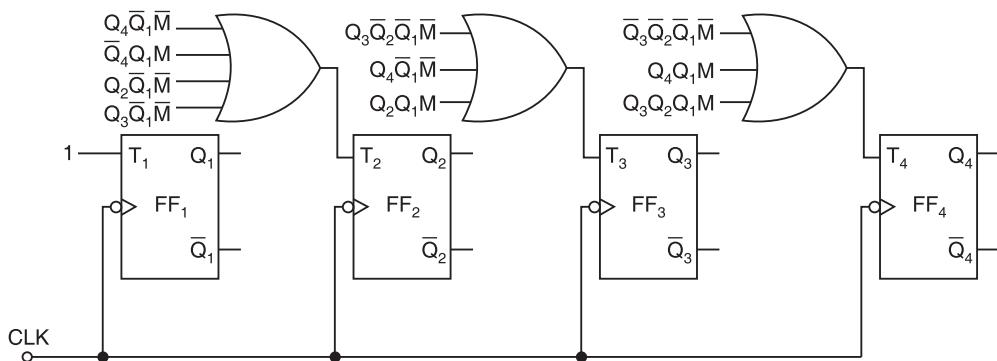


Figure 12.22 Logic diagram of synchronous mod-10 up-down counter.

In the K-maps, the remaining minterms are don't cares ( $\Sigma d(20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31)$ ).

From the excitation table we can see that  $T_1 = 1$  and the expressions for  $T_4$ ,  $T_3$  and  $T_2$  are:

$$T_4 = \Sigma m(0, 15, 16, 19) + d(20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31)$$

$$T_3 = \Sigma m(7, 8, 15, 16) + d(20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31)$$

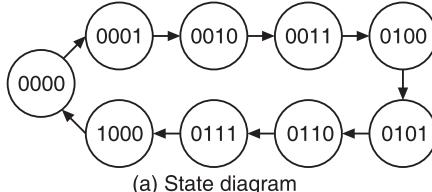
$$T_2 = \Sigma m(3, 4, 7, 8, 11, 12, 15, 16) + d(20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31)$$

### 12.5.6 Design a Modulo-9 Synchronous Counter Using T FFs

*Step 1. The number of flip-flops:* We know that the counting sequence for a modulo-9 counter is 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, .... It has 9 states. So it requires 4 FFs ( $9 \leq 2^4$ ). 4 flip-flops can have 16 states. 7 states 1001, 1010, 1011, 1100, 1101, 1110, and 1111 are invalid. The entries for excitations corresponding to invalid states are don't cares.

*Step 2. The state diagram:* The state diagram for the mod-9 counter is drawn as shown in Figure 12.23a.

*Step 3. The type of flip-flops and the excitation table:* T flip-flops are selected and the excitation table of a mod-9 counter using T FFs is drawn as shown in Figure 12.23b.



PS	NS				Required Excitations								
	Q <sub>4</sub>	Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>4</sub>	Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	T <sub>4</sub>	T <sub>3</sub>	T <sub>2</sub>	T <sub>1</sub>	
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	1	0	0	0	0	0	0	1	1	1
0	0	0	1	0	0	0	1	1	0	0	0	0	1
0	0	0	1	1	0	0	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	1	0	0	0	0	1
0	1	0	0	0	0	1	0	1	0	0	0	0	1
0	1	0	0	1	0	1	1	0	0	0	1	1	1
0	1	1	0	0	0	1	1	1	0	0	0	0	1
0	1	1	1	1	1	0	0	0	1	1	1	1	1
1	0	0	0	0	0	0	0	0	1	0	0	0	0

(b) Excitation table

Figure 12.23 A synchronous mod-9 counter.

*Step 4. The minimal expressions:* The K-maps for excitations T<sub>4</sub>, T<sub>3</sub>, T<sub>2</sub>, and T<sub>1</sub> in terms of the outputs of the FFs Q<sub>4</sub>, Q<sub>3</sub>, Q<sub>2</sub>, and Q<sub>1</sub>, their minimization and the minimal expressions for excitations obtained from them are shown in Figure 12.24.

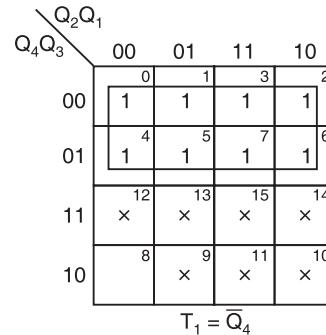
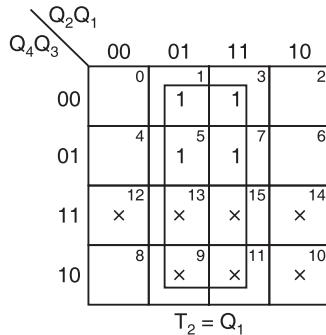
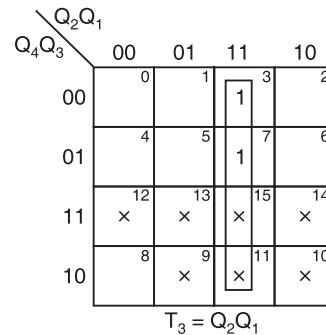
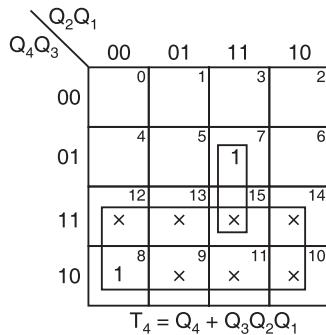


Figure 12.24 K-maps for excitations of synchronous mod-9 counter using T flip-flops.

**Step 5. The logic diagram:** The logic diagram based on those minimal expressions is drawn as shown in Figure 12.25.

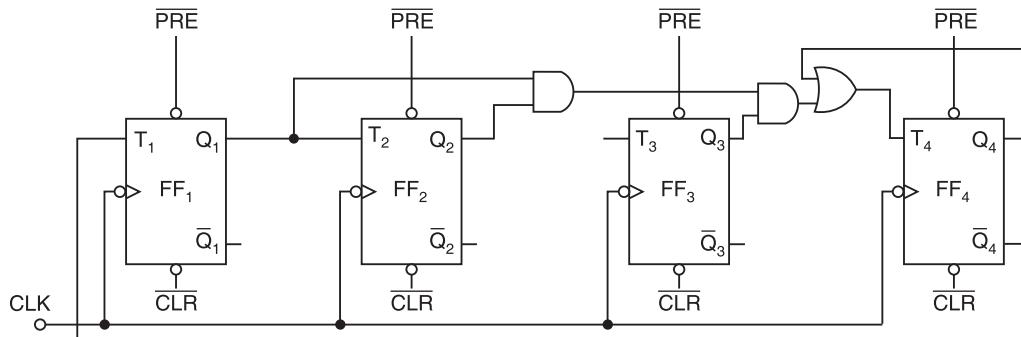


Figure 12.25 Logic diagram of the synchronous mod-9 counter using T flip-flops.

### 12.5.7 Design of a Synchronous Modulo-6 Gray Code Counter

**Step 1. The number of flip-flops:** We know that the counting sequence for a modulo-6 Gray code counter is 000, 001, 011, 010, 110 and 111. It requires  $n = 3$  FFs ( $N \leq 2^n$ , i.e.  $6 \leq 2^3$ ). 3 FFs can have 8 states. So the remaining two states 101 and 100 are invalid. The entries for excitations corresponding to invalid states are don't cares.

**Step 2. The state diagram:** The state diagram of the mod-6 Gray code counter is drawn as shown in Figure 12.26a.

**Step 3. Type of flip-flops and the excitation table:** T flip-flops are selected and the excitation table of the mod-6 Gray code counter using T flip-flops is written as shown in Figure 12.26b.

PS			NS			Required Excitations		
$Q_3$	$Q_2$	$Q_1$	$Q_3$	$Q_2$	$Q_1$	$T_3$	$T_2$	$T_1$
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	0	1
0	1	0	1	1	0	1	0	0
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

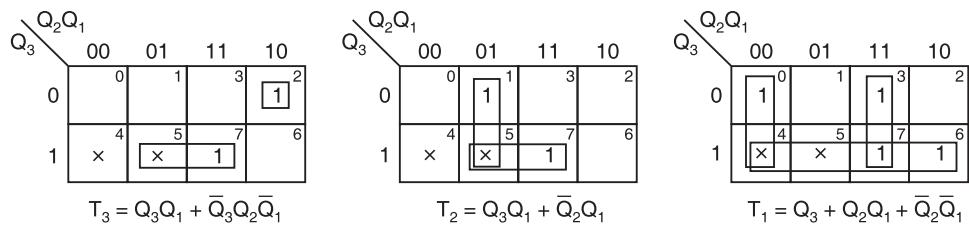
(a) State diagram

(b) Excitation table

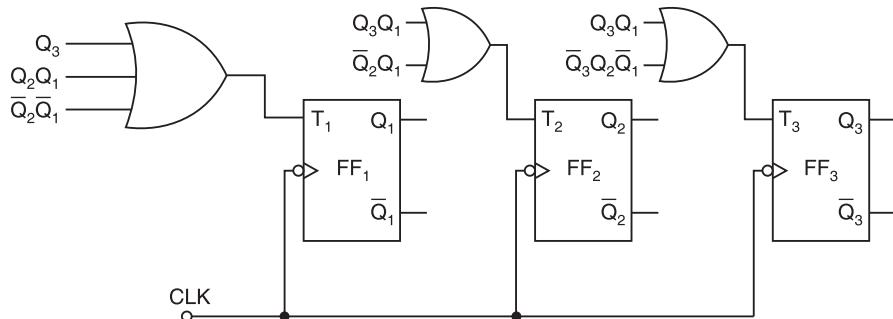
Figure 12.26 Synchronous mod-6 Gray code counter.

**Step 4. The minimal expressions:** The K-maps for excitations of FFs  $T_3$ ,  $T_2$  and  $T_1$  in terms of outputs of FFs  $Q_3$ ,  $Q_2$  and  $Q_1$ , their minimization and the minimal expressions for excitations obtained from them are shown in Figure 12.27.

**Step 5. The logic diagram:** The logic diagram based on those minimal expressions is drawn as shown in Figure 12.28.



**Figure 12.27** K-maps for excitations of a synchronous mod-6 Gray code counter.



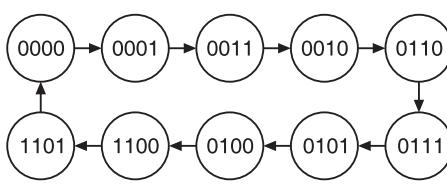
**Figure 12.28** Logic diagram of a synchronous mod-6 Gray code counter.

### 12.5.8 Design of a Synchronous Modulo-10 Gray Code Counter

**Step 1. The number of flip-flops:** We know that the counting sequence for a modulo-10 Gray code counter is 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 0000.... It has 10 states and so it requires  $n = 4$  FFs ( $N \leq 2^n$ , i.e.  $10 \leq 2^4$ ). Four FFs can have 16 states. So the remaining six states 1111, 1110, 1010, 1011, 1001 and 1000 are invalid. The entries for excitations corresponding to these invalid states are don't cares.

*Step 2. The state diagram:* The state diagram of the mod-10 Gray code counter is drawn as shown in Figure 12.29a.

*Step 3. The type of flip-flops and the excitation table:* T flip-flops are selected and the excitation table of the mod-10 Gray code counter using T FFs is written as shown in Figure 12.29b.



(a) State diagram

PS				NS				Required Excitations			
Q <sub>4</sub>	Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>4</sub>	Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	T <sub>4</sub>	T <sub>3</sub>	T <sub>2</sub>	T <sub>1</sub>
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	1	0	0	1	0
0	0	1	1	0	0	1	0	0	0	0	1
0	0	1	0	0	1	1	0	0	1	0	0
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	0	1	0	1	0	0	0	1
0	1	0	1	0	1	0	0	0	0	0	1
0	1	0	0	1	1	0	0	1	0	0	0
1	1	0	0	1	1	0	1	0	0	0	1
1	1	0	1	0	0	0	0	1	1	0	1

**Figure 12.29** Synchronous mod-10 Gray code counter.

**Step 4. The minimal expressions:** The K-maps for excitations of FFs  $T_4$ ,  $T_3$ ,  $T_2$  and  $T_1$  in terms of outputs of FFs  $Q_4$ ,  $Q_3$ ,  $Q_2$  and  $Q_1$ , their minimization, and the minimal expressions for excitations obtained from them are shown in Figure 12.30.

**Step 5. The logic diagram:** The logic diagram based on those minimal expressions is drawn as shown in Figure 12.31.

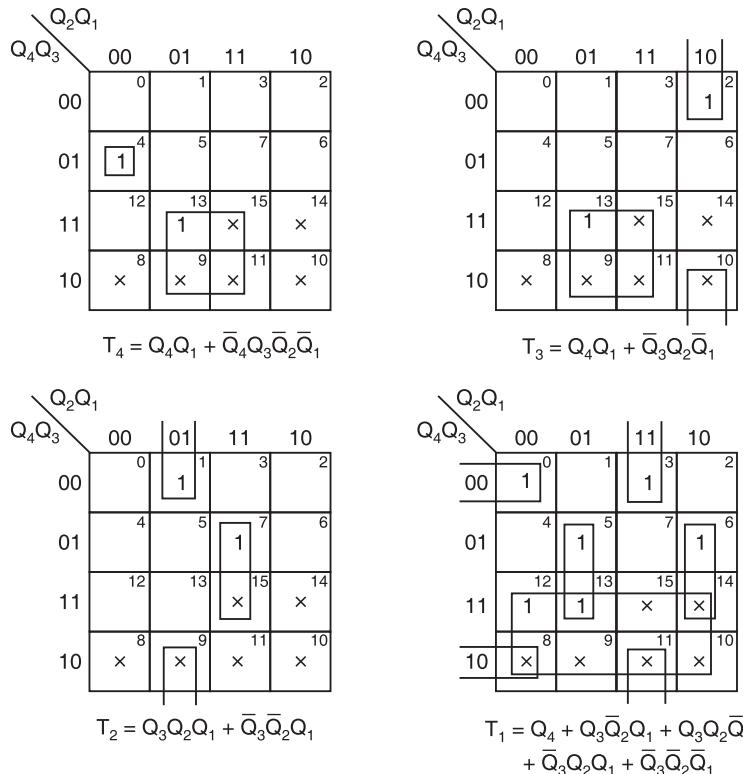


Figure 12.30 K-maps for excitations of a synchronous mod-10 Gray code counter.

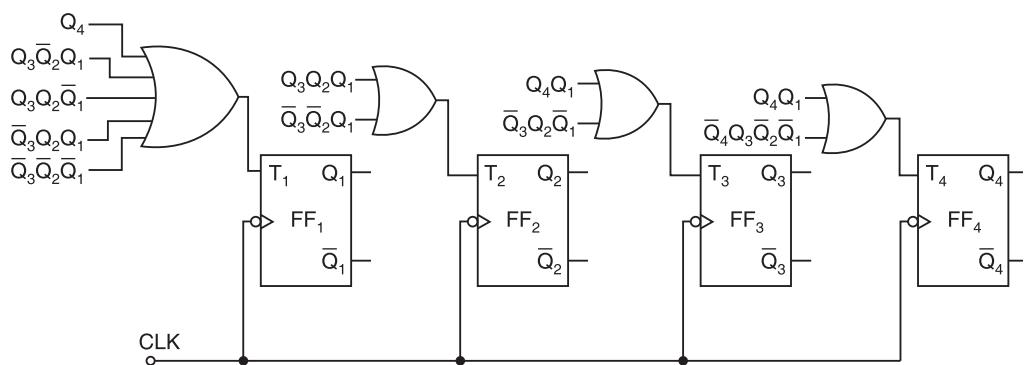


Figure 12.31 Logic diagram of synchronous mod-10 Gray code counter using T FFs.

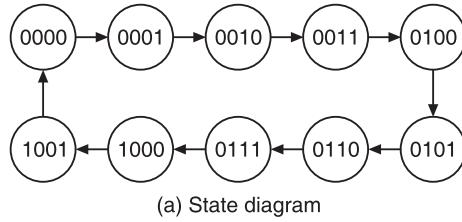
### 12.5.9 Design of a Synchronous BCD Counter Using J-K FFs

*Step 1. The number of flip-flops:* A BCD counter is nothing but a mod-10 counter. It is a decade counter. It has 10 states (0000 through 1001). It requires  $n = 4$  FFs ( $N \leq 2^n$ , i.e.  $10 \leq 2^4$ ). Four FFs can have 16 states. After the tenth clock pulse, the counter resets. So, states 1010 through, 1111 are invalid. The entries for excitations corresponding to invalid states are don't cares.

*Step 2. The state diagram:* The state diagram of the BCD counter is drawn as shown in Figure 12.32a.

*Step 3. The type of flip-flops and the excitation table:* JK flip-flops are selected and the excitation table of the BCD counter using J-K FFs is drawn as shown in Figure 12.32b.

*Step 4. The minimal expressions:* The K-maps for the excitations of FFs  $J_4$ ,  $K_4$ ,  $J_3$ ,  $K_3$ ,  $J_2$ ,  $K_2$ ,  $J_1$  and  $K_1$  in terms of the outputs of the FFs  $Q_4$ ,  $Q_3$ ,  $Q_2$  and  $Q_1$ , based on the excitation table, their minimization and the minimal expressions obtained from them are shown in Figure 12.33.



(a) State diagram

PS				NS				Required excitations							
$Q_4$	$Q_3$	$Q_2$	$Q_1$	$Q_4$	$Q_3$	$Q_2$	$Q_1$	$J_4$	$K_4$	$J_3$	$K_3$	$J_2$	$K_2$	$J_1$	$K_1$
0	0	0	0	0	0	0	1	0	x	0	x	0	x	1	x
0	0	0	1	0	0	1	0	0	x	0	x	1	x	x	1
0	0	1	0	0	0	1	1	0	x	0	x	x	0	1	x
0	0	1	1	0	1	0	0	0	x	1	x	x	1	x	1
0	1	0	0	0	1	0	1	0	x	x	0	0	x	1	x
0	1	0	1	0	1	1	0	0	x	x	0	1	x	x	1
0	1	1	0	0	1	1	1	0	x	x	0	x	0	1	x
0	1	1	1	1	0	0	0	1	x	x	1	x	1	x	1
1	0	0	0	1	0	0	1	x	0	0	x	0	x	1	x
1	0	0	1	0	0	0	0	x	1	0	x	0	x	x	1

(b) Excitation table

Figure 12.32 Synchronous BCD (mod-10) counter.

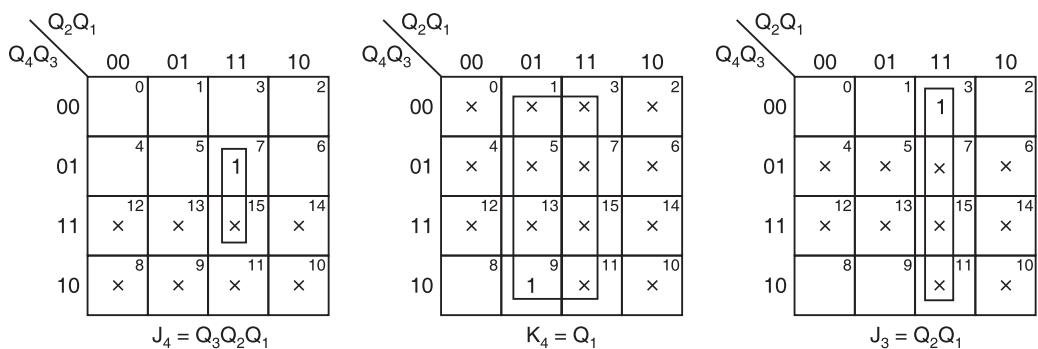


Figure 12.33 K-maps for excitations of synchronous BCD counter using J-K flip-flops (Contd.)...

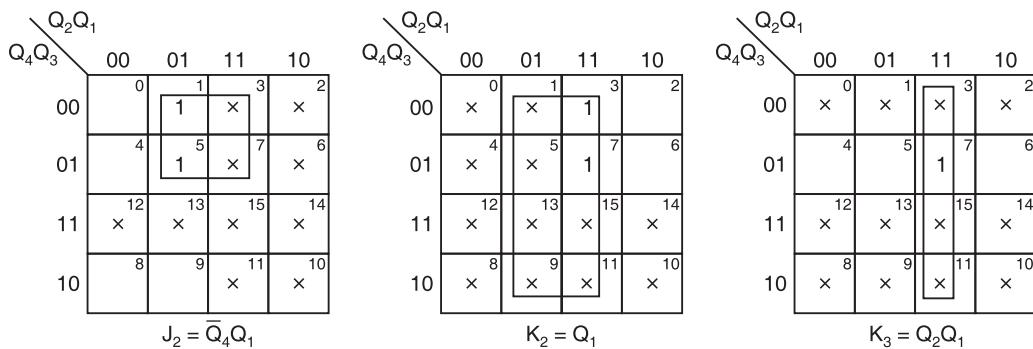


Figure 12.33 K-maps for excitations of synchronous BCD counter using J-K flip-flops.

*Step 5. The logic diagram:* The logic diagram based on those minimal expressions is drawn as shown in Figure 12.30.

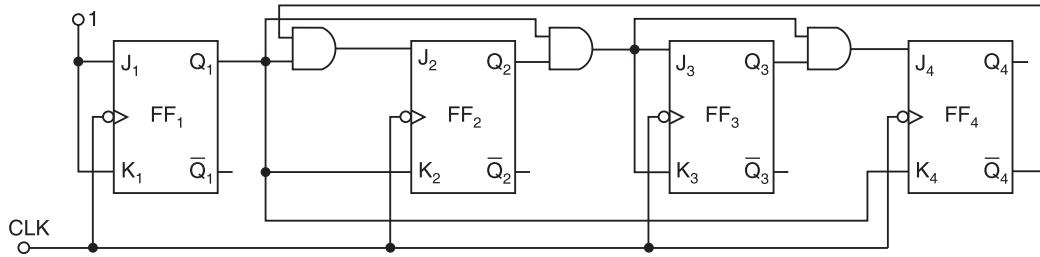
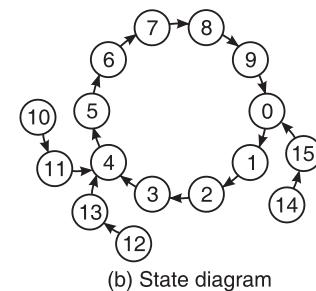


Figure 12.34 Logic diagram of synchronous BCD counter using J-K flip-flops.

The state diagram and the table to check for lock-out are shown in Figure 12.35. The table to check for lock-out shows that, if the counter finds itself in an invalid state initially, it moves to a valid state after one or two clock pulses and then counts in the normal way. Therefore, the counter is self-starting.

PS				Present inputs						NS					
$Q_4$	$Q_3$	$Q_2$	$Q_1$	$J_4$	$K_4$	$J_3$	$K_3$	$J_2$	$K_2$	$J_1$	$K_1$	$Q_4$	$Q_3$	$Q_2$	$Q_1$
1	0	1	0	0	0	0	0	0	0	1	1	1	0	1	1
1	0	1	1	0	1	1	1	0	1	1	1	0	1	0	0
1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	1
1	1	0	1	0	1	0	0	0	1	1	1	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	0	1	1	1	0	0	0	0

(a) Table to check for lock-out



(b) State diagram

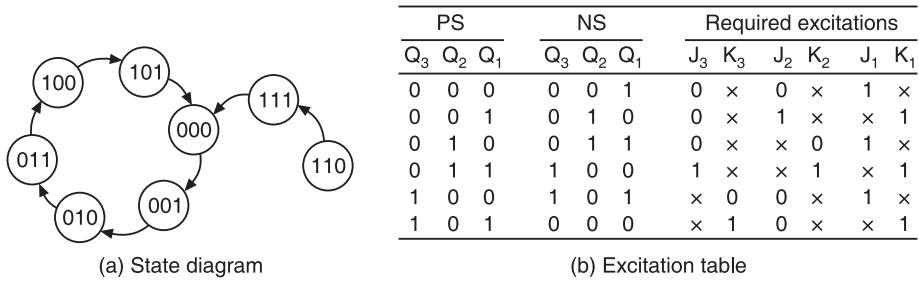
Figure 12.35 Checking for lock out of Synchronous BCD counter using J-K flip-flops.

### 12.5.10 Design of a Synchronous Mod-6 Counter Using J-K FFs

*Step 1. The number of flip-flops:* We know that the counting sequence for a mod-6 counter is 000, 001, 010, 011, 100, 101, 000, ... . It has six states. So it requires  $n = 3$  FFs ( $N \leq 2^n$ , i.e.  $6 \leq 2^3$ ). Three FFs can have eight states. So the remaining two states 110 and 111 are invalid. The entries for excitations corresponding to invalid states are don't cares.

*Step 2. The state diagram:* The state diagram for the mod-6 counter is drawn as shown in Figure 12.36a.

*Step 3. The type of flip-flops and the excitation table:* JK flip-flops are selected and the excitation table of a mod-6 counter using J-K FFs is drawn as shown in Figure 12.36b. (States 110 and 111 can be removed from the state diagram if it is not required to determine whether the counter is self starting or not).

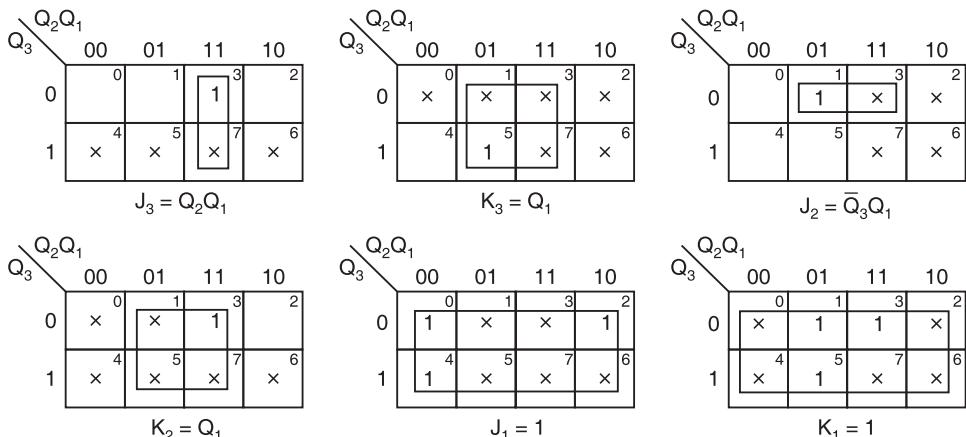


(a) State diagram

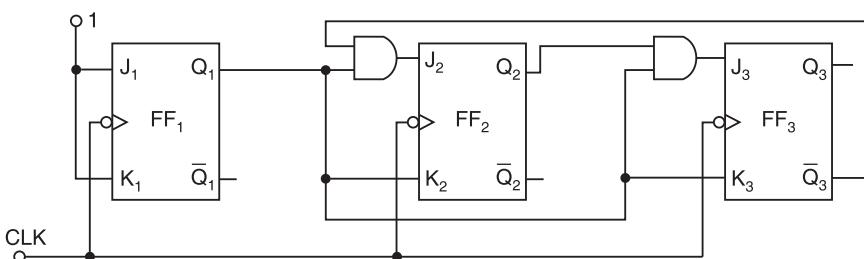
(b) Excitation table

**Figure 12.36** Synchronous mod-6 counter using J-K flip-flops.

*Step 4. The minimal expressions:* The K-maps for excitations of FFs  $J_3$ ,  $K_3$ ,  $J_2$ ,  $K_2$ ,  $J_1$  and  $K_1$  in terms of the outputs of FFs  $Q_3$ ,  $Q_2$  and  $Q_1$ , their minimization, and the minimal expressions for excitations obtained from them are shown in Figure 12.37.

**Figure 12.37** K-maps for excitations of synchronous mod-6 counter using J-K flip-flops.

*Step 5. The logic diagram:* The logic diagram based on those minimal expressions is drawn as shown in Figure 12.38.

**Figure 12.38** Logic diagram of synchronous mod-6 counter using J-K flip-flops.

**Check for self starting:** For this counter, 110 and 111 are the invalid states. We can determine the counter's sequence when its initial present state is 110 or 111. Given each present state, we can determine the J and K inputs from the logic diagram. With these inputs applied to the counter, which is in the present state, we can determine each of the next states of the counter. From Table 12.3 we see that, if the present state ( $Q_3Q_2Q_1$ ) is 110, the excitations are  $J_3 = 0$ ,  $K_3 = 0$ ,  $J_2 = 0$ ,  $K_2 = 0$ ,  $J_1 = 1$  and  $K_1 = 1$ . With these excitations, the counter changes from 110 to 111. If the present state is 111, the excitations are  $J_3 = 1$ ,  $K_3 = 1$ ,  $J_2 = 0$ ,  $K_2 = 1$ ,  $J_1 = 1$  and  $K_1 = 1$ . With these excitations the counter changes from 111 to 000. From the table, to check for lock-out, we see that the counter is self-starting because if the counter initially finds itself in state 111, it goes to 000 after one clock pulse and if the counter is initially in state 110, then it goes to 111 after one clock pulse and goes to 000 after two clock pulses.

**Table 12.3** Check for lock-out

PS			Present inputs						NS		
Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	J <sub>3</sub>	K <sub>3</sub>	J <sub>2</sub>	K <sub>2</sub>	J <sub>1</sub>	K <sub>1</sub>	Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>
1	1	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	0	1	1	1	0	0	0

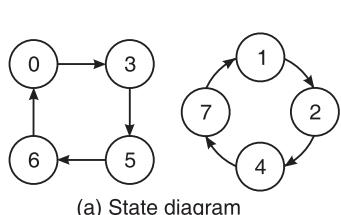
**EXAMPLE 12.4** Design a type T counter that goes through states 0, 3, 5, 6, 0 ... . Is the counter self-starting?

### *Solution*

*Step 1. The number of flip-flops:* This counter has only four stable states, but it requires three FFs, because it counts 101 and 110 ( $6 \leq 2^3$ ). Three FFs can have 8 states, out of which states 000, 011, 101, 110 are valid and states 001, 010, 100, 111 are invalid. The entries for excitations corresponding to invalid states are don't cares.

*Step 2. The state diagram:* The state diagram of the counter is shown in Figure 12.39a.

*Step 3. The type of flip-flops and the excitation table:* T flip-flops are selected and the excitation table of this counter using T flip-flops is drawn as shown in Figure 12.39b.



(a) State diagram

PS			NS			Required excitations		
Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	T <sub>3</sub>	T <sub>2</sub>	T <sub>1</sub>
0	0	0	0	1	1	0	1	1
0	1	1	1	0	1	1	1	0
1	0	1	1	1	0	0	1	1
1	1	0	0	0	0	1	1	1

(b) Excitation table

**Figure 12.39** Example 12.4: T type, 0, 3, 5, 6, 0 ... counter.

**Step 4. The minimal expressions:** The K-maps for  $T_3$ ,  $T_2$ , and  $T_1$  in terms of  $Q_3$ ,  $Q_2$ , and  $Q_1$  based on the excitation table, their minimization, and the minimal expressions obtained from them are shown in Figure 12.40.

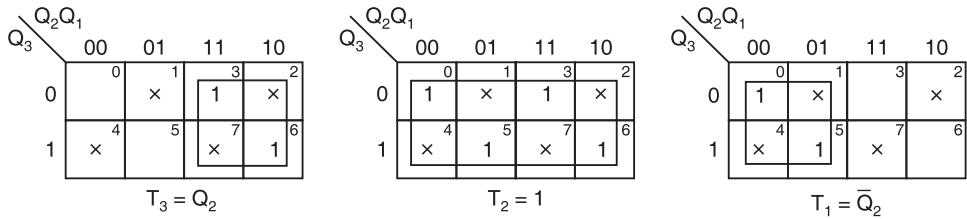


Figure 12.40 Example 12.4: K-maps for excitations of T type, 0, 3, 5, 6, 0... counter.

*Step 5. The logic diagram:* The logic diagram based on those minimal expressions is shown in Figure 12.41a.

The table to check for lock-out is shown in Figure 12.41b. We see that the counter is not self-starting. It suffers from the problem of lock-out. That is, initially if it enters an invalid state, it keeps on moving from one invalid state to another when clock pulses are applied and never returns to a valid state and, therefore, it serves no useful purpose.

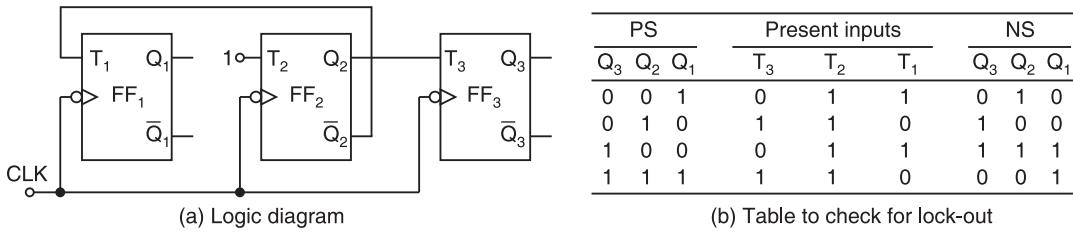


Figure 12.41 Example 12.4: T type counter that goes through states 0, 3, 5, 6, 0, ... .

**Elimination of lock-out:** There are two methods to eliminate lock-out.

*First method.* Obtain a reset pulse as shown in Figure 12.42 such that, whenever the counter goes to an invalid state, it automatically resets to the initial state 000.

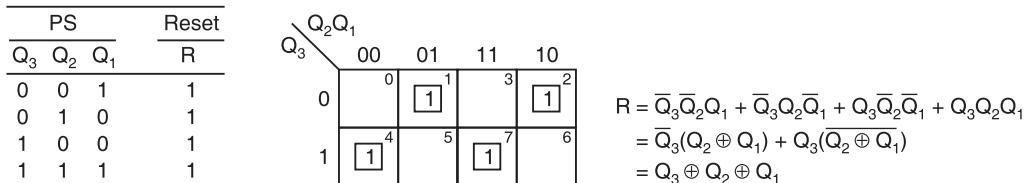


Figure 12.42 Example 12.4: Table for R, K-map, and the minimal expression for R.

Obtain this reset pulse from the counter already designed (taking the excitations corresponding to the invalid states as don't cares) and feed it to the CLR terminals of the FFs. The circuit of Figure 12.41a is, therefore, modified as shown in Figure 12.43. The counter comes out of the invalid state without requiring another clock pulse.

*Second method.* Redesign the counter assuming that the counter goes to the starting state, i.e. 000 whenever it enters any of the invalid states. The excitation requirements are shown in Figure 12.44a. In this method, no don't cares are available for excitations even if the present state is an invalid one. The counter designed by this method cannot come out of the invalid state

instantaneously. It requires one more clock pulse to let the counter come out of the invalid state. The circuit is also more complicated because no don't cares are present. The K-maps for  $T_3$ ,  $T_2$ , and  $T_1$  based on the excitation table and the minimal expressions obtained from them are shown in Figure 12.44b. A logic circuit may be realized using those minimal expressions.

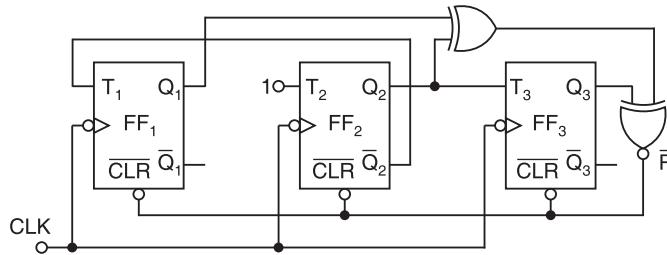
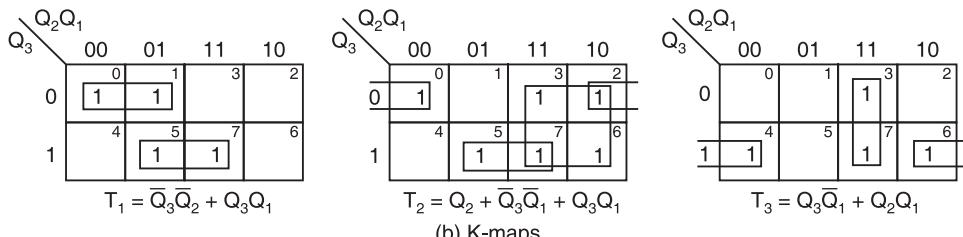


Figure 12.43 Logic diagram of the type T 0, 3, 5, 6, 0... counter modified to eliminate lock-out.

PS			NS			Required excitations		
$Q_3$	$Q_2$	$Q_1$	$Q_3$	$Q_2$	$Q_1$	$T_3$	$T_2$	$T_1$
0	0	0	0	1	1	0	1	1
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	0	1	0
0	1	1	1	0	1	1	1	0
1	0	0	0	0	0	1	0	0
1	0	1	1	1	0	0	1	1
1	1	0	0	0	0	1	1	0
1	1	1	0	0	0	1	1	1

(a) Excitation table



(b) K-maps

Figure 12.44 Example 12.4: Excitation table and K-maps.

**EXAMPLE 12.5** Design a type D counter that goes through states 0, 1, 2, 4, 0, ... . The undesired (unused) states must always go to zero (000) on the next clock pulse.

### Solution

**Step 1. The number of flip-flops:** This counter has only four stable states 0, 1, 2 and 4 (000, 001, 010, 100), but it requires three FFs because it counts 4 (100) as well. Three FFs can have eight states. So, the remaining four states (011, 101, 110, 111) are undesired. These undesired states must go to 000 after the next clock pulse. So, no don't cares.

**Step 2. The state diagram:** The state diagram of the 0, 1, 2, 4, 0, ... counter is drawn as shown in Figure 12.45a.

*Step 3. The type of flip-flops and the excitation table:* D flip-flops are selected and the excitation table of the counter using D FFs is written as shown in Figure 12.45b.

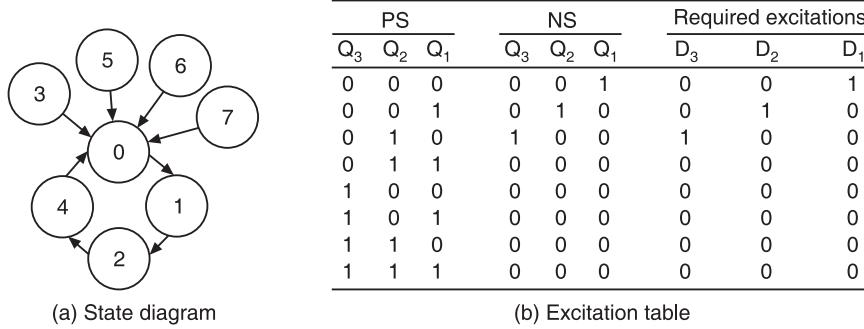


Figure 12.45 Example 12.5: 0, 1, 2, 4, 0, ... counter.

*Step 4. The minimal expressions:* From the excitation table we can see that no minimization is possible. So the expressions for excitations read from the excitation table are:

$$D_3 = \bar{Q}_3 Q_2 \bar{Q}_1; \quad D_2 = \bar{Q}_3 \bar{Q}_2 Q_1; \quad D_1 = \bar{Q}_3 \bar{Q}_2 \bar{Q}_1$$

*Step 5. The logic diagram:* The logic diagram based on these expressions is drawn as shown in Figure 12.46. The counter is self-starting, i.e. it does not suffer from the problem of lock-out as may be seen from the state diagram.

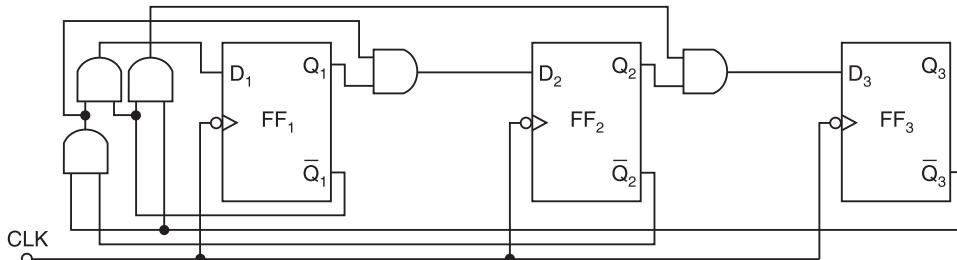


Figure 12.46 Example 12.5: Logic diagram of type D counter that goes through states 0, 1, 2, 4, 0, ... .

**EXAMPLE 12.6** Design a J-K counter that goes through states 3, 4, 6, 7 and 3... . Is the counter self-starting? Modify the circuit such that whenever it goes to an invalid state it comes back to state 3.

### Solution

*Step 1. The number of flip-flops:* The counter has only four states: 3, 4, 6, 7 (011, 100, 110, 111), but the maximum count is 7. So this counter requires three FFs ( $7 \leq 2^3$ ). A counter using three flip-flops can have eight states. So the remaining four states (000, 001, 010, 101) are invalid. The entries for excitations corresponding to these invalid states are don't cares.

*Step 2. The state diagram:* The state diagram of the 3, 4, 6, 7, 3, ... counter is shown in Figure 12.47a.

*Step 3. The type of flip-flops and the excitation table:* JK flip-flops are selected and the excitation table of the counter using JK FFs is drawn as shown in Figure 12.47b.

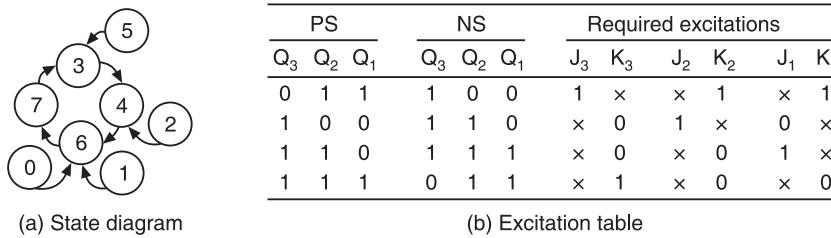


Figure 12.47 Example 12.6: Type J-K 3, 4, 6, 7, 3, ... counter.

*Step 4. The minimal expressions:* From the excitation table,  $J_3 = 1$  and  $J_2 = 1$ , since all the entries for  $J_3$  and  $J_2$  are either a 1 or a X. The K-maps for  $K_3$ ,  $K_2$ ,  $J_1$  and  $K_1$  based on the excitation table, their minimization, and the minimal expressions obtained from them are shown in Figure 12.48.

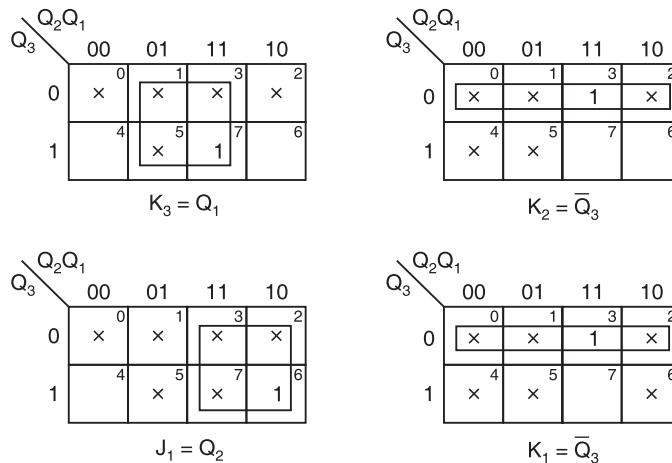


Figure 12.48 Example 12.6: K-maps for excitations of type J-K 3, 4, 6, 7, 3, ... counter.

*Step 5. The logic diagram:* The logic diagram for the counter based on those minimal expressions is drawn as shown in Figure 12.49.

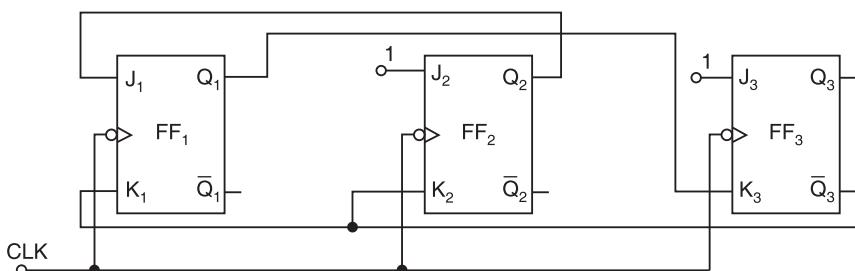


Figure 12.49 Example 12.3: Logic diagram of the J-K counter that goes through states 3, 4, 6, 7, 3, ....

**Test for lock-out:** The NS entries of Table 12.4 to check for lock-out show that there is no problem of lock-out and the counter is self-starting, because any time it goes into an invalid state, it comes out and goes into a useful state after one clock pulse. The state diagram of the counter is shown in Figure 12.47a.

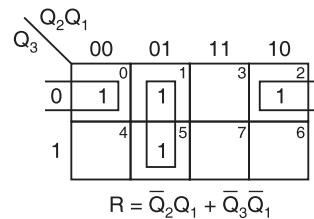
**Table 12.4** Example 12.6: Check for lock-out

PS			Present inputs						NS		
$Q_3$	$Q_2$	$Q_1$	$J_3$	$K_3$	$J_2$	$K_2$	$J_1$	$K_1$	$Q_3$	$Q_2$	$Q_1$
0	0	0	1	0	1	1	0	1	1	1	0
0	0	1	1	1	1	1	1	1	1	1	0
0	1	0	1	0	1	1	0	1	1	0	0
1	0	1	1	1	1	0	1	0	0	1	1

To see that the counter goes to state 3 (011) whenever it enters any of the invalid states, obtain an expression for reset R or set S as shown in Figure 12.50 and modify the circuit accordingly such that the counter goes to 011 after the next clock pulse.

PS		Reset	
$Q_3$	$Q_2$	$Q_1$	R
0	0	0	1
0	0	1	1
0	1	0	1
1	0	1	1

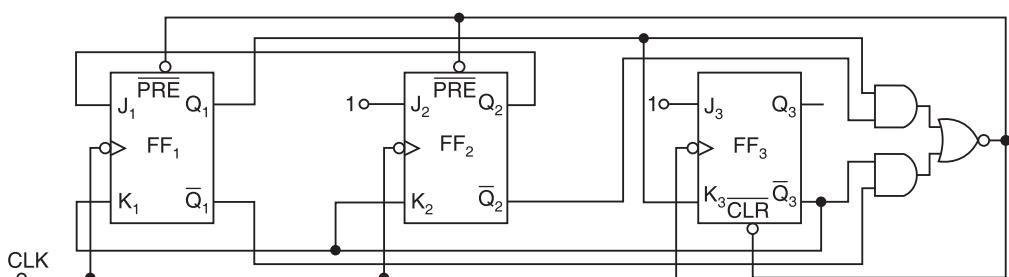
(a) Truth table



(b) K-map

**Figure 12.50** Example 12.6: Type J-K 3, 4, 6, 7, 3, ... counter.

Since we do not want the counter to reset to 000, and instead we want it to go to state 011, we apply the output to the clear terminal of  $FF_3$  and to the preset terminals of  $FF_2$  and  $FF_1$  as shown in the modified circuit of Figure 12.51.



**Figure 12.51** Example 12.6: Logic diagram of the J-K counter modified to go to state 3 from invalid states.

## 12.6 HYBRID COUNTERS

A hybrid counter is a counter in which the output of a synchronous counter drives the clock input of another counter to get a divide-by- $N$  operation. The block diagram of a hybrid counter is shown in Figure 12.52.

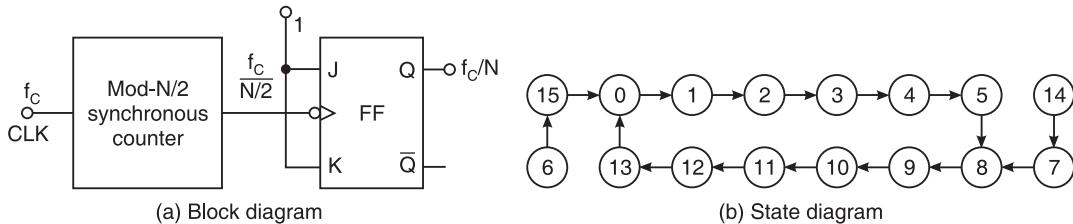


Figure 12.52 Hybrid counter.

Hybrid counters can be used to obtain a symmetrical divide by  $N$  output. For example, when  $N$  is any number divisible by 2, we can obtain a symmetrical divide by  $N$  counter, by making the output of a synchronous mod- $N/2$  counter drive a mod-2 counter. The output of the mod-2 counter has a frequency of  $f_c/N$ . The logic diagram of a mod-12 synchronous hybrid counter obtained by using a mod-6 counter and a mod-2 counter is shown in Figure 12.53. The timing diagram shown in Figure 12.54 indicates that the sequence of the states of the counter is 0, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 0, 1, 2, etc. The counter is self-starting. Whenever it goes to an invalid state, it comes back to the valid state in 1 or 2 clock pulses.

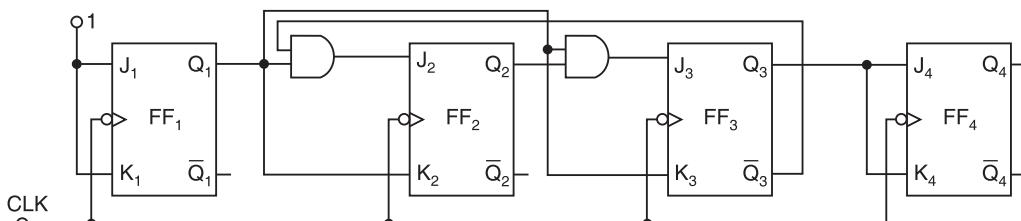


Figure 12.53 Logic diagram of a hybrid mod-12 counter.

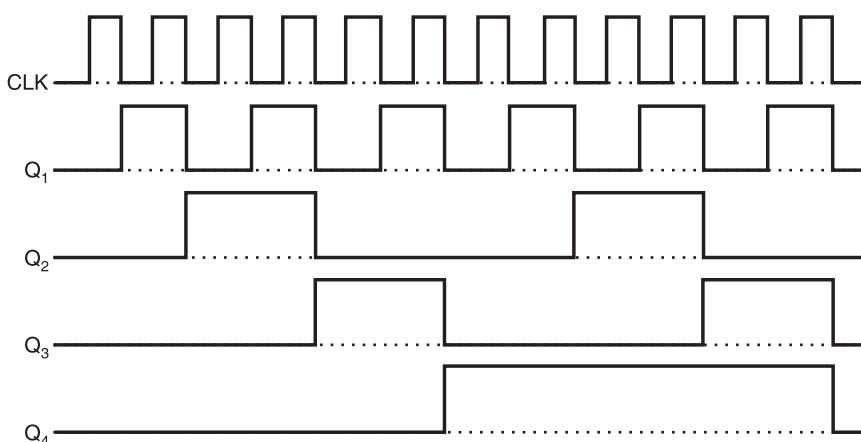


Figure 12.54 Timing diagram of a hybrid counter.

**EXAMPLE 12.7** Compare  $f_{\max}$  of a 4-bit ripple counter with that of a 4-bit synchronous counter using J-K FFs. The  $t_{pd}$  for each FF is 50 ns and the  $t_{pd}$  for each AND gate is 20 ns. What needs to be done to convert these counters to mod-32? Determine  $f_{\max}$  for the mod-32 ripple and parallel counters.

**Solution**

- (a) The total delay that must be allowed between the input clock pulses of a synchronous counter is equal to

$$t_{pd}(\text{FF}) + t_{pd}(\text{AND})$$

Thus,  $T_{\text{clock}} \geq 50 + 20 = 70$  ns. And so, the parallel counter has,  $f_{\max} = 1/(70 \text{ ns}) = 14.3$  MHz.

- (b) A mod-16 ripple counter uses four FFs with  $t_{pd} = 50$  ns. Thus,  $f_{\max}$  for the ripple counter is

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

- (c) A fifth FF must be added, since  $2^5 = 32$ .

- (d) The  $f_{\max}$  of the synchronous counters remains the same regardless of the number of FFs.

Thus, its  $f_{\max}$  is still 14.3 MHz. The  $f_{\max}$  of the 5-bit ripple counter will change to

$$\begin{aligned} f_{\max} &= \frac{1}{(5 \times 50)\text{ns}} \\ &= 4 \text{ MHz} \end{aligned}$$

## 12.7 PROGRAMMABLE COUNTERS

Most synchronous counters, available in IC form, can be pre-loaded with a binary number in parallel form (as with parallel-in shift registers) prior to initiation of counting. This pre-loading capability makes it possible to begin a count sequence from 0 or any other number. Such counters are said to be programmable. In a counter with asynchronous loading, the initial state is loaded using direct SET and direct CLEAR inputs on the FFs. Thus, loading occurs irrespective of the clock. In a counter with synchronous loading, the initial state is loaded on the occurrence of the clock edge, using the J-K inputs of the FFs. Synchronous loading generally requires the use of a rather elaborate logic circuitry connected to the J and K inputs, since these inputs are also needed for control of the normal counting sequence. In either case, the circuit will have a separate LOAD control input which must be made active to achieve loading. Some circuits can be loaded either synchronously or asynchronously and have separate control inputs such as SLOAD and ALOAD, to enable one type of loading or the other to take place.

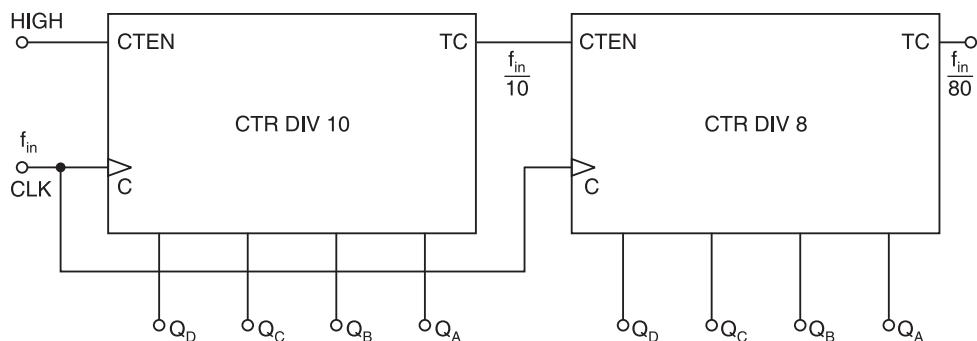
## 12.8 CASCADING OF SYNCHRONOUS COUNTERS

We have talked about cascading of ripple counters earlier. In cascaded counters, the output of the last stage of one counter drives the input of the next counter.

When operating synchronous counters in a cascaded configuration, it is necessary to use the count enable and the terminal count functions to achieve higher modulus operation on some devices. The count enable is labelled simply CTEN or some other similar designation, and terminal count (TC) is analogous to ripple clock or ripple carry out (RCO) on some IC counters.

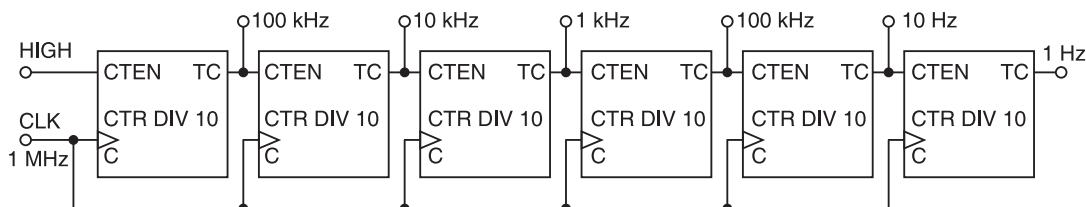
Figure 12.55 shows a mod-10 counter and a mod-8 counter connected in cascade. The terminal count (TC) output of counter 1 is connected to the count enable (CTEN) input of counter 2. The counter 2 is inhibited by the LOW on its CTEN input until counter 1 reaches its last or terminal state when its terminal count output goes HIGH. This HIGH now enables counter 2, so that when the first clock pulse after counter 1 reaches its last or terminal count (CLK10) is applied, counter 2 goes from its initial state to its second state. Upon completion of the entire second cycle of counter 1 (when counter 1 reaches TC the second time), counter 2 is again enabled and advances to its next state. This sequence continues. Since the first one is a decade counter, it must go through 10 complete cycles before counter 2 completes its first cycle. In other words, for every 10 cycles of counter 1, counter 2 goes through one cycle. Since the second counter is a mod-8 counter, it will complete one cycle only after 80 clock pulses. The overall modulus of these two cascaded counters is

$$10 \times 8 = 80.$$



**Figure 12.55** Mod-80 cascaded counter using mod-10 and mod-8 counters.

Figure 12.56 illustrates how to obtain a 1 Hz signal from a 1 MHz signal using decade counters.



**Figure 12.56** Logic diagram to obtain a 1 Hz signal from a 1 MHz signal using decade counters.

## 12.9 SHIFT REGISTER COUNTERS

One of the applications of shift registers is that they can be arranged to form several types of counters. Shift register counters are obtained from serial-in, serial-out shift registers by providing feedback from the output of the last FF to the input of the first FF. These devices are called counters because they exhibit a specified sequence of states. The most widely used shift register counter is the ring counter (also called the basic ring counter or the simple ring counter) as well as the twisted ring counter (also called the Johnson counter or the switch-tail ring counter).

### 12.9.1 Ring Counter

This is the simplest shift register counter. The basic ring counter using D FFs is shown in Figure 12.57. The realization of this counter using J-K FFs is shown in Figure 12.58. Its state diagram and the sequence table are shown in Figure 12.59. Its timing diagram is shown in Figure 12.60. The FFs are arranged as in a normal shift register, i.e. the Q output of each stage is connected to the D input of the next stage, but the Q output of the last FF is connected back to the D input of the first FF such that the array of FFs is arranged in a ring and, therefore, the name *ring counter*.

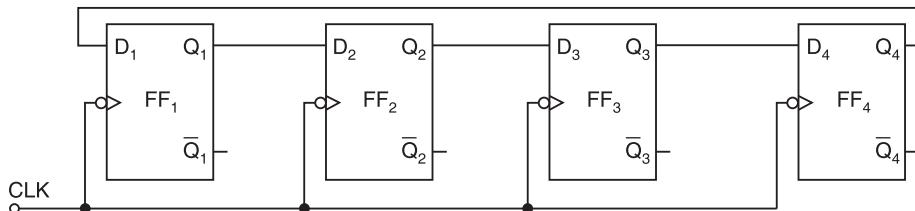


Figure 12.57 Logic diagram of a 4-bit ring counter using D flip-flops.

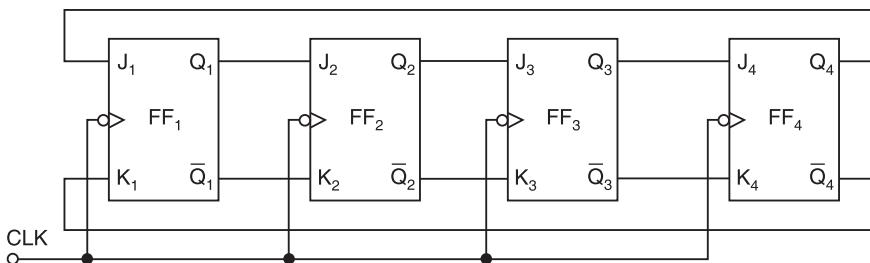


Figure 12.58 Logic diagram of a 4-bit ring counter using J-K flip-flops.

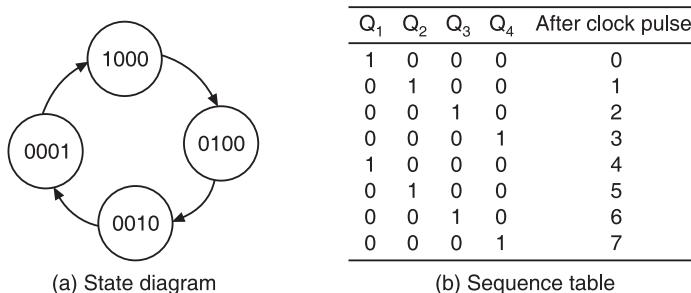
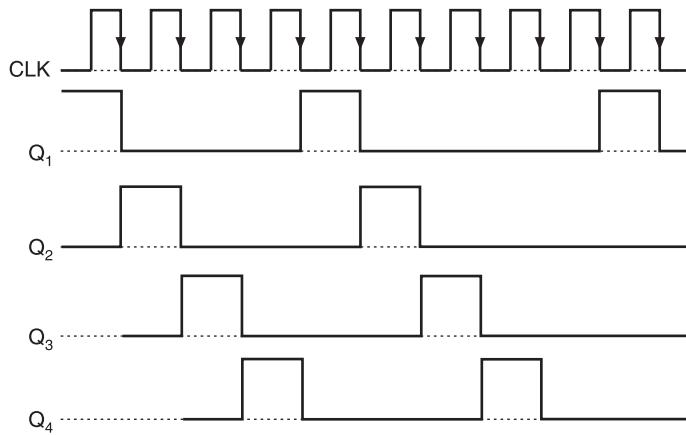


Figure 12.59 State diagram and sequence table of a 4-bit ring counter.

In most instances, only a single 1 is in the register and is made to circulate around the register as long as clock pulses are applied. Initially, the first FF is preset to a 1. So, the initial state is 1000, i.e.  $Q_1 = 1$ ,  $Q_2 = 0$ ,  $Q_3 = 0$  and  $Q_4 = 0$ . After each clock pulse, the contents of the register are shifted to the right by one bit and  $Q_4$  is shifted back to  $Q_1$ . The sequence repeats after four clock pulses. The number of distinct states in the ring counter, i.e. the mod of the ring counter is equal to the number of FFs used in the counter. An  $n$ -bit ring counter can count only  $n$  bits, whereas an  $n$ -bit ripple counter can count  $2^n$  bits. So, the ring counter is uneconomical compared to a ripple counter, but has the advantage of requiring no decoder, since we can read the count by simply noting which

FF is set. Since it is entirely a synchronous operation and requires no gates external to FFs, it has the further advantage of being very fast.

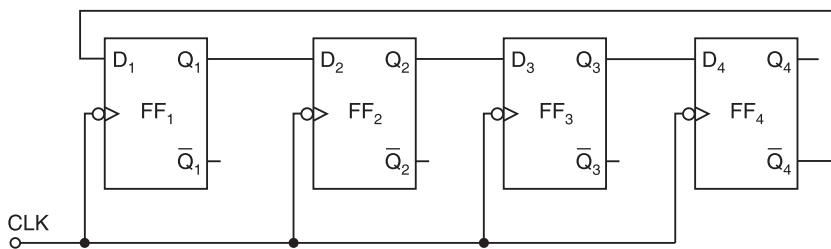


**Figure 12.60** Timing diagram of a 4-bit ring counter.

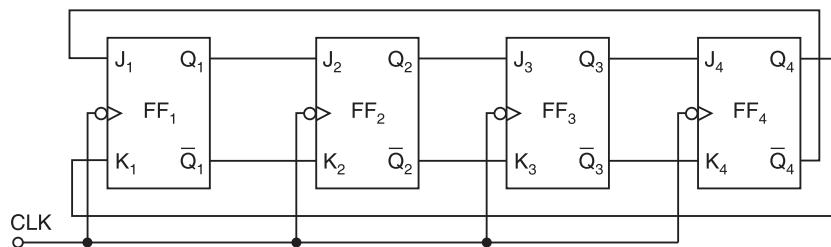
### 12.9.2 Twisted Ring Counter (Johnson Counter)

This counter is obtained from a serial-in, serial-out shift register by providing feedback from the inverted output of the last FF to the D input of the first FF. The Q output of each stage is connected to the D input of the next stage, but the  $\bar{Q}$  output of the last stage is connected to the D input of first stage, therefore, the name *twisted ring counter*. This feedback arrangement produces a unique sequence of states.

The logic diagram of a 4-bit Johnson counter using D FFs is shown in Figure 12.61. The realization of the same using J-K FFs is shown in Figure 12.62. The state diagram and the sequence table are shown in Figure 12.63. The timing diagram of a Johnson counter is shown in Figure 12.64.



**Figure 12.61** Logic diagram of a 4-bit twisted ring counter using D flip-flops.



**Figure 12.62** Logic diagram of a 4-bit twisted ring counter using J-K flip-flops.

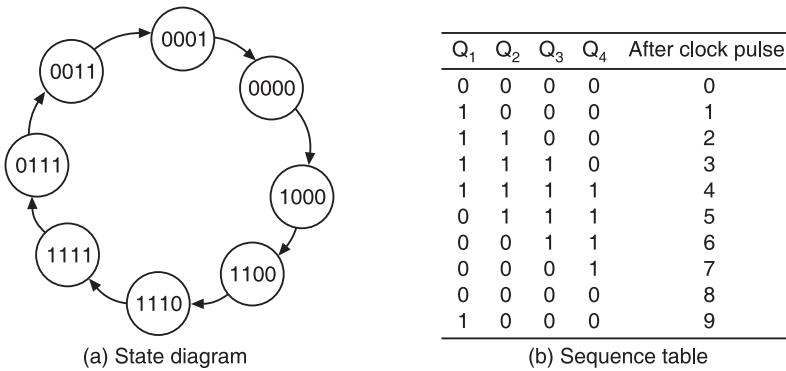


Figure 12.63 State diagram and sequence table of a twisted ring counter.

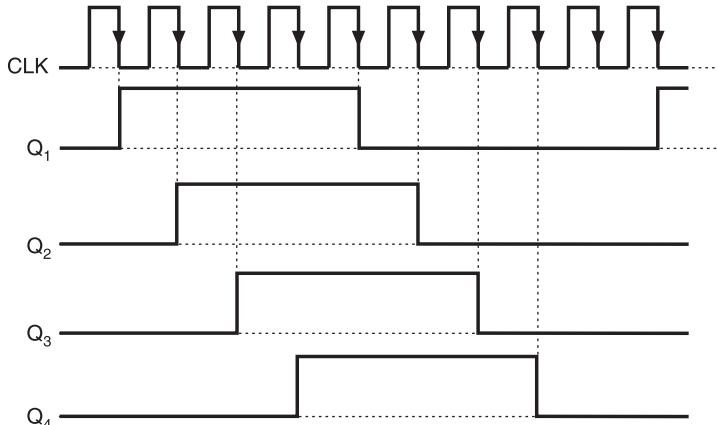


Figure 12.64 Timing diagram of a 4-bit twisted ring counter.

Let initially all the FFs be reset, i.e. the state of the counter be 0000. After each clock pulse, the level of Q<sub>1</sub> is shifted to Q<sub>2</sub>, the level of Q<sub>2</sub> to Q<sub>3</sub>, Q<sub>3</sub> to Q<sub>4</sub> and the level of  $\bar{Q}_4$  to Q<sub>1</sub> and the sequence given in Figure 12.63b is obtained. This sequence is repeated after every eight clock pulses.

An  $n$  FF Johnson counter can have  $2n$  unique states and can count up to  $2n$  pulses. So, it is a mod- $2n$  counter. It is more economical than the normal ring counter, but less economical than the ripple counter. It requires two input gates for decoding regardless of the size of the counter. Thus, it requires more decoding circuitry than that by the normal ring counter, but less than that by the ripple counter. It represents a middle ground between the ring counter and the ripple counter.

Both types of ring counters suffer from the problem of lock-out, i.e. if the counter finds itself in an unused state, it will persist in moving from one unused state to another and will never find its way to a used state. This difficulty can be corrected by adding a gate. With this addition, if the counter finds itself initially in an unused state, then after a number of clock pulses, depending on the state, the counter will find its way to a used state and thereafter, follow the desired sequence. A Johnson counter designed to prevent lock-out is shown in Figure 12.65. A self-starting ring counter (whatever may be the initial state, single 1 will eventually circulate) is shown in Figure 12.66.

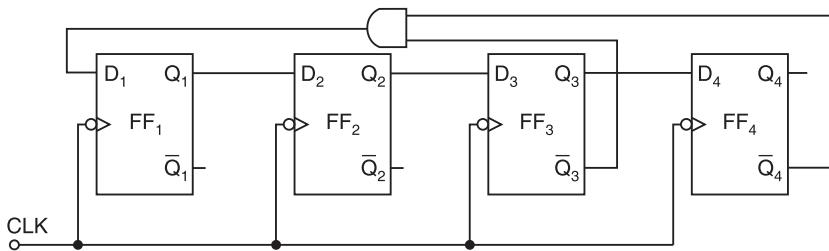


Figure 12.65 Logic diagram of a 4-bit Johnson counter designed to prevent lock-out.

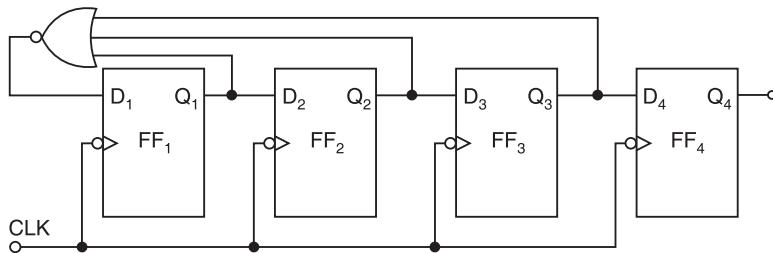


Figure 12.66 Logic diagram of a self-starting ring counter.

## 12.10 PULSE TRAIN GENERATORS (SEQUENCE GENERATORS)

Pulse trains (i.e. a particular repetitive series of pulses) are required on many occasions. A pulse generator or sequence generator is a system which generates, in synchronism with a clock, a prescribed sequence of logic bits. These pulse trains or sequences of bits can be used to open valves, close gates, turn on lights, turn off machines, or perform any of a variety of jobs. Pulse trains can be generated using either direct logic or indirect logic. In direct logic the output is taken directly from an FF, whereas in indirect logic, it is taken from a decoder gate.

### 12.10.1 Direct Logic

The output in direct logic is taken from an FF's Q or  $\bar{Q}$  lead; so, the FF is made to go to the state desired. The design procedure is:

1. Inspect the pulse train given and decide the number of unique states and the minimum number of FFs required and list the entire sequence in terms of 1s and 0s. The list may begin anywhere in the train. The number of unique states is equal to the number of bits in the sequence.
2. Taking that the 1s and 0s of the sequence will form the least significant bits of the state assignment; assign unique states to other FFs. If unique states are not possible with the least number of FFs  $n$ , such that the number of states  $N \leq 2^n$ , increase the number of FFs by one or more to get the unique states. (This is the disadvantage of direct logic.)

The number of flip-flops required to generate a particular sequence can also be determined as follows:

- Find the number of 1s in the sequence.

- Find the number of 0s in the sequence.
- Take the maximum out of the above two.
- If  $n$  is the required number of flip-flops, choose minimum value of  $n$  to satisfy the equation

$$\max(0s, 1s) \leq 2^{n-1}$$

3. Design the counter as usual. The output is at the Q or  $\bar{Q}$  lead of the LSB FF.

**EXAMPLE 12.8** Design a pulse train generator for the waveform shown below.



### Solution

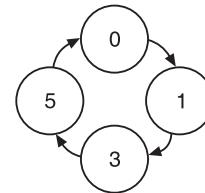
The pulse train is first copied down as 011101110... . It is a 4-bit sequence 0111 repeated. Write this sequence vertically in the LSB position as shown in Figure 12.67a and try to assign states using  $n$ , i.e. two FFs. This does not result in four unique states. Thus, two FFs are not sufficient even though there are only four bits in the sequence. So, increase the number of FFs by one, i.e. use  $n + 1$ , or three FFs and assign the states (000, 001, 011, 101) as shown in Figure 12.67a. Three FFs can have eight states. So the remaining four states (010, 100, 110, 111) are invalid and the entries for excitations corresponding to those states are don't cares.

The state diagram is shown in Figure 12.67b. The excitation requirements for JK FFs are shown in the excitation table of Figure 12.67c.

FF states		FF states		Decimal equivalent
LSB		LSB		
0	0	0	0	0
0	1	0	0	1
1	1	0	1	3
?	1	1	0	5

With 2 FFs                      With 3 FFs

(a) State assignment



(b) State diagram

PS			NS			Required excitations					
$Q_3$	$Q_2$	$Q_1$	$Q_3$	$Q_2$	$Q_1$	$J_3$	$K_3$	$J_2$	$K_2$	$J_1$	$K_1$
0	0	0	0	0	1	0	x	0	x	1	x
0	0	1	0	1	1	0	x	1	x	x	0
0	1	1	1	0	1	1	x	x	1	x	0
1	0	1	0	0	0	x	1	0	x	x	1

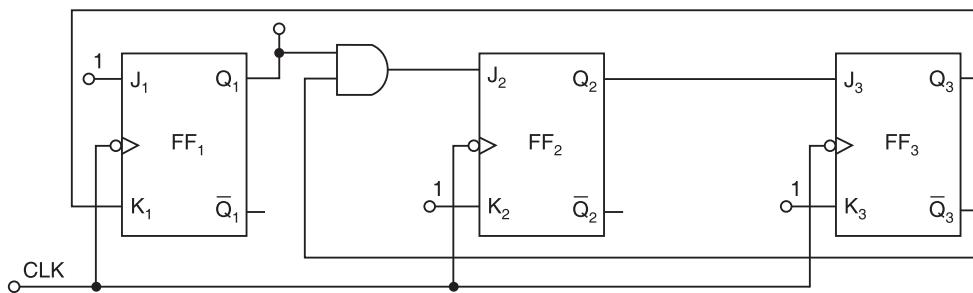
(c) Excitation table

**Figure 12.67** Example 12.8: Pulse train generator.

Drawing the K-maps for excitations of the FFs in terms of the present state variables, i.e. the outputs of the FFs and minimizing them, the minimal expressions for excitations of the FFs are:

$$J_3 = Q_2, K_3 = 1; \quad J_2 = \bar{Q}_3 Q_1, K_2 = 1; \quad J_1 = 1, K_1 = Q_3$$

The logic diagram based on these minimal expressions is shown in Figure 12.68.



**Figure 12.68** Example 12.8: Logic diagram of the pulse train generator.

Writing the table to test for lock-out, we see that the sequence generator is self-starting.

**EXAMPLE 12.9** Design a direct logic circuit to generate the following pulse trains.

(1)	0	1	0	0	0	0	1	0
(2)	1	0	1	1	0	1	1	0

### Solution

The pulse trains to be generated simultaneously are 100000 and 011011. The period of each pulse train is six time frames. The sequences are therefore 6 bits long. As there must be six unique states, the minimum number of FFs required is three. Try to assign the states assuming that pulse train (1) is to be available at  $Q_1$  output of  $FF_1$  and pulse train (2) at the  $Q_2$  output of  $FF_2$  as shown in Figure 12.69a.

Observe that it is not possible to assign the states using only three FFs as seen from the state assignment table. So, try with four FFs. It is now possible to assign the states (0001, 0010, 0110, 0000, 1010, 1110). Since four FFs can have 16 states, the remaining 10 states (0011, 0100, 0101, 0111, 1000, 1001, 1011, 1100, 1101, 1111) are invalid and the entries for excitations corresponding to those states are don't cares. The state assignment and the state diagram are shown in Figures 12.69a and b, respectively. The excitation requirements are shown in Figure 12.69c.

The design equations for the multiple pulse train generator using J-K FFs obtained by drawing the K-maps for the excitations based on the excitation table and minimizing them are:

$$\begin{array}{ll}
 J_4 = \bar{Q}_2 \bar{Q}_1 & K_4 = Q_3 \\
 J_3 = Q_2 & K_3 = 1 \\
 J_2 = 1 & K_2 = Q_3 \\
 J_1 = Q_4 Q_3 & K_1 = 1
 \end{array}$$

The logic diagram of the multiple pulse generator based on these minimal expressions is shown in Figure 12.70.

$Q_3$	$Q_2$	$Q_1$	$Q_4$	$Q_3$	$Q_2$	$Q_1$	States
0	0	1	0	0	0	1	1
0	1	0	0	0	1	0	2
1	1	0	0	1	1	0	6
0	0	0	0	0	0	0	0
?	1	0	1	0	1	0	10
?	1	0	1	1	1	0	14

With 3 FFs      With 4 FFs

(a) State assignment

(b) State diagram

PS				NS				Required excitations							
$Q_4$	$Q_3$	$Q_2$	$Q_1$	$Q_4$	$Q_3$	$Q_2$	$Q_1$	$J_4$	$K_4$	$J_3$	$K_3$	$J_2$	$K_2$	$J_1$	$K_1$
0	0	0	1	0	0	1	0	0	x	0	x	1	x	x	1
0	0	1	0	0	1	1	0	0	x	1	x	x	0	0	x
0	1	1	0	0	0	0	0	0	x	x	1	x	1	0	x
0	0	0	0	1	0	1	0	1	x	0	x	1	x	0	x
1	0	1	0	1	1	1	0	x	0	1	x	x	0	0	x
1	1	1	0	0	0	0	1	x	1	x	1	x	1	1	x

(c) Excitation table

Figure 12.69 Example 12.9: Multiple pulse train generator.

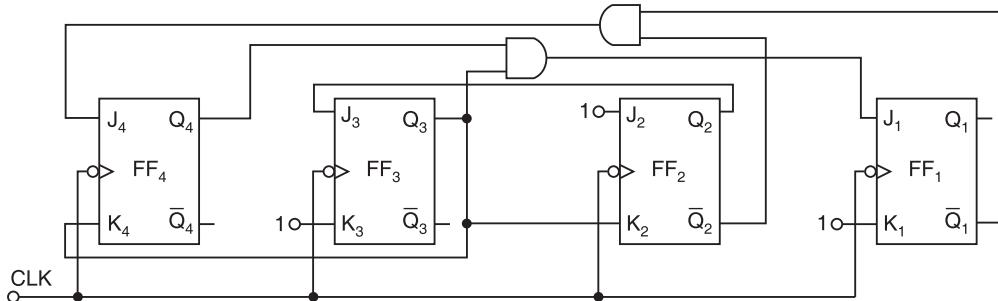


Figure 12.70 Example 12.9: Logic diagram of the multiple pulse train generator.

### 12.10.2 Indirect Logic

The block diagram of indirect logic is shown in Figure 12.71. Generation of pulse trains using indirect logic has the advantage that any counter (ripple or synchronous) with the correct number of states can form the generator. It is always assured that the number of FFs required,  $n$ , is such that  $N \leq 2^n$ . The gates can be used to detect the proper states when a 1 must be outputted. The same approach can be used for multiple outputs and multiple-output minimization can be used to reduce the logic.

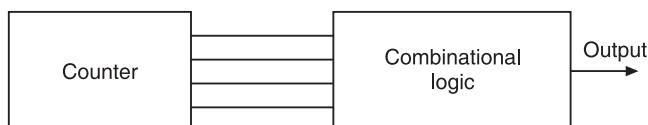


Figure 12.71 Block diagram of indirect logic.

**EXAMPLE 12.10** Generate the following pulse train using indirect logic.

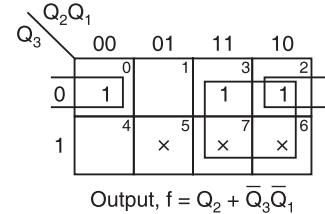
$\overline{1} \ 0 \ \boxed{1 \ 1} \ 0 \ \boxed{1 \ 0} \ \boxed{1 \ 1} \ 0 \ \boxed{1}$

**Solution**

The given sequence is 10110. It is written vertically under the column heading output (f) in the truth table of Figure 12.72a. It is 5 bits long. So we need five unique states to generate the above pulse train. So, any mod-5 counter can be used. For simplicity, we use a ripple counter. It goes through states 0, 1, 2, 3, 4, 0, ... . States 5, 6, 7 are invalid, so the corresponding outputs are don't cares. The K-map for the output f in terms of the outputs of the FFs, its minimization, and the minimal expression obtained from it are shown in Figure 12.72. The logic diagram (using a mod-5 ripple counter) based on that minimal expression for f is shown in Figure 12.73. While at state 0, it outputs a 1, i.e. the first bit of the sequence. While at state 1, it outputs a 0, i.e. the second bit of the sequence, and so on.

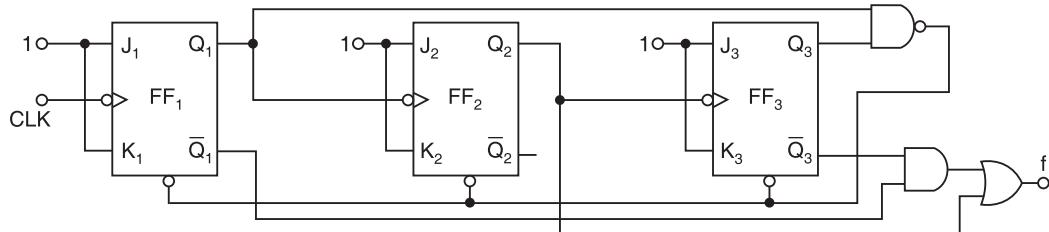
$Q_3$	$Q_2$	$Q_1$	Output (f)	States
0	0	0	1	0
0	0	1	0	1
0	1	0	1	2
0	1	1	1	3
1	0	0	0	4
1	0	1	x	5
1	1	0	x	6
1	1	1	x	7

(a) Truth table



(b) K-map

**Figure 12.72** Example 12.10: Pulse train generator.



**Figure 12.73** Example 12.10: Logic diagram of the pulse train generator.

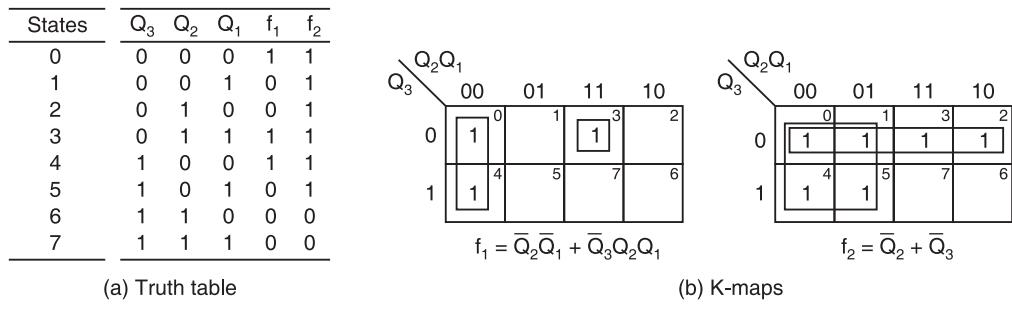
**EXAMPLE 12.11** Design a pulse generator using indirect logic to produce the following waveforms.

$\overline{0} \ \boxed{1} \ 0 \ 0 \ \boxed{1 \ 1} \ 0 \ 0 \ 0$   
 $\overline{0} \ \boxed{1 \ 1} \ 1 \ 1 \ 1 \ 1 \ 0 \ 0$

**Solution**

The pulse trains to be generated written vertically under column headings  $f_1$  and  $f_2$  in the truth table of Figure 12.74a are: (a) 10011000 and (b) 11111100. These are both eight bits long. So we need eight unique states to generate those two pulse trains. Therefore, a mod-8, i.e. a 3-bit ripple counter can be used. Let  $f_1$  and  $f_2$  be the outputs of the combinational

circuits. The state assignment is shown in the truth table. The K-maps for outputs  $f_1$  and  $f_2$  in terms of the outputs of the flip-flops, their minimization and the minimal expressions obtained from them are shown in Figure 12.74b. The logic diagram (using a mod-8 ripple counter) based on those minimal expressions for  $f_1$  and  $f_2$  is shown in Figure 12.75.



(a) Truth table

(b) K-maps

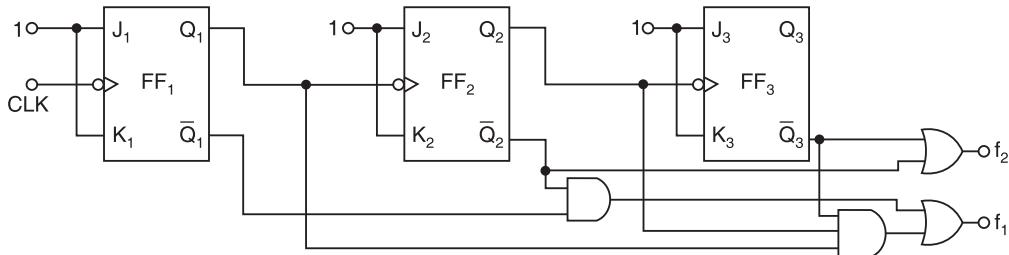
Figure 12.74 Example 12.11: Truth table and K-maps for  $f_1$  and  $f_2$ .

Figure 12.75 Example 12.11: Logic diagram of the pulse train generator.

## 12.11 PULSE GENERATORS USING SHIFT REGISTERS

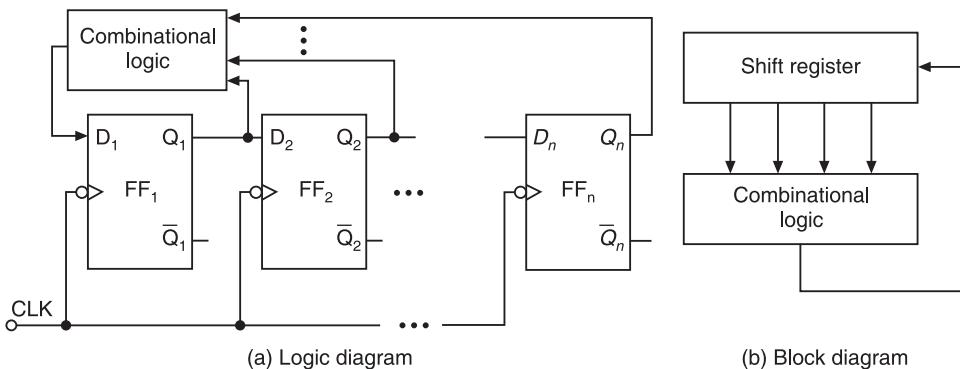
Shift registers can be used to generate single pulse trains. In general, they cannot be used for multiple pulse trains without being derailed. Shift registers can be adopted to serve as sequence generators. The length of a sequence is the number of bits in the sequence before the pattern of bits repeats itself. The sequence length is analogous to the period of a periodic analog waveform. In many applications, the length of the sequence is more important than the exact pattern of the repeated bits. Since the generator repeats the pattern over and over again, the sequence can be read in a number of ways depending on the starting point. If the sequence available at the complementary outputs of the FFs is read, a different pattern may be obtained.

A shift register is quite restrictive in the sense that it cannot go from any one state to any other state of our choice. For example, it cannot go from 0110 to 0101 directly; it can only go to 0011 or 1100 upon receipt of the next clock pulse. So, the pulse train is first examined to see if it can be generated by shifting.

The basic structure of a sequence generator is shown in Figure 12.76. Here  $n$  FFs are cascaded in the usual manner of a shift register. Type D FFs have been used and the clock signal is applied to all FFs. It is similar to a shift register counter but the difference is that in a ring counter, only the output of the last FF is given as data input to the first FF, whereas in a sequence generator, the

logic level of  $D_1$  is determined not only by  $Q_n$  but also by the output of other FFs in the cascade. That is,

$$D_1 = f(Q_1, Q_2, \dots, Q_n)$$

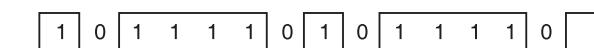


**Figure 12.76** Basic structure of a sequence generator.

The prescribed sequence appears at the output of each of the FFs. Of course, as we go from  $FF_1$  to  $FF_2$ , etc. we find successive delays by one clock interval. In order to build a sequence generator capable of generating a sequence of length  $S$ , it is necessary to use at least  $n$  FFs, where  $n$  satisfies the condition  $S \leq 2^n - 1$ . If the order of 1s and 0s in the sequence is prescribed, generally it will not be possible to generate a length  $S$  in which the minimum number of FFs is used. On the other hand, for any  $n$ , there is at least one sequence for which the sequence length  $S$  is a maximum, i.e.  $2^n - 1$ . The design procedure is as follows:

From the information of the waveform (or the sequence of bits, if given), decide the number of FFs required for pulse generation. The minimum number of FFs required for  $N$  states is  $n$  such that  $N \leq 2^n$ . Convert the waveform to 1s and 0s. Write the bits in vertical order. Form groups of  $n$  bits starting from the top bit and convert them to state numbers (decimal) moving down one bit at a time. The procedure is repeated until the cycle is completed. If the entire period of the train can be examined without repeating a state, the job can be done by  $n$  FFs. If there is a repetition of states, try  $(n + 1)$  FFs. If repeated states still occur, try  $(n + 2)$  FFs, and so on.

**EXAMPLE 12.12** Design a pulse train generator using a shift register to generate the following waveform.



### Solution

The given sequence 1011110 is of 7-bit length. So seven unique states are required. So, the minimum number of FFs required is three. Write the sequence 1011110 in vertical form and make groups of three bits starting from the top bit and write the states in decimal as shown in Figure 12.77a.

In 3-bit groups, states 7 and 5 are repeated. So, we cannot get seven unique states using three FFs. Next, make groups of four bits as shown in Figure 12.77b. The states are not repeated, i.e. seven unique states (11, 7, 15, 14, 13, 10, 5, that is 1011, 0111, 1111, 1110, 1101, 1010, 0101) can be obtained using four FFs. Make the truth table with the states of

the register, and the output of the combinational circuit which is to be fed as input to the shift register. For each state corresponding to the particular group of four bits, the next lower bit in the vertical order represents the output of the combinational circuit. For the first state, i.e. 1011 ( $11_{10}$ ), the next lower bit in the column is 1. So, the output of the combinational circuit for that state will be a 1. For the next state, i.e. 0111 ( $7_{10}$ ), the next lower bit in the column is 1. So, the output of the combinational circuit for that state will be a 1, and so on. Four FFs can have 16 states. So the remaining nine states (0, 1, 2, 3, 4, 6, 8, 9, 12, that is 0000, 0001, 0010, 0011, 0100, 0110, 1000, 1001, 1100) are invalid. Form the truth table and draw the K-map for  $f$ —the output of the combinational circuit in terms of the outputs of the FFs. The minimization of the K-map, the minimal expression obtained from it, and the realization of the logic diagram based on that minimal expression are shown in Figure 12.78.

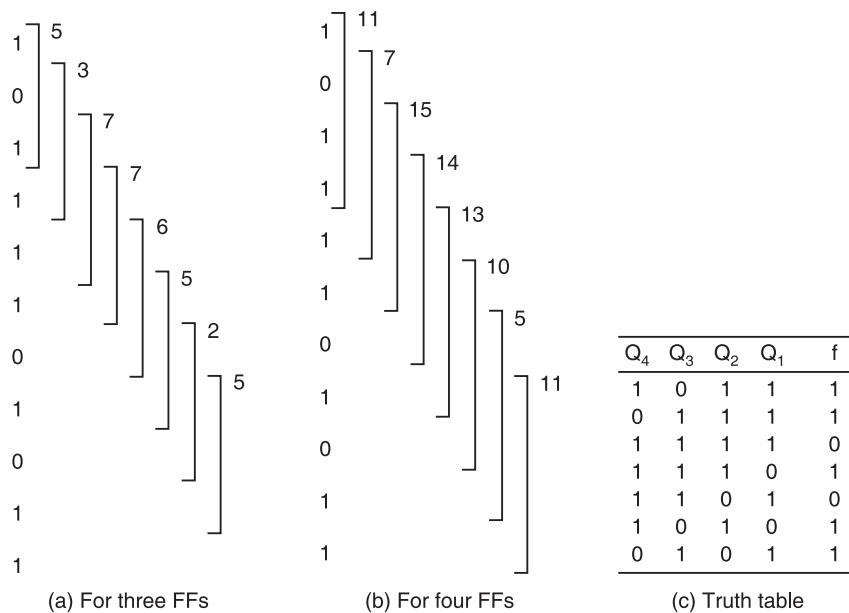


Figure 12.77 Example 12.12: State assignment for the pulse train generator.

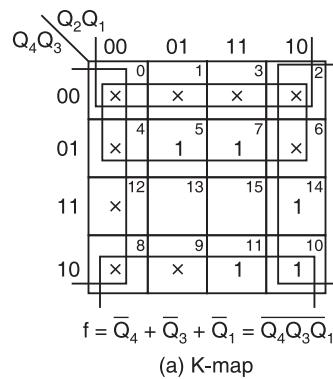
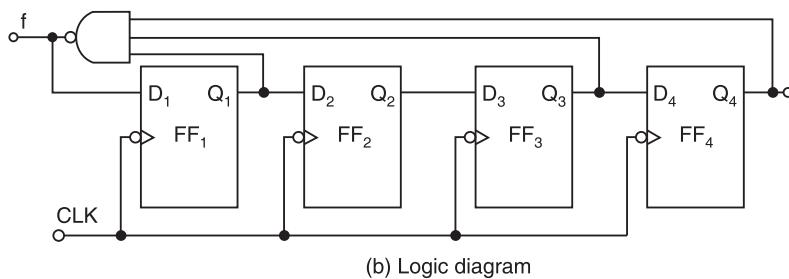
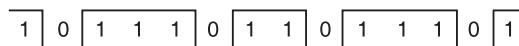


Figure 12.78 Example 12.12: Pulse train generator (Contd.)...



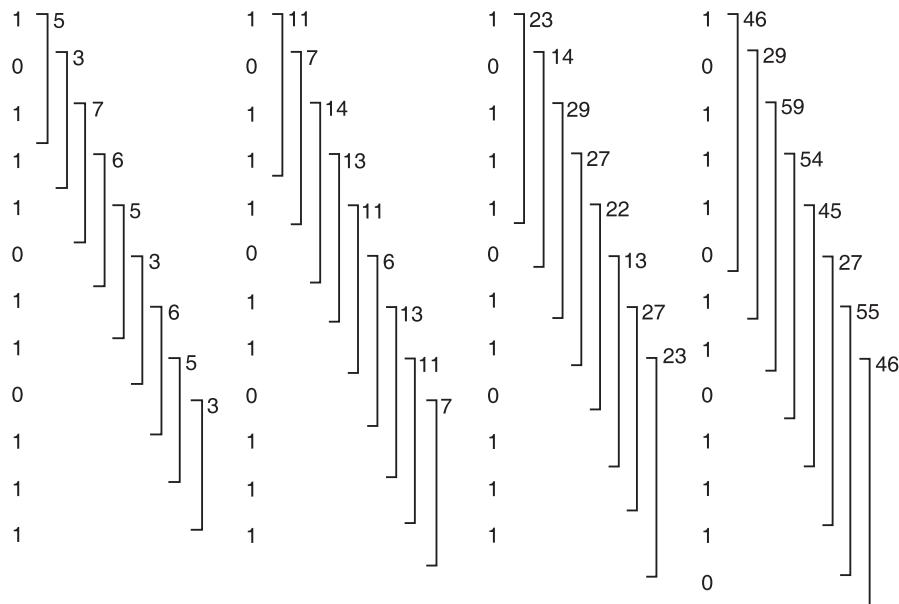
**Figure 12.78** Example 12.12: Pulse train generator.

**EXAMPLE 12.13** Design a pulse generator using a shift register to generate the waveform shown below.



**Solution**

The given pulse train is 1011101. There are seven states. So, the minimum number of FFs required is three. Make groups of three bits and form the states as shown in Figure 12.79a.



(a) For three FFs

(b) For four FFs

(c) For five FFs

(d) For six FFs

**Figure 12.79** Example 12.13: State assignment for the pulse train generator.

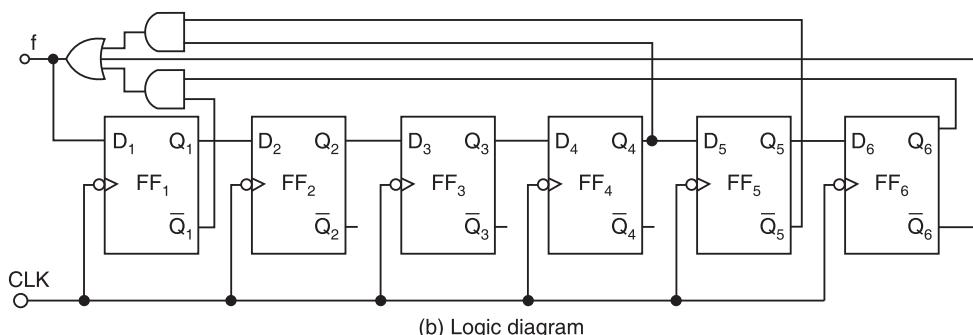
As seen from Figures 12.79a, b and c, it is not possible to generate seven unique states using three or four or even five FFs. So, the required number of FFs is six. The corresponding truth table is shown in Figure 12.80a. The six FFs can have 32 states. So, the remaining 25 states are invalid. Drawing a 6-variable K-map for f in terms of the outputs of the FFs and minimizing it, the minimal expression for f is

$$f = \bar{Q}_6 + \bar{Q}_5 Q_4 + Q_6 \bar{Q}_1$$

The logic diagram of the pulse generator based on that minimal expression is shown in Figure 12.80b.

$Q_6$	$Q_5$	$Q_4$	$Q_3$	$Q_2$	$Q_1$	$f$
0	1	1	1	0	1	1
1	0	1	1	1	0	1
1	1	0	1	1	1	0
0	1	1	0	1	1	1
1	0	1	1	0	1	1
1	1	0	1	1	0	1
1	1	1	0	1	1	0

(a) Truth table



(b) Logic diagram

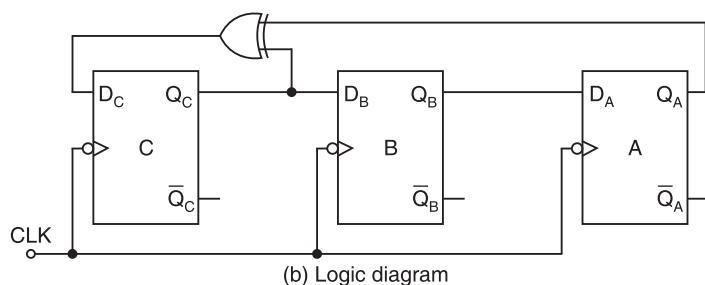
Figure 12.80 Example 12.13: Pulse train generator.

## 12.12 LINEAR SEQUENCE GENERATOR

One of the disadvantages of using a shift register as a state generating device is, that it seems very difficult to generate very many unique states. There is one type of circuit called the *linear sequence generator* that can generate  $2^n - 1$  unique states for an  $n$ -FF shift register. It will generate all states except the state number (decimal) 0. The sequence of states of a 3-FF shift register as a sequence generator is shown in Figure 12.81. It can generate a 7-bit linear sequence 0100111. This is accomplished by exclusively ORing A and C outputs of the shift register and feeding this XOR output as input to flip-flop C. There is no guarantee that any other 7-bit sequence can be generated by this 3-bit shift register.

A	B	C	Decimal	$A \oplus C$
1	1	1	7	0
1	1	0	6	1
1	0	1	5	0
0	1	0	2	0
1	0	0	4	1
0	0	1	1	1
0	1	1	3	1

(a) Sequence table



(b) Logic diagram

Figure 12.81 Sequence generator using three type D flip-flops.

The expressions for feedback signals for particular lengths of FFs are shown in Table 12.5.

**Table 12.5** Expressions for feedback signals for particular lengths of flip-flops

Shift register length	Expression for feedback	Shift register length	Expression for feedback
1	A	11	A/C
2	A $\oplus$ B	12	A $\oplus$ B $\oplus$ C $\oplus$ K
3	A $\oplus$ C	13	A $\oplus$ B $\oplus$ C $\oplus$ M
4	A $\oplus$ B	14	A $\oplus$ B $\oplus$ C $\oplus$ M
5	A $\oplus$ C	15	A $\oplus$ B
6	A $\oplus$ B	16	A $\oplus$ C $\oplus$ D $\oplus$ F
7	A $\oplus$ B	17	A $\oplus$ D
8	A $\oplus$ E $\oplus$ F $\oplus$ G	18	A $\oplus$ H
9	A $\oplus$ E	19	A $\oplus$ B $\oplus$ C $\oplus$ F
10	A $\oplus$ D	20	A $\oplus$ D

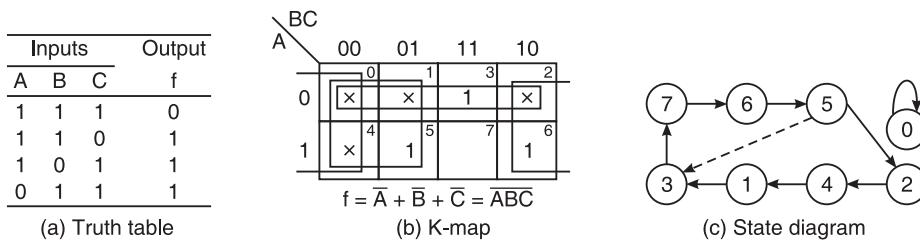
A linear sequence can be modified to reduce the number of states to any size by using the following rules:

1. Knowing the number of desired states,  $N$ , find the smallest number of FFs,  $n$ , needed for that number of states, such that  $N \leq 2^n$ .
2. Draw the state diagram for a linear sequence generator using  $n$  FFs.
3. Examine all pairs of states separated by  $2^n - N - 2$  states and locate that pair in which the smaller state number (decimal) is an even number and one less than the larger state number (decimal).
4. Find the state number (decimal) that precedes the smaller state number of the above pair. Draw an arrow from that state number to the larger state number of the pair. This modified state diagram will be the final state diagram of the sequence.

**EXAMPLE 12.14** Modify a 3-bit linear sequence generator to output 4 states.

#### Solution

It requires three FFs because the state number 0 is not permitted. So, draw a state diagram for a 3-bit linear sequence generator (Figure 12.82c) and locate that pair of state numbers, separated by  $2^n - N - 2 = 8 - 4 - 2 = 2$  states, in which the smaller state number (decimal) of the pair is even and one less than the larger state number (decimal).



**Figure 12.82** Example 2.14: Modification of the linear sequence generator.

The following pairs of state numbers separated by two states in between are examined to find the difference between their decimal values.

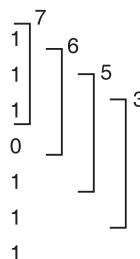
States 7–2 differ by more than 1.

States 6–4 differ by more than 1.

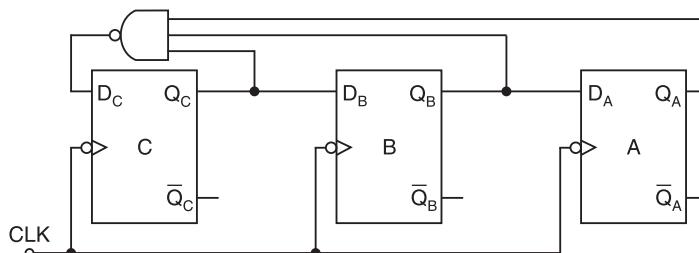
States 5–1 differ by more than 1.

States 2–3 differ by 1. So, this pair fits.

Draw an arrow from the state before 2, that is state 5 to state 3. The required states are, therefore,  $7 \rightarrow 6 \rightarrow 5 \rightarrow 3$ . The truth table is shown in Figure 12.82a. The K-map and its minimization are shown in Figure 12.82b. The value of f, i.e. the output of the combinational circuit for each state is taken as the next lower bit in the sequence as shown below.



The modified logic diagram based on the value of f is shown in Figure 12.83.



**Figure 12.83** Example 12.14: Modified 3-bit linear sequence generator.

### SHORT QUESTIONS AND ANSWERS

1. What is a counter?
- A. A digital counter is a set of flip-flops interconnected such that their combined state at any time is the binary equivalent of the total number of pulses applied up to that time, i.e. it is a logic circuit used to count the number of pulses.
2. What are the applications of counters?
- A. Counters are used to count pulses. They can also be used as frequency dividers. They are also used to perform the timing function as in digital watches, to create time delays, to produce non-sequential binary counts, to generate pulse trains, and to act as frequency counters, etc.
3. What are the two basic types of counters?
- A. The two basic types of counters are (a) asynchronous counters and (b) synchronous counters.

4. What is the other name of asynchronous counters? Why is that name?  
A. Asynchronous counters are also called ripple counters. This is because in these counters the transition of the first stage ripples through to the last stage. Ripple counters are also called series counters or serial counters.
5. What are the advantages and disadvantages of ripple counters?  
A. The ripple counter is the simplest type of counter, easiest to design and requires the least amount of hardware but has the disadvantage that unwanted spikes will be present. Also ripple counters are serial counters and hence are slow. If the clock frequency is high owing to accumulation of propagation delay skipping of states may occur.
6. What is the basic difference between asynchronous and synchronous counters?  
A. Basically asynchronous counters are serial or series counters, whereas synchronous counters are parallel counters. The external clock is applied to only the first flip-flop in asynchronous counters, whereas the clock is applied to all the flip-flops simultaneously in synchronous counters.
7. Define an up-counter, a down-counter and an up-down counter.  
A. An up-counter is a counter which counts in the upward direction, i.e. 0, 1, 2, 3,..., N. A down-counter is a counter which counts in the downward direction, i.e. N, N - 1, N - 2, ..., 1, 0. An up-down counter is a counter which can count in upward as well as downward directions.
8. What do you mean by state of the counter?  
A. Each of the counts of the counter is called the state of the counter.
9. What is the modulus of a counter?  
A. The number of states through which the counter passes before returning to the starting state is called the modulus of the counter. Hence the modulus of a counter is equal to the total number of distinct states (counts) including zero that a counter can store. In other words, the number of input pulses that causes the counter to reset to its initial count is called the modulus of the counter.
10. What is a shortened modulus counter?  
A. A shortened modulus counter is a counter which does not utilize all the possible states. In this some of the states are unutilized, i.e. invalid.
11. What is a full modulus counter?  
A. A counter which goes through all the possible states before restarting is called the full modulus counter. There are no invalid states.
12. What is a variable modulus counter?  
A. A counter in which the maximum number of states can be changed is called the variable modulus counter.
13. What do you mean by terminal count?  
A. The final state of the counter sequence is called the terminal count.
14. What is full modulus cascading?  
A. If the overall modulus is the product of the individual modulus of all the cascaded counters, such cascading is called full modulus cascading.
15. Compare asynchronous and synchronous counters.  
A. Asynchronous counters are serial or series counters whereas synchronous counters are parallel counters. Asynchronous counters are slow and synchronous counters are fast. In asynchronous counters clock signal is applied only to the first flip-flop, whereas in synchronous counters clock signal is applied to all the flip-flops simultaneously. In asynchronous counters propagation delays of individual flip-flops add together and if the clock frequency is very high, owing to accumulation

of propagation delay skipping of states may occur. In synchronous counters, propagation delays of individual flip-flops do not add together and hence no problem of skipping of states. Asynchronous counters have more decoding problems but synchronous counters have less severe decoding problems. In asynchronous counters, the invalid states are bypassed by providing suitable feedback. In synchronous counters, the invalid states are taken care of by treating the corresponding excitations as don't cares. Asynchronous counters require less circuitry and synchronous counters require more circuitry.

**16. What is problem of lock-out?**

- A. Sometimes when the counter is switched on, or any time during counting, because of noise spikes the counter may find itself in some unused (invalid) state. Subsequent clock pulses may cause the counter to move from one unused state to another unused state and the counter may never come to a valid state. So the counter becomes useless.

A counter whose unused states have this feature is said to suffer from the problem of lockout.

**17. Which counter may suffer from the problem of lock-out?**

- A. A shortened modulus counter may suffer from the problem of lock-out.

**18. What do you mean by self-starting type counter?**

- A. A counter is said to be of self-starting type if it returns to a valid state and counts normally after one or more clock pulses, even if it enters an invalid state.

**19. How is the problem of lock-out eliminated?**

- A. To eliminate the problem of lock-out, external logic circuitry is provided which ensures by properly resetting each flip-flop that at start up the counter is in its initial state. The logic circuitry presetting the counter to its initial state can be provided either by obtaining an expression for reset/preset for the flip-flops or by modifying the design such that the counter goes from each invalid state to the initial state after the clock pulse.

**20. What is a hybrid counter?**

- A. A hybrid counter is a counter in which the output of a synchronous counter drives the clock input of another counter to get a divide-by- $N$  operation.

**21. What are shift register counters? Why is that name?**

- A. Shift register counters are counters obtained from serial-in, serial-out shift registers by providing feedback from the output of the last flip-flop to the input of the first flip-flop. These devices are called counters because they exhibit a specified sequence of states.

**22. What is a ring counter? Why is that name?**

- A. A ring counter also called the basic ring counter is the most widely used shift register counter. It is a serial-in, serial-out shift register in which the Q output of the last stage is connected back to the D input of the first stage. Since the array of flip-flops is arranged in a ring, it is called a ring counter.

**23. What is a twisted-ring counter?**

- A. A twisted-ring counter also called the Johnson counter or the switch-tail ring counter is a serial-in, serial-out shift register in which the  $\bar{Q}$  output of the last stage is connected back to the D input of the first stage.

**24. What is the advantage and disadvantage of ring counter compared to ripple counter?**

- A. An  $n$ -bit ring counter can count only  $n$  bits, whereas an  $n$ -bit ripple counter can count  $2^n$  bits. So the ring counter is uneconomical compared to a ripple counter, but has the advantage of requiring no decoder, since we can read the counter by simply noting which flip-flop is set. Since it is entirely a synchronous operation and requires no gates external to flip-flops, it has the further advantage of being very fast. An  $n$ -bit Johnson counter can have  $2n$  unique states and can count upto  $2n$  pulses.

So it is a mod- $2n$  counter. It is more economical than a normal ring counter but less economical than a ripple counter. It requires two input gates for decoding regardless of the size of the counter. Thus, it requires more decoding circuitry than that by the normal ring counter, but less than that by the ripple counter. Both types of ring counters suffer from the problem of lock-out.



### REVIEW QUESTIONS

1. With general block diagrams, show how each of the following counters can be obtained using flip-flop, a decade counter and a 4-bit binary counter or any combination of these.
 

(a) Divide-by-40 counter	(b) Divide-by-32 counter
(c) Divide-by-160 counter	(d) Divide-by-320 counter
(e) Divide-by-10,000 counter	(f) Divide-by-1000 counter
2. With general block diagrams, show how to obtain the following frequencies from a 1 MHz clock using single flip-flops, mod-5 counters and decade counters.
 

(a) 500 kHz	(b) 250 kHz	(c) 125 kHz	(d) 40 kHz
(e) 10 kHz	(f) 1 kHz		
3. How do you test for the problem of lock-out of a counter? How do you eliminate this problem?
4. Write the design steps of synchronous counters.

### FILL IN THE BLANKS

1. A circuit used to count the number of pulses is called a \_\_\_\_\_.
2. The two basic types of counters are (a) \_\_\_\_\_ counters and (b) \_\_\_\_\_ counters.
3. Asynchronous counters are also called \_\_\_\_\_ counters.
4. Ripple counters are \_\_\_\_\_ counters and hence are slow.
5. Synchronous counters are \_\_\_\_\_ counters and hence are fast.
6. Each of the counts of the counter is called the \_\_\_\_\_ of the counter.
7. The number of states through which the counter passes before returning to the starting state is called the \_\_\_\_\_ of the counter.
8. A \_\_\_\_\_ counter is a counter which does not utilize all the possible states.
9. A counter which has no invalid states is called a \_\_\_\_\_ counter.
10. A counter in which the maximum number of states can be changed is called a \_\_\_\_\_ counter.
11. The final state of the counter sequence is called the \_\_\_\_\_ count.
12. Synchronous counters are \_\_\_\_\_ than asynchronous counters.
13. Synchronous counters require \_\_\_\_\_ circuitry than asynchronous counters.
14. \_\_\_\_\_ means the counter keeps going from one invalid state to another when clock pulses are applied.
15. \_\_\_\_\_ counters may suffer from the problem of lock-out.
16. \_\_\_\_\_ is the most widely used shift register counter.
17. A twisted-ring counter is also called a \_\_\_\_\_ counter.
18. A \_\_\_\_\_ ring counter is more economical than a \_\_\_\_\_ ring counter.
19. An  $n$ -bit ring counter is a mod \_\_\_\_\_ counter whereas an  $n$ -bit twisted ring counter is a mod \_\_\_\_\_ counter.
20. A \_\_\_\_\_ ring counter requires no decoding circuitry.
21. Determination of the decimal equivalent of the binary outputs is called \_\_\_\_\_ a counter.

22. The terminal count of a 5-bit counter in the up-mode is \_\_\_\_\_ and in the down mode it is \_\_\_\_\_.
  23. The maximum modulus of a counter with seven flip-flops is \_\_\_\_\_.
  24. Undesired voltage spikes of short duration produced at the outputs of the decoder are called \_\_\_\_\_.
  25. Bringing the counter to a specific state at the beginning of the count is called \_\_\_\_\_ the counter.
  26. The number of flip-flops required in a counter to count 68 pulses is \_\_\_\_\_.
  27. \_\_\_\_\_ switching circuits have no memory.
  28. \_\_\_\_\_ switching circuits require memory elements.
  29. Sequential circuits which have a master oscillator are called \_\_\_\_\_ sequential circuits.
  30. Sequential circuits in which there is no master oscillator are called \_\_\_\_\_ sequential circuits.

## **OBJECTIVE TYPE QUESTIONS**

## PROBLEMS

- 12.1** Design the following counters:

  - (a) A mod-7 asynchronous counter using J-K FFs
  - (b) A mod-12 asynchronous counter using J-K FFs
  - (c) A mod-9 synchronous counter using J-K FFs
  - (d) A mod-7 synchronous counter using S-R FFs
  - (e) A mod-12 synchronous counter using T FFs
  - (f) A mod-14 synchronous counter using D FFs

**12.2** Design a J-K counter that goes through states 2, 4, 5, 7, 2, 4, ... Is the counter self-starting? Modify the circuit such that whenever it goes to any invalid state it comes back to state 2.

**12.3** Design the following types of counters:

  - (a) A type-D counter that goes through states 0, 2, 4, 6, 0, ... . The undesired states must always go to a 0 on the next clock pulse.

**688 FUNDAMENTALS OF DIGITAL CIRCUITS**

- (b) A type-T counter that goes through states 6, 3, 7, 8, 2, 9, 1, 12, 14, 0, 6, 3, ... .  
Is the counter self-starting?
- (c) A type-T counter that goes through states 0, 5, 4, 2, 0, ... . Is the counter self-starting?
- (d) A type T counter that must go through states 0, 2, 4, 6, 0, ... if the control line is HIGH, and through states 0, 4, 2, 7, 0, ... if the control line is LOW.

**12.4** Design the following types of generators:

- (a) A sequence generator to generate the sequence 1011010...
- (b) A pulse train generator to generate the sequence 1100010...
- (c) A pulse generator using indirect logic to produce the pulse trains 10000111 and 10011010.
- (d) A shift register pulse train generator to generate the pulse train 101110.
- (e) A shift register pulse train generator to generate the pulse train 11110010.

**12.5** Design a direct logic circuit to generate the sequences 100111 and 011101 simultaneously.

**12.6** Generate the pulse train 100110 using indirect logic.

**12.7** Design a BCD up/down counter using S-R FFs.

**12.8** Design an up/down counter using D FFs to count 0, 3, 2, 6, 4, 0, ... .

## VHDL PROGRAMS

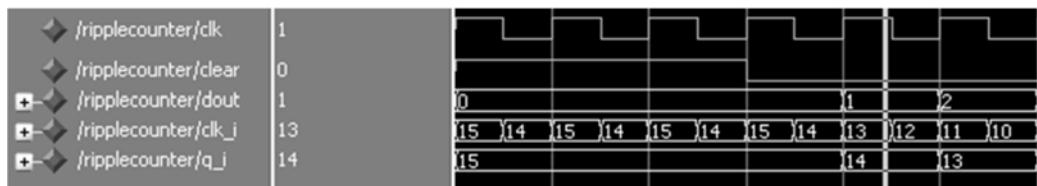
### 1. VHDL PROGRAM FOR RIPPLE COUNTER

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ripple_counter is
    generic (n : natural := 4);
    port (
        clk      : in std_logic;
        clear   : in std_logic;
        dout    : out std_logic_vector(n-1 downto 0) );
end ripple_counter;
architecture Behavioral of ripple_counter is
    signal clk_i, q_i    : std_logic_vector(n-1 downto 0);
begin
    clk_i(0)    <= clk;
    clk_i(n-1 downto 1) <= q_i(n-2 downto 0);
    gen_cnt: for i in 0 to n-1 generate
        dff: process(clear, clk_i)
            begin
                if (clear = '1') then
                    q_i(i) <= '1';
                elsif (clk_i(i)'event and clk_i(i) = '1') then
                    q_i(i) <= not q_i(i);
                end if;
            end process dff;
        end generate;
        dout <= not q_i;
    end Behavioral;

```

### SIMULATION OUTPUT:



### 2. VHDL PROGRAM FOR 4-BIT UP / DOWN COUNTER

```

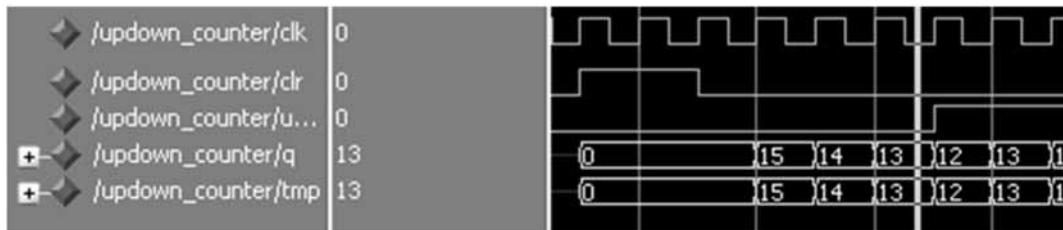
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

```

## 690 FUNDAMENTALS OF DIGITAL CIRCUITS

```
entity updown_counter is
    port(clk, clr, UP_DOWN : in std_logic;
         Q : out std_logic_vector(3 downto 0));
end updown_counter;
architecture Behavioral of updown_counter is
    signal tmp: std_logic_vector(3 downto 0);
begin
    process (clk, clr)
    begin
        if (clr='1') then
            tmp <= "0000";
        elsif (clk'event and clk='1') then
            if (UP_DOWN='1') then
                tmp <= tmp + 1;
            else
                tmp <= tmp - 1;
            end if;
        end if;
    end process;
    Q <= tmp;
end Behavioral;
```

### SIMULATION OUTPUT:



### 3. VHDL PROGRAM FOR MOD-10 GRAY CODE COUNTER

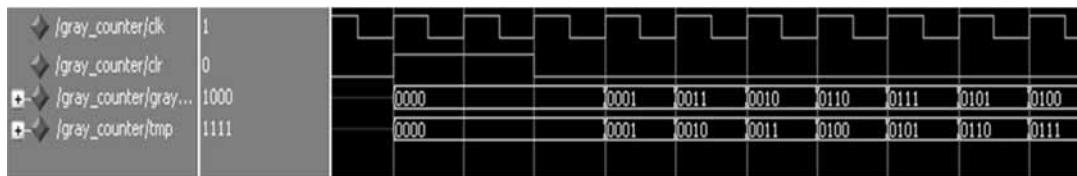
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
entity gray_counter is
port(clk, clr : in std_logic;
      gray_cnt : out std_logic_vector(3 downto 0));
end gray_counter;
architecture Behavioral of gray_counter is
signal tmp: std_logic_vector(3 downto 0);
begin
    process (clk, clr)
    begin
```

```

if (clr='1') then
    tmp <= "0000";
elsif (clk'event and clk='1') then
    tmp <= tmp + 1;
end if;
end process;
gray_cnt(3) <= tmp(3);
gray_cnt(2) <= tmp(3) xor tmp(2);
gray_cnt(1) <= tmp(2) xor tmp(1);
gray_cnt(0) <= tmp(1) xor tmp(0);

end Behavioral;

```

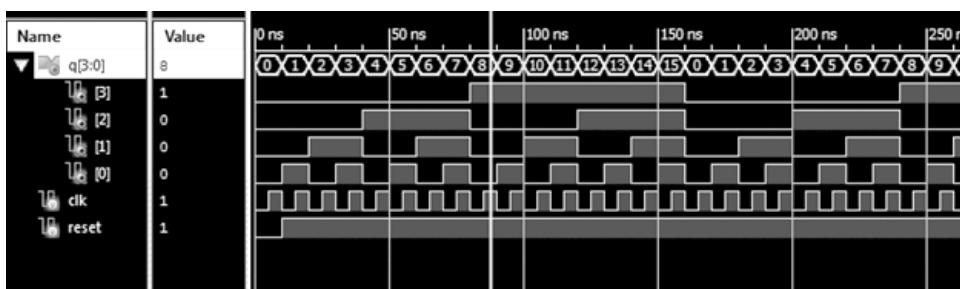
**SIMULATION OUTPUT:****VERILOG PROGRAMS****1. VERILOG PROGRAM IN STRUCTURAL MODELING FOR 4-BIT RIPPLE COUNTER**

```

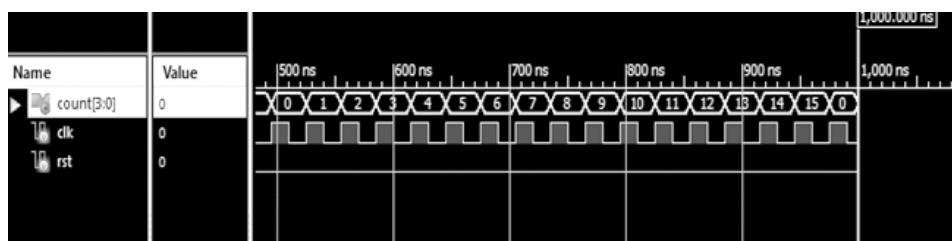
module TFF(q, clk, reset);
output q;
input clk, reset;
reg q;
always @ (negedge clk or negedge reset)
if (~reset)
q = 1'b0;
else
q = (~q);
endmodule

module ripple_counter(q, clk, reset);
output [3:0] q;
input clk, reset;
TFF tff0(q[0],clk,reset);
TFF tff1(q[1],q[0],reset);
TFF tff2(q[2],q[1],reset);
TFF tff3(q[3],q[2],reset);
endmodule

```

**SIMULATION OUTPUT:****2. VERILOG PROGRAM FOR 4-BIT SYNCHRONOUS UP COUNTER**

```
module counter(clk,rst,count);
input clk,rst;
output reg [3:0] count;
always@(posedge clk or posedge rst)
begin
if(rst)
count <= 4'b0000;
else if (clk)
count <= count + 1'b1;
else
count <= 4'b0000;
end
endmodule
```

**SIMULATION OUTPUT:****3. VERILOG PROGRAM FOR 4-BIT SYNCHRONOUS DOWN COUNTER**

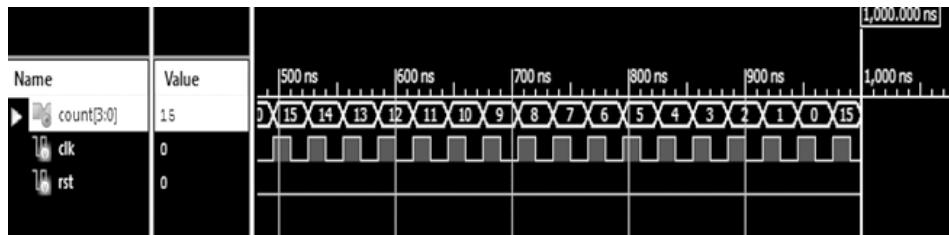
```
module counter(clk,rst,count);
input clk,rst;
output reg [3:0] count;
always@(posedge clk or posedge rst)
begin
if(rst)
```

```

count <= 4'b1111;
else if (clk)
count <= count - 1'b1;
else
count <= 4'b0000;
end
endmodule

```

### SIMULATION OUTPUT:



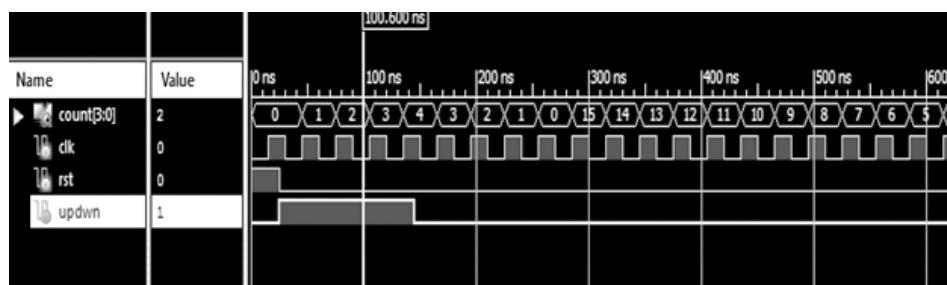
### 4. VERILOG PROGRAM FOR 4-BIT SYNCHRONOUS UP/DOWN COUNTER

```

module counter(clk,rst,updwn,count);
input clk,rst,updwn;
output reg [3:0] count;
always@(posedge clk or posedge rst)
begin
if(rst)
count <= 4'b0000;
else if (updwn)
count <= count + 1'b1;
else
count <= count - 1'b1;
end
endmodule

```

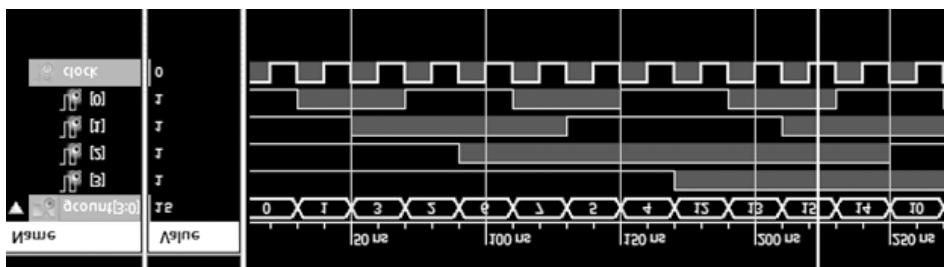
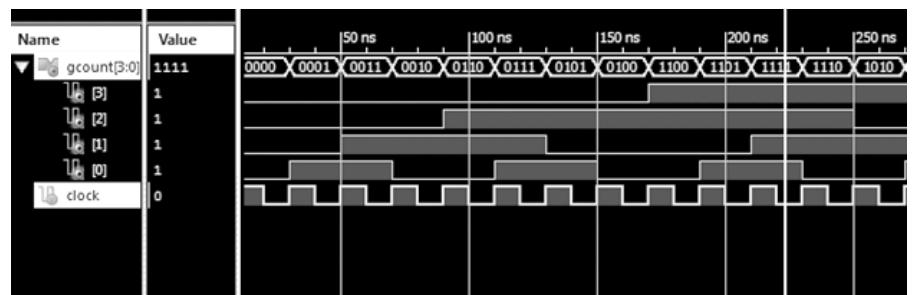
### SIMULATION OUTPUT:



## 5. VERILOG PROGRAM FOR 4-BIT GRAY COUNTER USING BEHAVIORAL MODELING

```
module Gray_Counter (clock,gcount);
input clock;
output reg [3:0] gcount;
reg [3:0] count=4'b0000;
always @ (posedge clock)
begin
count <= count + 1;
gcount = {(count[3]),(count[3] ^ count[2]), (count[2] ^ count[1]),(count[1] ^ count[0]) };
end
endmodule
```

### SIMULATION OUTPUT:



## 6. VERILOG PROGRAM FOR MOD-10 COUNTER USING BEHAVIORAL MODELING

```
module Mod_Ten_Counter(clk,rst,counter);
input clk,rst;
output reg [3:0]counter;
always@(posedge clk or posedge rst)
begin
if(rst)
counter <= 4'b0000;
```

```

else if (counter == 4'b1001)
counter <= 4'b0000;
else
counter <= counter + 1;
end
endmodule

```

### SIMULATION OUTPUT:



### 7. VERILOG PROGRAM FOR EVEN COUNTER

```

module sequence(clk,rst,cnt);
input clk,rst;
output [4:0] cnt;
reg [4:0] cnt;
always@(posedge clk or posedge rst)
begin
if (rst) cnt <= 5'b00000;
else if( cnt == 5'd20)
cnt <= 5'b00000;
else
cnt <= cnt + 5'b00010;
end
endmodule

```

### SIMULATION OUTPUT:

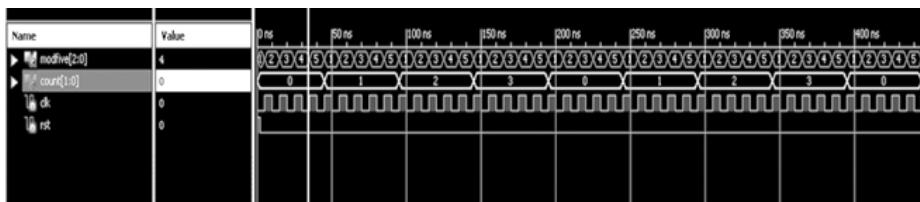


**8. VERILOG PROGRAM FOR COUNTING MOD-3 VALUE 5 TIMES**

```
module countseq (clk,rst,modfive,count);
input clk,rst;
output [2:0] modfive;
reg [2:0] modfive;
output [1:0] count;
reg [1:0] count;
wire s1;
wire [2:0] b1,b2;
wire [1:0] b3,b4;

assign s1 = (modfive == 5);
assign b1 = modfive + 1;
assign b2 = s1 ? 1:b1;

always@ (posedge clk or posedge rst)
begin
if(rst) modfive <= 1;
else modfive<= b2;
end
```

**SIMULATION OUTPUT:****9. VERILOG PROGRAM IN BEHAVIORAL MODELING FOR 4-BIT RING COUNTER**

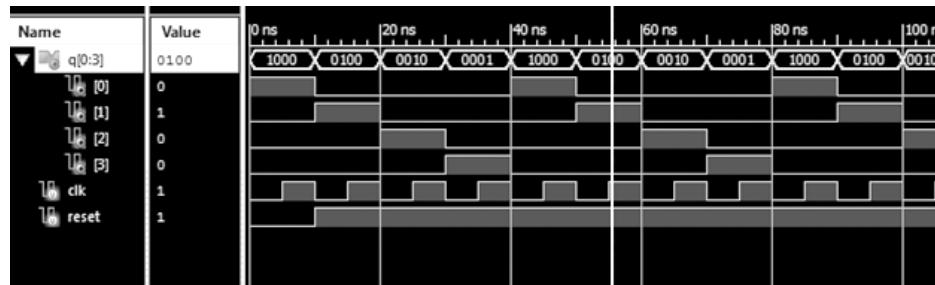
```
module ring_counter (q,clk,reset);
output [0 : 3]q;
input clk,reset;
reg [0 : 3] q;
always @ (negedge clk or negedge reset)
begin
if (~reset)
q <= 4'b 1000;
else if (reset)
begin
q[0] <= q[3];
q[1] <= q[0];
```

```

q[2] <= q[1];
q[3] <= q[2];
end
end
endmodule

```

### SIMULATION OUTPUT:



### 10. VERILOG PROGRAM FOR 4-BIT ALU

```

module alu(a,b,s,y);
input [3:0]a;
input [3:0]b;
input [2:0]s;
output [7:0]y;
reg [7:0]y;
always@(a,b,s)
begin
case(s)
3'b000:y=a+b;
3'b001:y=a-b;
3'b010:y=a&b;
3'b011:y=a|b;
3'b100:y=4'b1111^a;
3'b101:y=(4'b1111^a)+1'b1;
3'b110:y=a*b;
3'b111:begin y=a;
y=y>>1'b1;
end
endcase
end
endmodule

```

## 698 FUNDAMENTALS OF DIGITAL CIRCUITS

### SIMULATION OUTPUT:

Name	Value	300 ns	400 ns	500 ns	600 ns	700 ns
► [x] y[7:0]	00000010	00001001	00001101	00000010	00000011	01110101
► [x] a[3:0]	1101			1101		
► [x] b[3:0]	1001			1001		
▼ [x] s[2:0]	100	010	011	100	101	110
[2]	1					
[1]	0					
[0]	0					

# 13

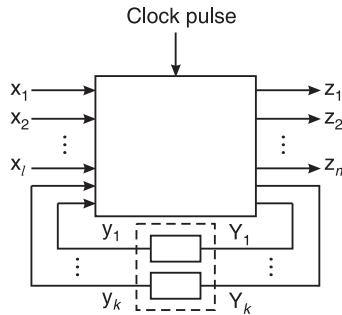
## SEQUENTIAL CIRCUITS-I

### 13.1 THE FINITE STATE MODEL

A finite state machine is an abstract model describing the synchronous sequential machine. The behaviour of a finite state machine is described as a sequence of events that occur at discrete instants designated as  $t = 1, 2, 3 \dots$  etc. Suppose that the machine has been receiving input signals and also responding by producing output signals. If, now at time  $t$ , we were to apply an input signal  $x(t)$  to the machine, the response  $z(t)$  would depend on  $x(t)$  as well as on the past inputs to the machine, and since a given machine might have an infinite varieties of possible histories, it would need an infinite capacity for storing them. Since in practice, it is impossible to implement machines which have infinite storage capabilities, we will concentrate on those machines whose past histories can affect their future behaviour only in a finite number of ways. These are called the *finite state machines*, that is, machines with a fixed number of states. These machines can distinguish among a finite number of classes of input histories. These classes of input histories are referred to as the internal states of the machine. Every finite state machine, therefore, contains a finite number of memory devices.

Figure 13.1 shows the schematic diagram of a synchronous sequential machine. The circuit has a finite number  $l$  of input terminals. The signals entering the circuit via these terminals constitute the set  $[x_1, x_2, x_3, \dots, x_l]$  of input variables. Each input variable may take one of two possible values, a 0 or a 1. An ordered set of  $l$  0s and 1s is an input configuration. The set  $p$  of all possible combinations of  $l$  inputs, i.e.  $p = 2^l$  is called an input alphabet  $I$ , and each one of these configurations is referred to as the symbol of the alphabet

$$I = (I_1, I_2, \dots, I_p)$$



**Figure 13.1** Block diagram of a finite state model.

Similarly, the circuit has a finite number  $m$  of output terminals which define the set  $[z_1, z_2, z_3, \dots, z_m]$  of output variables. Each output variable is a binary variable. An ordered set of  $m$  0s and 1s is an output configuration. The set  $q$  of all possible combinations of  $m$  outputs, i.e.  $q = 2^m$  is called the output alphabet and is given by

$$O = [O_1, O_2, O_3, \dots, O_q]$$

Each output configuration is called a symbol of the output alphabet.

The signal value at the output of each memory element is referred to as the state variable and the set  $[y_1, y_2, \dots, y_k]$  constitutes the set of state variables. The combination of values at the outputs of  $k$  memory elements  $y_1, y_2, \dots, y_k$  defines the present internal state or the present state of the machine. The set  $S$  of  $n = 2^k$  combinations of state variables constitutes the entire set of states of the machine.

$$S = [S_1, S_2, S_3, \dots, S_n]$$

The external inputs  $x_1, x_2, \dots, x_l$  and the values of the state variables  $y_1, y_2, \dots, y_k$  are supplied to the combinational circuit, which in turn produces outputs  $z_1, z_2, \dots, z_m$  and the values  $Y_1, Y_2, \dots, Y_k$ . The values of the  $Y$ s which appear at the output of the combinational circuit at time  $t$  determine the state variables at time  $t + 1$ , and therefore, the next state of the machine.

Synchronization is achieved by means of clock pulses. The clock pulses may be applied to various AND gates to which input signal is applied. This allows the gates to transmit signals only at instants which coincide with the arrival of clock pulses.

Before going for the design of sequential machines one should be familiar with the following things.

In synchronous or clocked sequential circuits, clocked flip-flops are used as memory elements, which change their individual states in synchronism with the periodic clock signal. Therefore, the change in states of flip-flops and change in state of the entire circuit occurs at the transition of the clock signal.

The synchronous or clocked sequential circuits are represented by two models.

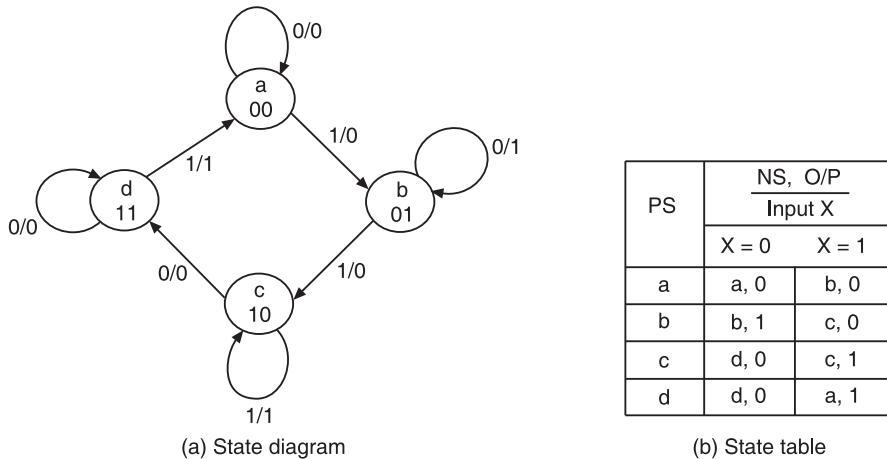
1. *Moore circuit*: In this model, the output depends only on the present state of the flip-flops.
2. *Mealy circuit*: In this model, the output depends on both the present state of the flip-flop(s) and the input(s).

Sequential circuits are also called finite state machines (FSMs). This name is due to the fact that the functional behaviour of these circuits can be represented using a finite number of states.

### 13.1.1 State Diagram

The state diagram or state graph is a pictorial representation of the relationships between the present state, the input, the next state, and the output of a sequential circuit, i.e. the state diagram is a pictorial representation of the behaviour of a sequential circuit.

Figure 13.2a shows the state diagram of a Mealy circuit. The state is represented by a circle also called the node or vertex and the transition between states is indicated by directed lines connecting the circles. A directed line connecting a circle with itself indicates that the next state is the same as the present state. The binary number inside each circle identifies the state represented by the circle. The directed lines are labelled with two binary numbers separated by a symbol /. The input value that causes state transition is labelled first and the output value that occurs when this input is applied during the present state is labelled after the symbol /.



**Figure 13.2** Mealy circuit.

In the case of a Moore circuit, the directed lines are labelled with only one binary number representing the input that causes the state transition. The output is indicated within the circle below the present state, because the output depends only on the present state and not on the input. Figure 13.3a shows the state diagram of a Moore circuit.

### 13.1.2 State Table

Even though the behaviour of a sequential circuit can be conveniently described using a state diagram, for its implementation the information contained in the state diagram is to be translated into a state table. The state table is a tabular representation of the state diagram. Figure 13.2b shows the state table for the state diagram shown in Figure 13.2a. Figure 13.3b shows the state table for the state diagram shown in Figure 13.3a. The present state designates the state of the flip-flops before the application of the clock pulse. The next state is the state of the flip-flops after the application of the clock pulse and the output section gives the value of output variables during the present state.

Both the state diagram and the state table contain the same information and the choice between the two representations is a matter of convenience. Both have the advantage of being precise, unambiguous, and thus, more suitable for describing the operation of a sequential machine than

that by any verbal description. The succession of states through which a sequential machine passes and the output sequence which it produces in response to a known input sequence, are specified uniquely by the state diagram or by the state table and initial state.

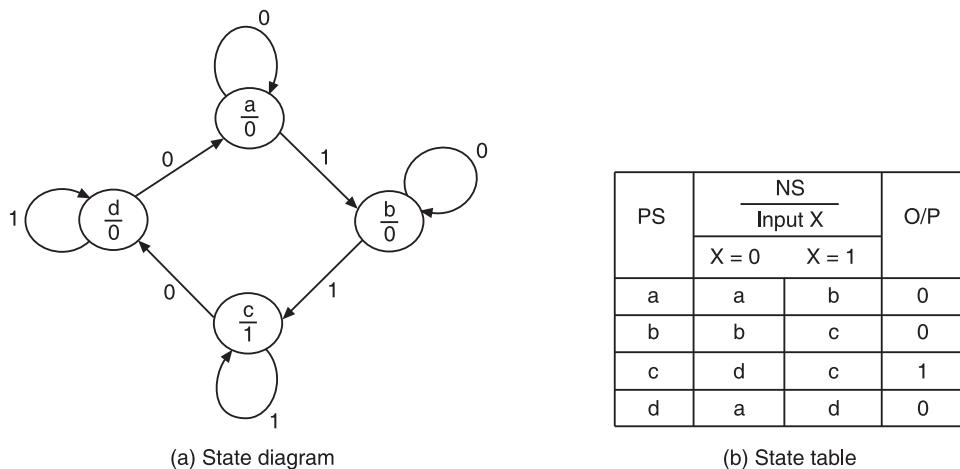


Figure 13.3 Moore circuit.

The state of a memory element is specified by the value of its output, which may assume either a 0 or a 1. The present state of the sequential machine indicates the present outputs of the memory elements used in the machine. The next state of the machine indicates the next outputs of the flip-flops that will be obtained when the present inputs are applied to the machine in the present state.

### 13.1.3 State Reduction

The state reduction technique basically avoids the introduction of redundant states. The reduction in redundant states reduces the number of flip-flops and logic gates, reducing the cost of the final circuit. Two states are said to be equivalent if every possible set of inputs generate exactly the same output and the same next state. When two states are equivalent, one of them can be removed without altering the input-output relationship. Let us illustrate the reduction technique with an example. Consider the sequential circuit whose state diagram is shown in Figure 13.4a. As shown in the figure, the states are represented by letter symbols instead of their binary values because in state reduction technique, the binary designations of states are not important, but input-output sequences are important. State reduction is done in two steps given as follows.

*Step 1.* Determine the state table for the given state diagram. Figure 13.4b shows the state table for the given state diagram.

*Step 2.* Find equivalent states.

As mentioned earlier, in equivalent states, every possible set of inputs generate exactly the same output and the same next state. In the given circuit there are two input combinations  $X = 0$ , and  $X = 1$ . Looking at the state table for two present states that go to the same next state and have the same output for both input combinations, we can easily find that states a and c are equivalent. Also states b and d are equivalent. This is because, both states a and c go to states b and d and have

outputs 0 and 1 for  $X = 0$  and  $X = 1$  respectively. Also states b and e both go to states e and f and have outputs 0 and 1 for  $X = 0$  and  $X = 1$  respectively. Therefore, state c can be removed and replaced by a. Also state e can be removed and replaced by state b. The final reduced state table is shown in Figure 13.5a. The state diagram for the reduced state table consists of only four states and is shown in Figure 13.5b.

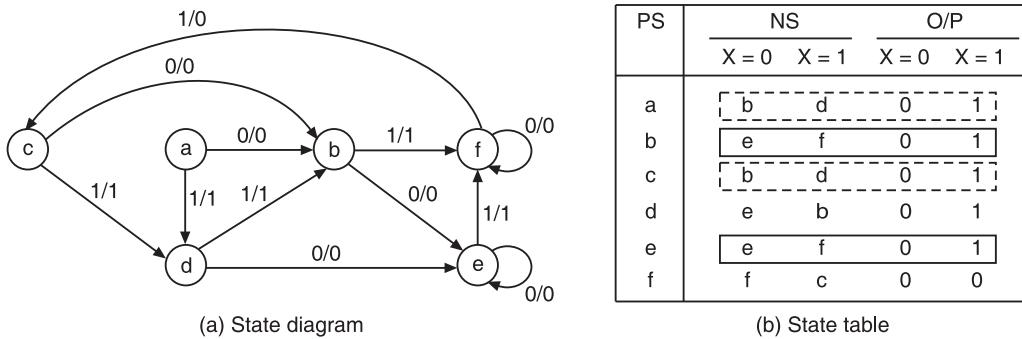


Figure 13.4 Sequential circuit.

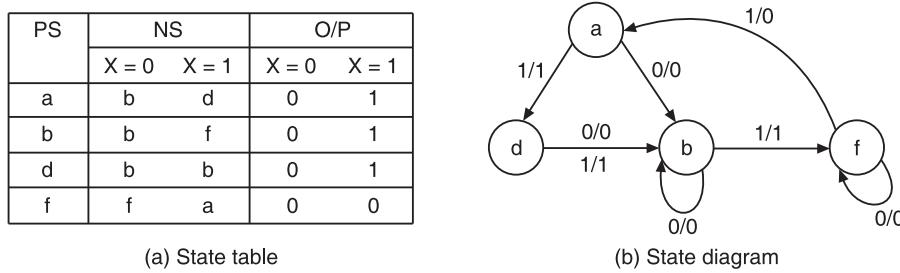


Figure 13.5 Reduced state table and state diagram.

### 13.1.4 State Assignment

The process of assigning binary values to the states of the sequential machine is known as *state assignment*. The binary values are to be assigned to the states in such a way that it is possible to implement flip-flop input functions using minimum logic gates. The output values of the physical devices are referred to as state variables.

**Rules for state assignment:** There are two basic rules for making state assignments.

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

*Rule 2.* States that are the next states of a single state should have assignments which can be grouped into logically adjacent cells in a K-map.

### 13.1.5 Transition and Output Table

The transition and output table can be obtained from the state table by modifying the entries of the state table to correspond to the states of the machine in accordance with the selected state assignment. In this table, the next state and output entries are separated into two sections.

The entries of the next state part of this table define the necessary state transitions of the machine and, thus, specify the next values of the outputs of the FFs used. The next state part of the state table is called the *transition table*. The output part of the table indicates the output of the sequential machine for various input combinations applied to the machine, which is in the present state.

### 13.1.6 Excitation Table

The excitation table of a sequential machine gives information about the excitations or inputs required to be applied to the memory elements in the sequential circuit to bring the sequential machine from the present state to the next state. It also gives information about the outputs of the machine after application of the present inputs. The minimal expressions for the excitations of the FFs and outputs of the machine can be obtained by minimizing the expressions obtained from the excitation table using K-maps. The circuit can be realized using these minimal expressions.

**EXAMPLE 13.1** Obtain reduced state table and reduced state diagram for the sequential machine whose state diagram is shown in Figure 13.6a.

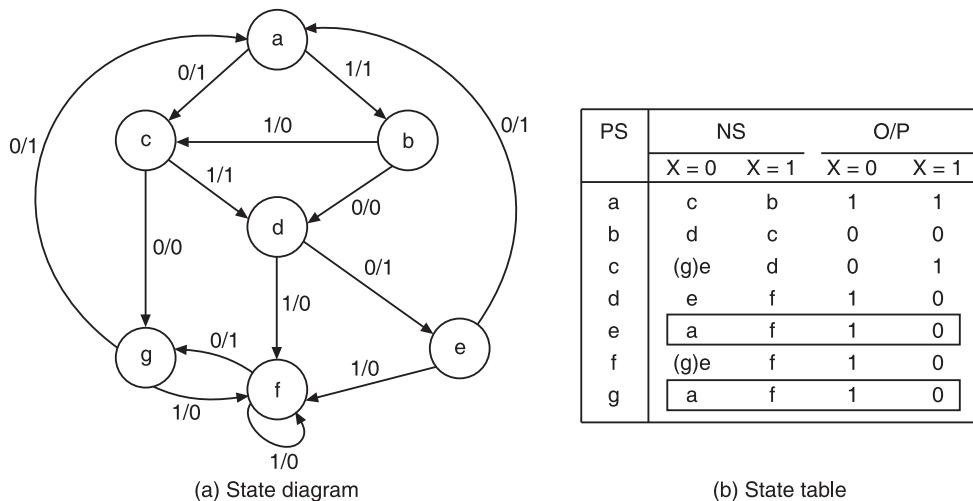
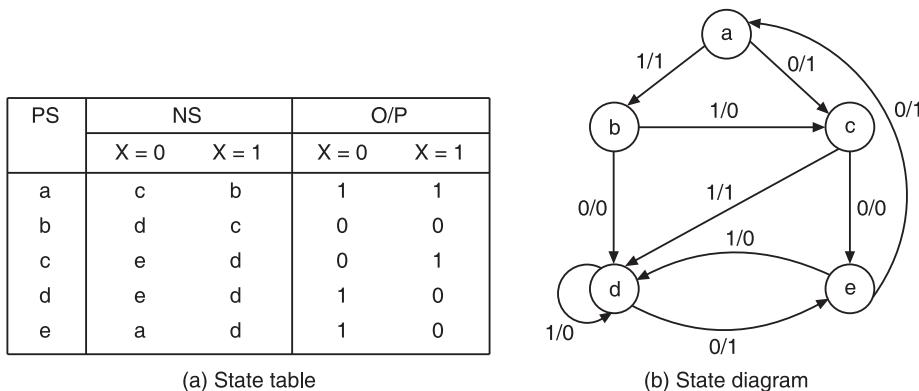


Figure 13.6 Example 13.1.

### Solution

The state table for the given sequential circuit will be as shown in Figure 13.6b.

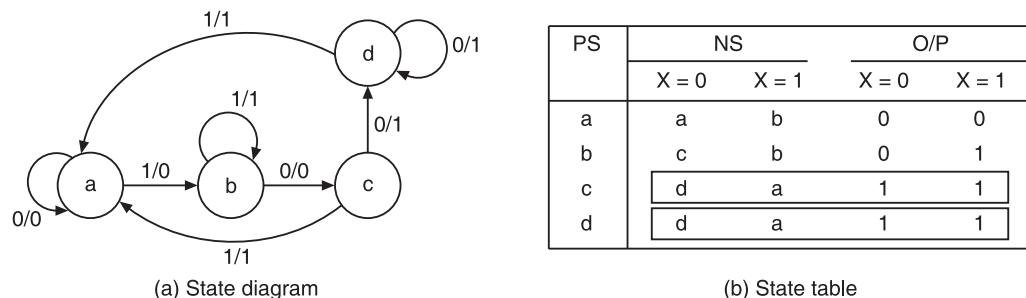
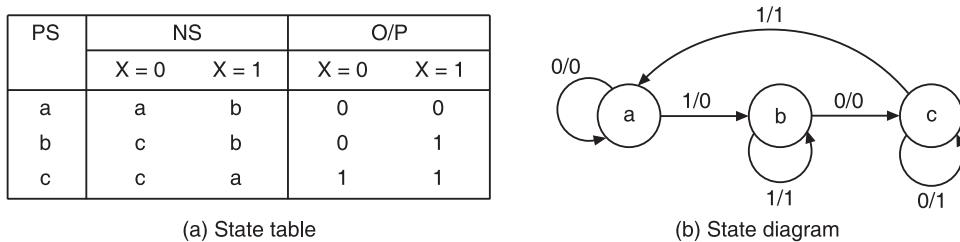
From the state table of Figure 13.6b we observe that states g and e are equivalent because they have the same next state and the same output for each one of the inputs. So one of them becomes redundant and can be removed. Let us remove state g. Replacing g by e in the state table we observe that states d and f are equivalent. So one of them becomes redundant and can be removed. Let us remove f and replace f by d. So we are left with five states a, b, c, d, and e. The reduced state table is shown in Figure 13.7a. The reduced state diagram is shown in Figure 13.7b.

**Figure 13.7** Example 13.1: Reduced state table and state diagram.

**EXAMPLE 13.2** Obtain a reduced state table and reduced state diagram for the sequential machine whose state diagram is shown in Figure 13.8a.

**Solution**

The state table for the given state diagram is shown in Figure 13.8b. From the state table we observe that states c and d are equivalent. So state d can be removed and d is replaced by c at other places. The reduced state table is shown in Figure 13.9a. The reduced state diagram is shown in Figure 13.9b.

**Figure 13.8** Example 13.2.**Figure 13.9** Example 13.2: Reduced state table and state diagram.

## 13.2 MEMORY ELEMENTS

### 13.2.1 D Flip-Flop

The excitation table, the state diagram (Mealy model), and the state table of a D flip-flop are shown in Figures 13.10a, b, and c respectively.

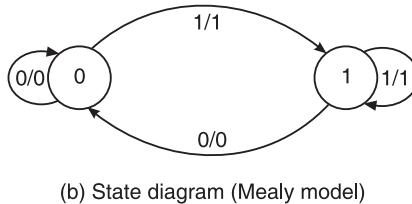
The general state diagram, the K-map for the next state, and the Moore model of the state diagram are shown in Figures 13.10d, e, and f respectively. In the general state diagram,  $S_1$  and  $S_2$  represent the states of the FF and  $I_1$  and  $I_2$  correspond to the input conditions.

The expression for the next state of the flip-flop is

$$Q(t+1) = D(t)$$

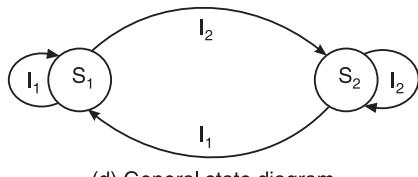
Present state (PS)	Input to FF	Next state (NS)
$Q(t)$	$D(t)$	$Q(t+1)$
0	0	0
0	1	1
1	0	0
1	1	1

(a) Excitation table

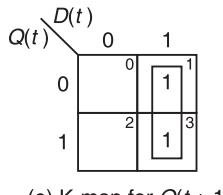
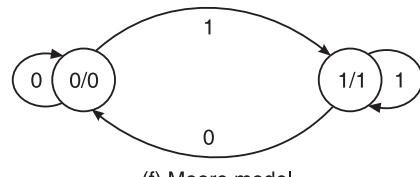


PS	N/S, O/P
	$D = 0$ $D = 1$
0	0, 0    1, 1
1	0, 0    1, 1

(c) State table



(d) General state diagram

(e) K-map for  $Q(t+1)$ 

(f) Moore model

**Figure 13.10** D flip-flop.

### 13.2.2 T Flip-Flop

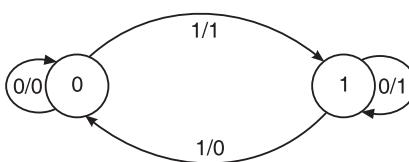
The excitation table, the state diagram(Mealy model) and the state table of the T flip-flop are shown in Figures 13.11a, b and c respectively. The general state diagram, the K-map for the next state, and the Moore model of the state diagram are shown in Figures 13.11d, e, and f respectively.

The expression for the next state of the flip-flop is

$$Q(t+1) = \bar{Q}(t) T(t) + Q(t) \bar{T}(t) = Q(t) \oplus T(t)$$

Present state (PS)	Input to FF	Next state (NS)
$Q(t)$	$T(t)$	$Q(t+1)$
0	0	0
0	1	1
1	0	1
1	1	0

(a) Excitation table



(b) State diagram (Mealy model)

PS	N/S, O/P
	$T = 0$ $T = 1$
0	0, 0    1, 1
1	1, 1    0, 0

(c) State table

**Figure 13.11** T flip-flop (Contd.)...

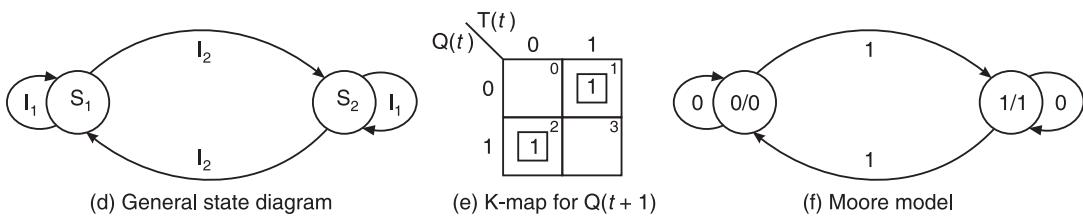


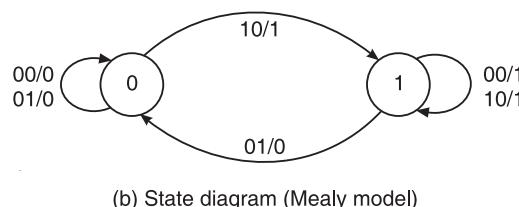
Figure 13.11 T flip-flop.

### 13.2.3 S-R Flip-Flop

The excitation table, the state diagram(Mealy model), and the state table of the S-R flip-flop are shown in Figures 13.12a, b, and c respectively. The general state diagram, the K-map for the next state, and the Moore model of the state diagram are shown in Figures 13.12d, e and f respectively.

Present state (PS)	Inputs to FF		Next state (NS)
	S(t)	R(t)	
Q(t)			Q(t + 1)
0	0	0	0
0	0	1	0
0	1	0	1
1	0	0	1
1	0	1	0
1	1	0	1

(a) Excitation table



PS	NS, O/P		
	S R	S R	S R
	0 0	0 1	1 0
0	0, 0	0, 0	1, 1
1	1, 1	0, 0	1, 1

(c) State table

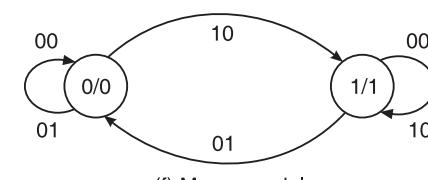
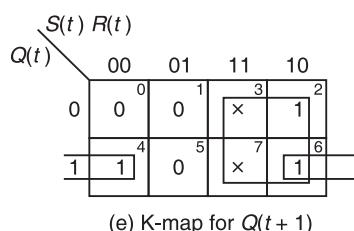
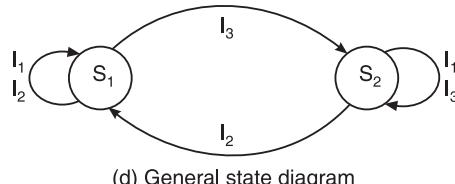


Figure 13.12 S-R flip-flop.

The expression for the next state of the flip-flop is

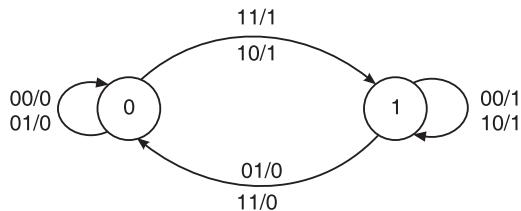
$$Q(t + 1) = Q(t) \bar{R}(t) + S(t)$$

### 13.2.4 J-K Flip-Flop

The excitation table, the state diagram (Mealy model), and the state table are shown in Figures 13.13a, b and c respectively. The general state diagram, the K-map for the next state and the Moore model of the state diagram are shown in Figures 13.13d, e and f respectively.

Present state (PS)	Inputs to FF		Next state (NS)
	J(t)	K(t)	
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

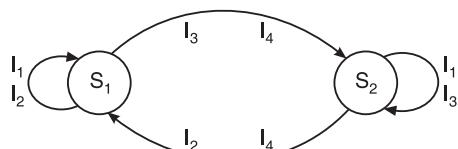
(a) Excitation table



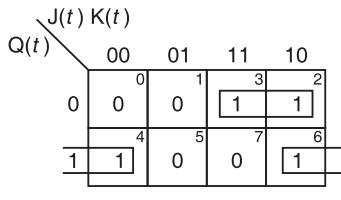
(b) State diagram (Mealy model)

PS	NS, O/P			
	J K	J K	J K	J K
0	0, 0	0, 0	1, 1	1, 1
1	1, 1	0, 0	1, 1	0, 0

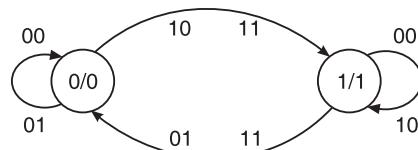
(c) State table



(d) General state diagram



(e) K-map for Q(t + 1)



(f) Moore model

Figure 13.13 J-K flip-flop.

The expression for the next state of the flip-flop is

$$Q(t+1) = J(t) \bar{Q}(t) + Q(t) \bar{K}(t)$$

### 13.3 SYNTHESIS OF SYNCHRONOUS SEQUENTIAL CIRCUITS

The main steps in the general method for designing sequential circuits using various types of memory elements are as follows:

*Step 1. Word statement:* State the purpose of the machine in simple, unambiguous words. It should be realizable with a finite number of memory elements.

*Step 2. State diagram:* Based on the word description of the machine, draw the state diagram which depicts the complete information about it.

*Step 3. State table:* Write the state table which contains all the information of the state diagram in tabular form.

*Step 4. Reduced standard form state table:* Remove the redundant states if any in step 2 and write the reduced standard form state table.

*Step 5. State assignment and transition table:* Assign binary names to the states and write the transition table.

*Step 6. Choose type of flip-flops and form the excitation table:* Based on the entries in the transition table write the excitation table after choosing the type of FFs.

*Step 7. K-maps and minimal expressions:* Based on the contents of the excitation table draw the K-maps and synthesize the logic functions for each of the excitations as functions of input variables and state variables.

*Step 8. Realization:* Draw the circuit to realize the minimal expressions obtained above.

### 13.4 SERIAL BINARY ADDER

*Step 1. Word statement of the problem:* The block diagram of a serial binary adder is shown in Figure 13.14. It is a synchronous circuit with two input terminals designated  $X_1$  and  $X_2$  which carry the two binary numbers to be added and one output terminal  $Z$  which represents the sum. The inputs and outputs consist of fixed-length sequences of 0s and 1s. The addition is performed serially, i.e. the least significant digits of the numbers  $X_1$  and  $X_2$  arrive at the corresponding input terminals at  $t_1$ ; a unit time later the next significant digits arrive at the input terminals, and so on. The time interval between the arrivals of two consecutive input digits is determined by the frequency of the circuit's clock. The delay within the combinational circuit is small with respect to the clock frequency and thus the sum digit arrives at the  $Z$  terminal immediately following the arrival of the corresponding input digits at the input terminals.

The output of the serial adder  $z_i$  at time  $t_i$  is a function of the inputs  $x_1(t_i)$  and  $x_2(t_i)$  at that time  $t_i$  and of a carry which had been generated at  $t_{i-1}$ . The carry which represents the past history of the serial adder may be a 0 or a 1. So, one FF is required to store it and one state variable is required to represent it. Thus, the circuit has two states. If one state indicates that the carry from the previous addition is a 0, the other state indicates that the carry from the previous addition is a 1.

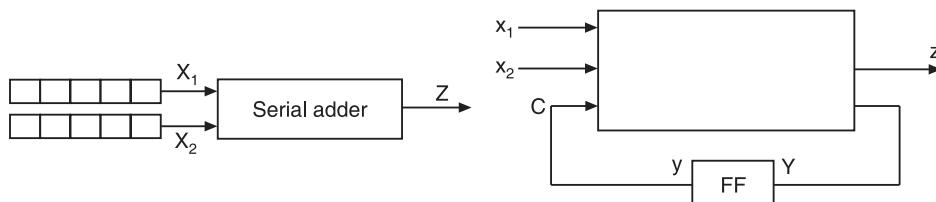


Figure 13.14 Block diagram of the serial binary adder.

*Steps 2 and 3. State diagram and state table:* Let A designate the state of the serial adder at  $t_i$  if a carry 0 was generated at  $t_{i-1}$ , and let B designate the state of the serial adder at  $t_i$  if a carry 1 was generated at  $t_{i-1}$ . The state of the adder at the time when the present inputs are applied is referred to as the *present state* (PS), and the state to which the adder goes as a result of the new carry value is referred to as the *next state* (NS).

The behaviour of a serial adder may be conveniently described by its state diagram and the state table as shown in Figure 13.15. The state diagram shows that if the machine is in state A, i.e. carry from the previous addition is a 0, the inputs  $x_1 = 0, x_2 = 0$  give sum 0 and carry 0. So, the machine remains in state A and outputs a 0. An input  $x_1 = 0$  and  $x_2 = 1$  give sum 1 and carry 0. So, the machine remains in state A and outputs a 1. Inputs  $x_1 = 1, x_2 = 0$  give sum 1 and carry 0. So, the machine remains in state A and outputs a 1, but the inputs  $x_1 = 1, x_2 = 1$  give sum 0 and carry 1. So, the machine goes to state B and outputs a 0.

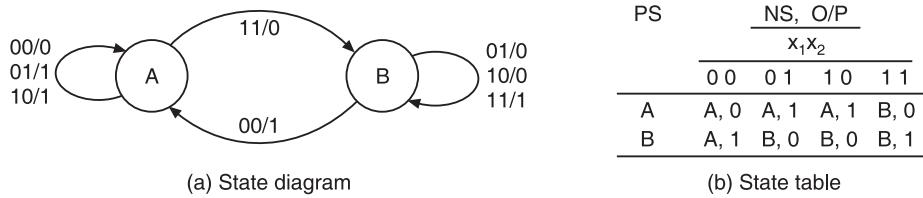


Figure 13.15 Serial adder.

If the machine is in state B, i.e. carry from the previous addition is a 1, inputs  $x_1 = 0, x_2 = 1$  give sum 0 and carry 1. So, the machine remains in state B and outputs a 0. Inputs  $x_1 = 1, x_2 = 0$  give sum 0 and carry 1. So, the machine remains in state B and outputs a 0. Inputs  $x_1 = 1, x_2 = 1$  give sum 1 and carry 1. So, the machine remains in state B and outputs a 1. Inputs  $x_1 = 0, x_2 = 0$  give sum 1 and carry 0. So, the machine goes to state A and outputs a 1. The state table also gives the same information.

*Step 4. Reduced standard form state table:* The machine is already in this form. So no need to do anything.

*Step 5. State assignment and transition and output table:* The states, A = 0 and B = 1 have already been assigned. So, the transition and output table is as shown in Table 13.1.

Table 13.1 Transition and output table

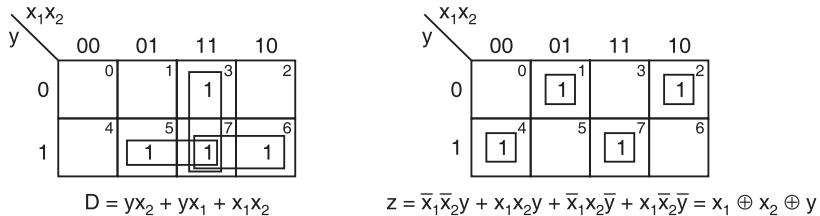
PS	NS				O/P			
	$x_1x_2$				$x_1x_2$			
	00	01	10	11	00	01	10	11
0	0	0	0	1	0	1	1	0
1	0	1	1	1	1	0	0	1

*Step 6. Choose type of flip-flops and form the excitation table:* To write the excitation table, select the memory element. Let us say, we want to use D flip-flop. The excitation table is shown in Table 13.2.

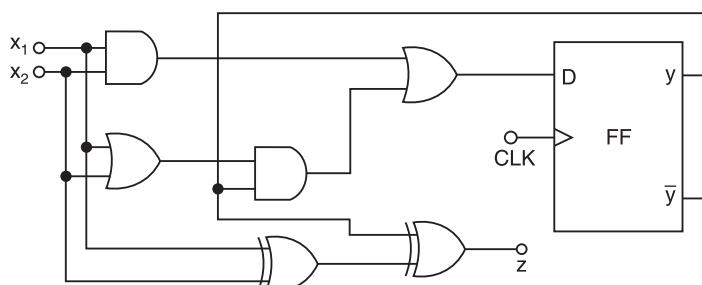
**Table 13.2** Excitation table

PS y	I/P		NS Y	I/P to FF		O/P z
	x <sub>1</sub>	x <sub>2</sub>		D		
0	0	0	0	0		0
0	0	1	0	0		1
0	1	0	0	0		1
0	1	1	1	1		0
1	0	0	0	0		1
1	0	1	1	1		0
1	1	0	1	1		0
1	1	1	1	1		1

*Step 7. K-maps and minimal expressions:* Obtain the minimal expressions for D and z in terms of the state variable y and inputs x<sub>1</sub>, and x<sub>2</sub> by using K-maps as shown in Figure 13.16.

**Figure 13.16** K-maps for the serial adder.

*Step 8. Implementation:* Implement the circuit using those minimal expressions as shown in Figure 13.17.

**Figure 13.17** Logic diagram of the serial binary adder.

### 13.4.1 Moore Type Finite State Machine for a Serial Adder

In a Moore type machine the output depends only on the present state of the machine. It does not depend on the input at all. We see that in a serial adder, the output in each state may be 0 or 1 depending on the values of the inputs. So a Moore type state machine will need more than two states. Figure 13.18a shows a Moore type state diagram for a serial adder. We can define additional

states by splitting both the states  $a$  and  $b$  into two states each. Instead of  $a$  we will use  $a_0$  and  $a_1$  to denote the fact that the carry is 0 and the sum is either 0 or 1 respectively. When the state is split, we have to direct transition associated with output 0 to state 0 and transition associated with output 1 to state 1. So state  $a$  with output 1 is directed to state  $a_1$  and state  $a$  without 0 is directed to state  $a_0$ . Figure 13.18b shows the state table for the Moore type serial adder. Once, the Moore type state diagram and state table are drawn the Moore circuit can be designed following the normal procedure.

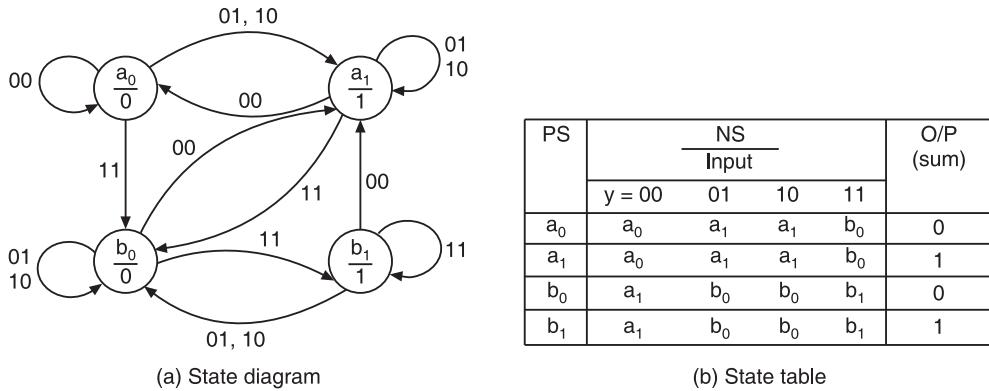


Figure 13.18 Moore type serial adder.

## 13.5 THE SEQUENCE DETECTOR

### 13.5.1 Mealy Type Model

*Step 1. Word statement of the problem:* A sequence detector is a sequential machine which produces an output 1 every time the desired sequence is detected and an output 0 at all other times.

Suppose we want to design a sequence detector to detect the sequence 1010 and say that overlapping is permitted, i.e. for example, if the input sequence is 01101010 the corresponding output sequence is 00000101. We can start the synthesis procedure by constructing the state diagram of the machine.

*Steps 2 and 3. State diagram and state table:* The state diagram and the state table of the sequence detector are shown in Figure 13.19. At time  $t_1$ , the machine is assumed to be in the initial state designated arbitrarily as A. While in this state, the machine can receive first bit input, either a 0 or a 1. If the input bit is a 0, the machine does not start the detection process because the first bit in the desired sequence is a 1. So, it remains in the same state and outputs a 0. Hence, an arc labelled 0/0 starting and terminating at A is drawn. If the input bit is a 1, the detection process starts. So, the machine goes to state B and outputs a 0. Hence, an arc labelled 1/0 starting at A and terminating at B is drawn. While in state B, the machine may receive a 0 or a 1 bit. If the bit is a 0, the machine goes to the next state, say state C, because the previous two bits are 10 which are a part of the valid sequence, and outputs a 0. So, draw an arc labelled 0/0 from B to C. If the bit is a 1, the two bits become 11 and this is not a part of the

valid sequence. Since overlapping is permitted, the second 1 may be used to start the sequence, so, the machine remains in state B only and outputs a 0. So, an arc labelled 1/0 starting and terminating at B is drawn. While in state C, the machine may receive a 0 or a 1 bit. If it receives a 0, the last three bits received will be 100 and this is not a part of the valid sequence and also none of the last two bits can be used to start the new sequence. So, the machine goes to state A to restart the detection process and outputs a 0. Hence, an arc labelled 0/0 starting at C and terminating at A is drawn. If it receives a 1 bit, the last three bits will be 101 which are a part of the valid sequence. So, the machine goes to the next state, say state D, and outputs a 0. So, an arc labelled 1/0 is drawn from C to D. While in state D, the machine may receive a 0 or a 1. If it receives a 0, the last four bits become 1010 which is a valid sequence and the machine outputs a 1. Since overlapping is permitted, the machine can utilize the last two bits 10 to get another 1010 sequence. So, the machine goes to state C (if overlapping is not permitted or the machine has to restart after outputting a 1, the machine goes to state A). So, an arc labelled 0/1 is drawn from D to C. If it receives a 1 bit, the last four bits received will be 1011 which is not a valid sequence. So, the machine outputs a 0. Since the fourth bit 1 can become the starting bit for the valid sequence, the machine goes to state B. So, an arc labelled 1/0 is drawn from D to B.

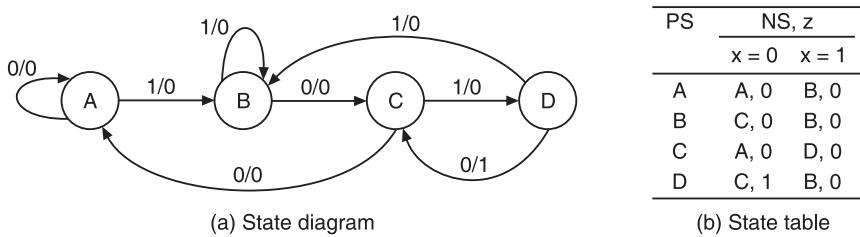


Figure 13.19 Sequence (1010) detector.

*Step 4. Reduced standard form state table:* The machine is already in this form. So no need to do anything.

*Step 5. State assignment and transition and output table:* There are four states; therefore, two state variables are required. Two state variables can have a maximum of four states. So, all states are utilized and thus there are no invalid states. Hence, there are no don't cares. Assign the states arbitrarily. Let  $A \rightarrow 00$ ,  $B \rightarrow 01$ ,  $C \rightarrow 10$ , and  $D \rightarrow 11$  be the state assignment. With this assignment draw the transition and output Table 13.3.

Table 13.3 Transition and output table

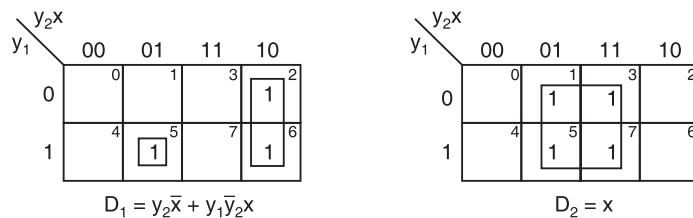
PS ( $y_1y_2$ )	NS ( $Y_1Y_2$ )		O/P (z)	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
$A \rightarrow 0\ 0$	0 0	0 1	0	0
$B \rightarrow 0\ 1$	1 0	0 1	0	0
$C \rightarrow 1\ 0$	0 0	1 1	0	0
$D \rightarrow 1\ 1$	1 0	0 1	1	0

*Step 6. Choose type of Flip-flops and form the excitation table:* Select the D flip-flops as memory elements and draw the excitation Table 13.4.

**Table 13.4** Excitation table

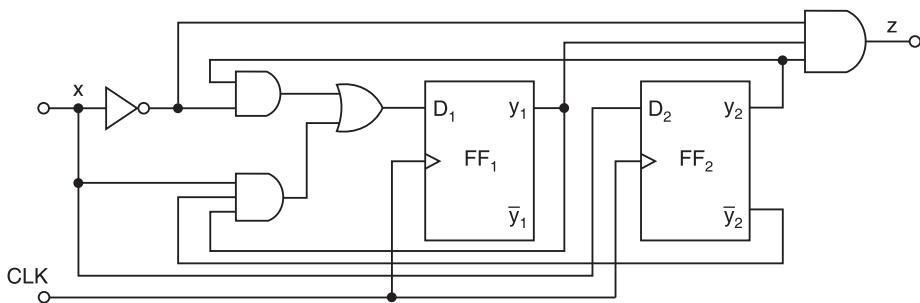
<b>PS</b>		<b>I/P</b>	<b>NS</b>		<b>I/P to FFs</b>		<b>O/P</b>
<b>y<sub>1</sub></b>	<b>y<sub>2</sub></b>	<b>x</b>	<b>Y<sub>1</sub></b>	<b>Y<sub>2</sub></b>	<b>D<sub>1</sub></b>	<b>D<sub>2</sub></b>	<b>z</b>
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	1	0
1	0	0	0	0	0	0	0
1	0	1	1	1	1	1	0
1	1	0	1	0	1	0	1
1	1	1	0	1	0	1	0

*Step 7. K-maps and minimal expressions:* Based on the contents of the excitation table, draw the K-maps and simplify them to obtain the minimal expressions for  $D_1$  and  $D_2$  in terms of  $y_1$ ,  $y_2$ , and  $x$  as shown in Figure 13.20. The expression for  $z$  ( $z = y_1 y_2$ ) can be obtained directly from Table 13.3.



**Figure 13.20** K-maps for  $D_1$  and  $D_2$  of the sequence (1010) detector.

*Step 8. Implementation:* The logic diagram based on these minimal expressions is shown in Figure 13.21.



**Figure 13.21** Logic diagram of the sequence (1010) detector using D flip-flops.

### 13.5.2 Moore Type Circuit

For the design of Moore type of circuit, the same steps are to be followed. The state diagram and the state table of a Moore type sequence detector to detect the sequence 1010 are shown in Figure 13.22. In the Moore type state diagram the outputs are written inside the circle below the state name. The state diagram is drawn in the normal way. The machine will be in state D if the last three bits are 101. If the next bit is a 0, the last four bits will be 1010 which is a valid sequence. So the machine outputs a 1, but to utilize overlapping it cannot go to state C because the output at C is 0. Instead it goes to a new state E where output is equal to 1. While at E, if the next bit is a 0 the last five bits become 10100. So the machine goes to state A to restart the detection. If the next bit is a 1, the last five bits become 10101. So to utilize the last three bits, i.e. 101 it goes to state D. Once, the Moore type state diagram and state table are drawn, the Moore circuit can be designed following the normal procedure.

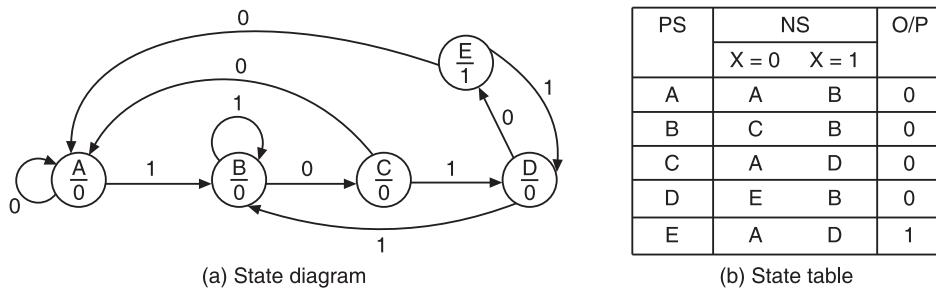


Figure 13.22 Moore circuit.

**EXAMPLE 13.3** Draw the state diagram and the state table for a Moore type sequence detector to detect the sequence 110.

#### Solution

The Moore type state diagram and state table of sequence detector to detect the sequence 110 are shown in Figure 13.23. The state diagram is drawn in the normal way. The machine is in state C when the last two bits received are 11. If the next bit is a 0, the last three bits become 110 which is a valid sequence, hence it outputs a 1, but the machine cannot go to state A to restart the detection process because state A outputs a 0. So the machine goes to a new state D and outputs a 1. While at D if the next bit received is a 0, the last four bits will be 1100. So the machine goes to state A to restart the process. If the bit received is a 1, the last four bits will be 1101. So the machine goes to state B to utilize the last bit.

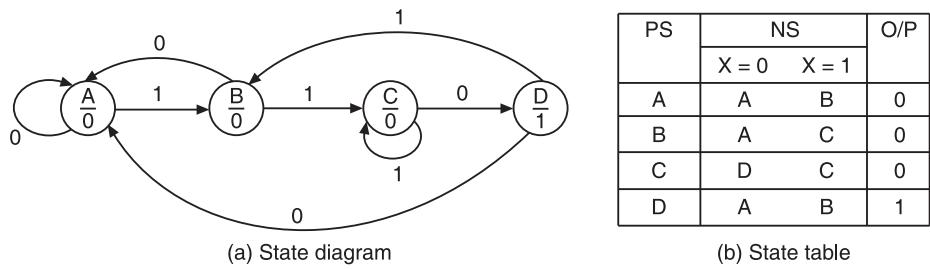


Figure 13.23 Example 13.3.

## 13.6 PARITY-BIT GENERATOR

### 13.6.1 Odd Parity-bit Generator

*Step 1. Word statement of the problem:* A serial parity-bit generator is a two-terminal circuit which receives coded messages and adds a parity bit to every  $m$  bits of the message, so that the resulting outcome is an error-detecting coded message. The inputs are assumed to arrive in strings of three symbols ( $m = 3$ ) and the strings are spaced apart by single time units (i.e. the fourth place is a blank). The parity bits are inserted in the appropriate spaces so that the resulting outcome is a continuous string of symbols without spaces. For even parity, a parity bit 1 is inserted, if and only if the number of 1s in the preceding string of three symbols is odd. For odd parity, a parity bit 1 is inserted, if and only if the number of 1s in the preceding string of three symbols is even.

*Steps 2 and 3. State diagram and state table:* The state diagram and the state table of an odd parity-bit generator are shown in Figure 13.24. States B, D and F correspond to even number of 1s out of 1, 2 and 3 incoming inputs, respectively. Similarly, states C, E and G correspond to odd number of 1s out of 1, 2 and 3 incoming inputs, respectively. From either state F or state G, the machine goes to state A regardless of the input. In fact, the fourth input is a blank.

Since the state diagram contains seven states, three state variables are needed for an assignment. But since three state variables can have a total of eight states, one of the states will not be assigned and its entries in the corresponding state table may be considered as don't cares.

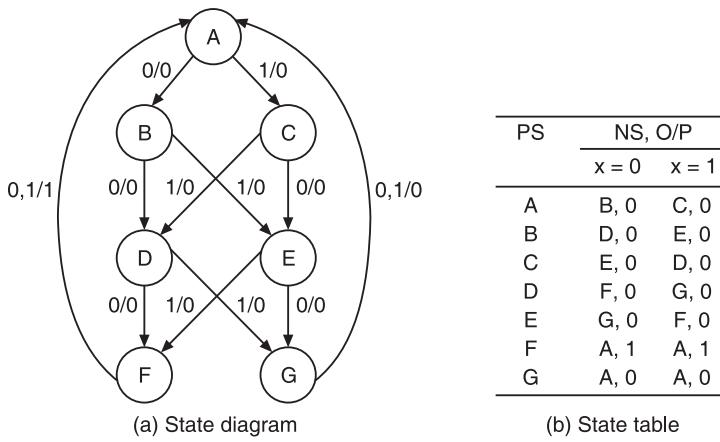


Figure 13.24 3-bit odd-parity generator.

*Step 4. Reduced standard form state table:* The machine is already in this form. So no need to do anything.

*Step 5. State assignment and transition and output table:* The state assignment is not unique. Many possible assignments are there. One possible assignment is A  $\rightarrow$  000, B  $\rightarrow$  010, C  $\rightarrow$  011, D  $\rightarrow$  110, E  $\rightarrow$  111, F  $\rightarrow$  100 and G  $\rightarrow$  101. With this assignment the transition and output table is given in Table 13.5.

*Step 6. Choose type of flip-flops and form the excitation table:* Select the memory elements. Suppose J-K flip-flops are selected. For implementing the parity-bit generator using J-K flip-flops, draw the excitation table as shown in Table 13.6.

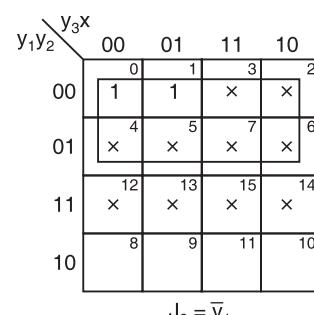
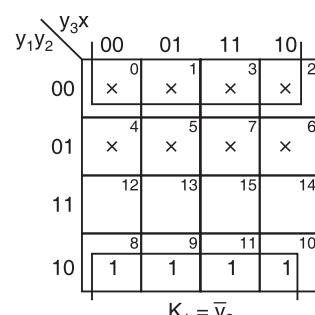
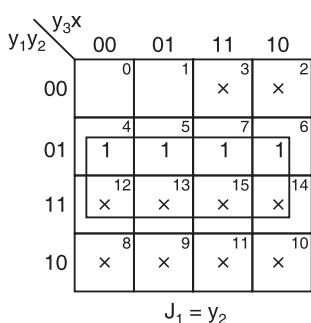
**Step 7. K-maps and minimal expressions:** The minimal expressions for excitations  $J_1, K_1, J_2, K_2, J_3$  and  $K_3$  of flip-flops and the output of the odd-parity generator  $z$  in terms of the present state variables  $y_1, y_2, y_3$ , and the input  $x$  can be obtained using K-maps as shown in Figure 13.25.

**Table 13.5** Transition and output

PS			NS ( $Y_1 Y_2 Y_3$ )		O/P (z)	
$y_1$	$y_2$	$y_3$	$x = 0$	$x = 1$	$x = 0$	$x = 1$
0	0	0	0 1 0	0 1 1	0	0
0	1	0	1 1 0	1 1 1	0	0
0	1	1	1 1 1	1 1 0	0	0
1	1	0	1 0 0	1 0 1	0	0
1	1	1	1 0 1	1 0 0	0	0
1	0	0	0 0 0	0 0 0	1	1
1	0	1	0 0 0	0 0 0	0	0

**Table 13.6** Excitation table

PS			I/P	NS			Present excitations required				O/P		
$y_1$	$y_2$	$y_3$	$x$	$Y_1$	$Y_2$	$Y_3$	$J_1$	$K_1$	$J_2$	$K_2$	$J_3$	$K_3$	$z$
0	0	0	0	0	1	0	0	$\times$	1	$\times$	0	$\times$	0
0	0	0	1	0	1	1	0	$\times$	1	$\times$	1	$\times$	0
0	1	0	0	1	1	0	1	$\times$	$\times$	0	0	$\times$	0
0	1	0	1	1	1	1	1	$\times$	$\times$	0	1	$\times$	0
0	1	1	0	1	1	1	1	$\times$	$\times$	0	$\times$	0	0
0	1	1	1	1	1	0	1	$\times$	$\times$	0	$\times$	1	0
1	1	0	0	1	0	0	$\times$	0	$\times$	1	0	$\times$	0
1	1	0	1	1	0	1	$\times$	0	$\times$	1	1	$\times$	0
1	1	1	0	1	0	1	$\times$	0	$\times$	1	$\times$	0	0
1	1	1	1	1	0	0	$\times$	0	$\times$	1	$\times$	1	0
1	0	0	0	0	0	0	$\times$	1	0	$\times$	0	$\times$	1
1	0	0	1	0	0	0	$\times$	1	0	$\times$	0	$\times$	1
1	0	1	0	0	0	0	$\times$	1	0	$\times$	$\times$	1	0
1	0	1	1	0	0	0	$\times$	1	0	$\times$	$\times$	1	0



Also from K-maps for  $K_2, J_3, K_3$  and  $z$

$$K_2 = y_1; \quad J_3 = y_2 x + \bar{y}_1 x; \quad K_3 = x + \bar{y}_2; \quad z = y_1 \bar{y}_2 \bar{y}_3$$

**Figure 13.25** K-maps for excitations of 3-bit odd-parity generator using J-K flip-flops.

*Step 8. Implementation:* The logic diagram based on those expressions is shown in Figure 13.26.

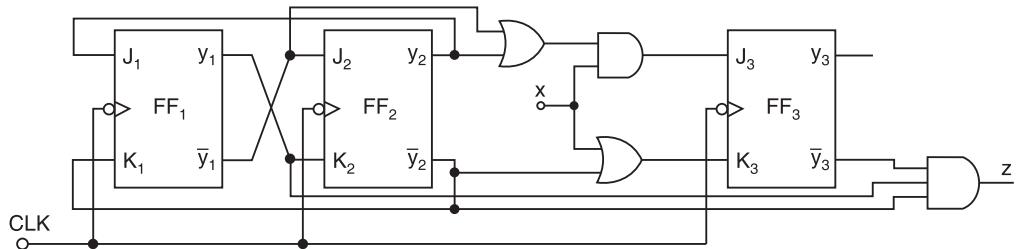


Figure 13.26 Logic diagram of a 3-bit odd-parity generator using J-K flip-flops.

## 13.7 COUNTERS

### 13.7.1 Design of a 3-bit Gray Code Counter

*Step 1. Word statement of the problem:* The counter is to be designed with one input terminal (which receives pulse signals) and one output terminal. It should be capable of counting in the Gray system up to 7 and producing an output pulse for every 8 input pulses. After the count 7 is reached, the next pulse will reset the counter to its initial state, i.e. to a count of zero.

*Steps 2 and 3. State diagram and state table:* The state diagram, and the state table of the 3-bit Gray code counter are shown in Figure 13.27.

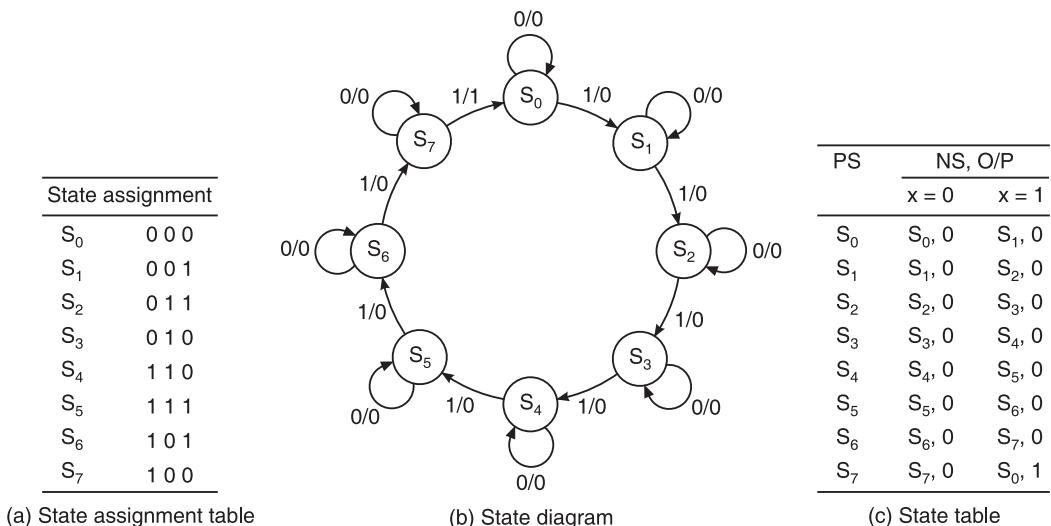


Figure 13.27 A 3-bit Gray code counter.

*Step 4. Reduced standard form state table:* The machine is already in this form. So no need to do anything.

*Step 5. State assignment and transition and output table:* There are eight states for a 3-bit counter. So, three state variables are required which can give a maximum of eight possible states. So, no

invalid states exist. The state assignment cannot be arbitrary, since the counter has to change the states in a definite manner. Hence, the state assignment is

$$S_0 \rightarrow 000, S_1 \rightarrow 001, S_2 \rightarrow 011, S_3 \rightarrow 010, S_4 \rightarrow 110, S_5 \rightarrow 111, S_6 \rightarrow 101, S_7 \rightarrow 100$$

The transition and output table for the counter is shown in Table 13.7.

**Table 13.7** Transition and output table

PS	NS		O/P	
	x = 0	x = 1	x = 0	x = 1
0 0 0	0 0 0	0 0 1	0	0
0 0 1	0 0 1	0 1 1	0	0
0 1 1	0 1 1	0 1 0	0	0
0 1 0	0 1 0	1 1 0	0	0
1 1 0	1 1 0	1 1 1	0	0
1 1 1	1 1 1	1 0 1	0	0
1 0 1	1 0 1	1 0 0	0	0
1 0 0	1 0 0	0 0 0	0	1

*Step 6. Choose type of flip-flops and form the excitation table:* Select T type flip-flops. The Excitation table is as shown in Table 13.8.

**Table 13.8** Excitation table

PS	I/P	NS			Inputs to FFs			O/P
		Y <sub>3</sub>	Y <sub>2</sub>	Y <sub>1</sub>	x	T <sub>3</sub>	T <sub>2</sub>	
0 0 0	0	0	0	0	0	0	0	0
0 0 0	1	0	0	1	0	0	1	0
0 0 1	0	0	0	1	0	0	0	0
0 0 1	1	0	1	1	0	1	0	0
0 1 1	0	0	1	1	0	0	0	0
0 1 1	1	0	1	0	0	0	1	0
0 1 0	0	0	1	0	0	0	0	0
0 1 0	1	1	1	0	1	0	0	0
1 1 0	0	1	1	0	0	0	0	0
1 1 0	1	1	1	1	0	0	1	0
1 1 1	0	1	1	1	0	0	0	0
1 1 1	1	1	0	1	0	1	0	0
1 0 1	0	1	0	1	0	0	0	0
1 0 1	1	0	0	0	0	0	1	0
1 0 0	0	1	0	0	0	0	0	0
1 0 0	1	0	0	0	1	0	0	1

*Step 7. K-maps and minimal expressions:* The minimal expressions for excitation functions to T flip-flops, T<sub>1</sub>, T<sub>2</sub> and T<sub>3</sub> in terms of the present state variables y<sub>1</sub>, y<sub>2</sub>, y<sub>3</sub>, and the input x can be obtained using K-maps as shown in Figure 13.28. From the excitation table, the expression for output z is

$$z = y_3 \bar{y}_2 \bar{y}_1 x$$

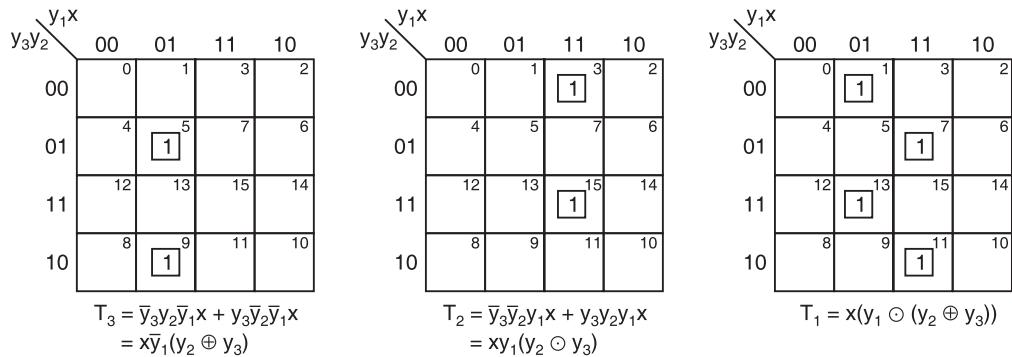


Figure 13.28 K-maps for excitations of 3-bit Gray code counter using T flip-flops.

*Step 8. Implementation:* The logic diagram based on these expressions is shown in Figure 13.29.

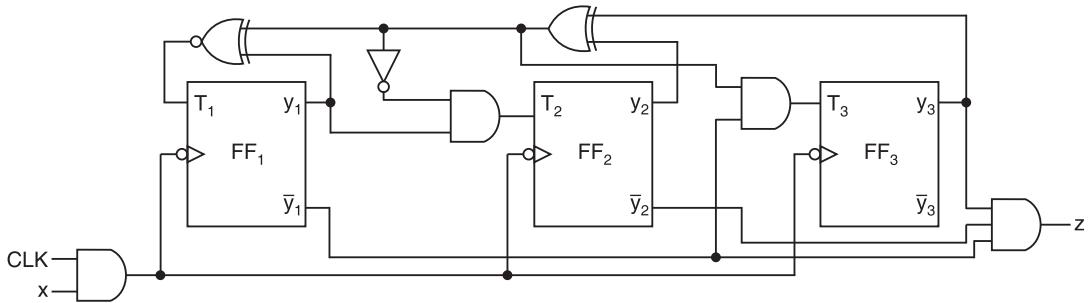
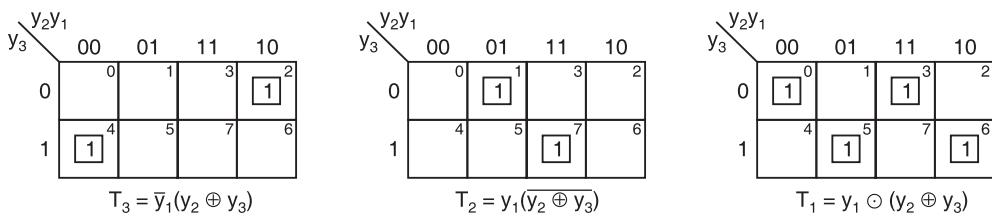


Figure 13.29 Logic diagram of a 3-bit Gray code counter.

The input signal  $x$  can be ANDed with the clock signal, and the output of the AND gate given as clock signal to each FF. The counter thus changes state only when  $x = 1$ , because only at that time the AND gate transmits the external clock to the clock terminal of FFs. When  $x = 0$ , the AND gate is disabled, the FFs receive no clock and the counter remains in the previous state. The excitation table is shown in Table 13.9. The minimal expressions obtained from K-maps are shown in Figure 13.30. The expressions in both the cases are the same, but the design is simpler in the second case.

Table 13.9 Excitation table

PS			NS			O/P	Excitation		
$y_3$	$y_2$	$y_1$	$x = 1$ , clock present				$T_3$	$T_2$	$T_1$
0	0	0	0	0	1	0	0	0	1
0	0	1	0	1	1	0	0	1	0
0	1	1	0	1	0	0	0	0	1
0	1	0	1	1	0	0	1	0	0
1	1	0	1	1	1	0	0	0	1
1	1	1	1	0	1	0	0	1	0
1	0	1	1	0	0	0	0	0	1
1	0	0	0	0	0	1	1	0	0



**Figure 13.30** K-maps for excitations when  $x$  is ANDed with clock.

**EXAMPLE 13.4** A long sequence of pulses enters a 2-input 2-output synchronous sequential circuit which is required to produce an output  $z = 1$ , whenever the sequence 1111 occurs. Overlapping sequences are accepted. For example, if the input is 01011111, the required output is 00000011. Design the circuit.

### Solution

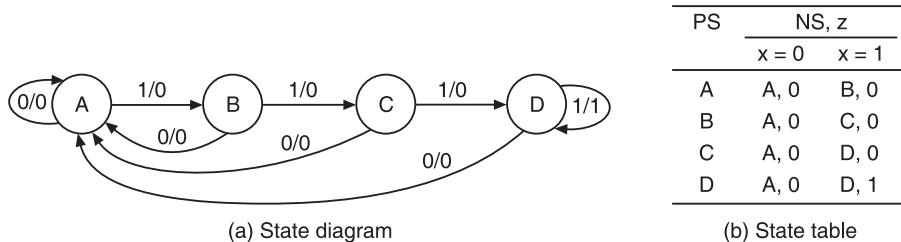
*Step 1. Word statement of the problem:* The block diagram of the 2-input, 2-output synchronous sequential circuit (sequence detector) with one input terminal and one output terminal is shown in Figure 13.31. The sequence detector has to detect and produce an output 1 whenever the sequence 1111 occurs. Overlapping sequences are accepted.



**Figure 13.31** Example 13.4: Block diagram of a 2-input 2-output synchronous sequential machine.

*Steps 2 and 3. State diagram and state table:* Let the machine be initially in state A. While at A the first bit received may be a 0 or a 1. If it is a 0, it is invalid, so it will not start the detection process and so it will remain at A itself and outputs a 0. If the first bit is a 1, it is valid and so the detection process starts and the machine goes to next state B and outputs a 0. While at B the next bit received may be a 0 or a 1. If the second bit is a 0, the first two bits become 10 which is not a part of the valid sequence and so the detection has to start afresh. So machine goes back to A the starting state and outputs a 0. If the second bit is a 1, the first two bits become 11 which is a part of the valid sequence. So machine goes to next state C and outputs a 0. While at C, the machine may receive the next bit as a 0 or a 1. If the third bit is a 0, the sequence becomes 110 which is not a part of the valid sequence. So machine will output a 0 and goes to the initial state A. If the third bit is a 1, the sequence becomes 111 which is a part of the valid sequence. So the machine goes to the next state D and outputs a 0. While at D, the machine may receive the next bit as a 0 or a 1. If the fourth bit is a 0, the sequence becomes 1110, which is not valid and so the machine outputs a 0 and goes to state A. If the fourth bit is a 1, the four bits become 1111 which is a valid sequence. The machine outputs a 1 and remains at D itself because overlapping is permitted. It can utilize the second, third and fourth bits, i.e. 111 and continue the detection process. So if the fifth bit is a 0, the last four bits become 1110, so it will output a 0 and goes to state A. If the fifth bit is a 1, the last four bits become 1111, so it will output a 1 and remain at D itself and so on.

Based on the above description of the working of the machine, the state diagram and the state table indicating the transition of states are shown in Figure 13.32.



**Figure 13.32** Example 13.4: Sequence (1111) detector.

*Step 4. Reduced standard form state table:* The machine is already in this form. So no need to do anything.

**Step 5. State assignment and transition and output table:** The state assignment is arbitrary. There are four states. So, two state variables are needed, which can give a maximum of four states. Therefore, there are no invalid states. Let the states be assigned as A → 00, B → 01, C → 10, and D → 11. With this state assignment, the transition and output table is as shown in Table 13.10.

**Table 13.10** Example 13.4: Transition and output table

PS		NS ( $Y_1 Y_2$ )		Output, z	
$y_1$	$y_2$	x = 0	x = 1	x = 0	x = 1
0	0	0 0	0 1	0	0
0	1	0 0	1 0	0	0
1	0	0 0	1 1	0	0
1	1	0 0	1 1	0	1

*Step 6. Choose type of flip-flops and form the excitation table:* Let us say D flip-flops are used as memory elements. With D flip-flops as memory elements the excitation table is as shown in Table 13.11.

**Table 13.11** Example 13.4: Excitation table

**Step 7. K-maps and minimal expressions:** The minimal expressions for excitations of FFs of the sequential circuit in terms of the present state variables  $y_1, y_2$  and input  $x$  are obtained using K-maps as shown in Figure 13.33. From the excitation table the output  $z$  is given by  $z = y_1 y_2 x$ .

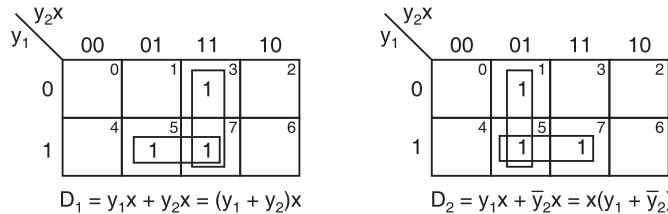


Figure 13.33 Example 13.4: K-maps for the sequence (1111) detector using D flip-flops.

**Step 8. Implementation:** The logic diagram of the sequence detector based on those minimal expressions is shown in Figure 13.34.

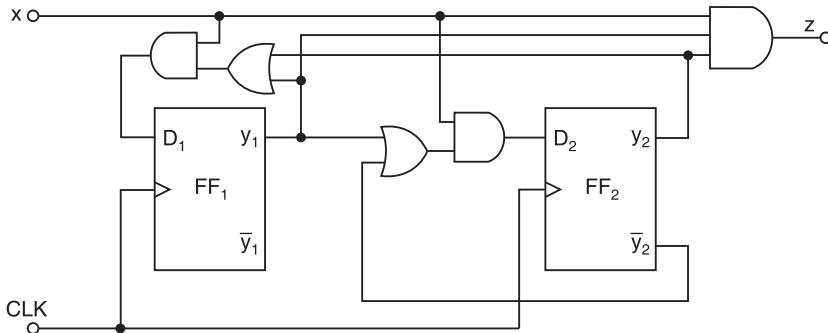


Figure 13.34 Example 13.4: Logic diagram of the sequence (1111) detector using D flip-flops.

**EXAMPLE 13.5** A synchronous sequential machine has a single control input  $x$  and the clock, and two outputs A and B. On consecutive rising edges of the clock, the code on A and B changes from 00 to 01 to 10 to 11 and repeats itself if  $x = 1$ ; if at any time,  $x = 0$ , it holds to the present state. Draw the state diagram and implement the circuit using T flip-flops.

### Solution

**Step 1. Word statement of the problem:** The block diagram of the sequential machine is shown in Figure 13.35. The given machine is nothing but a sequential circuit with two flip-flops. Let A and B be the outputs of the flip-flops. So, two state variables are required which can have a maximum of four states. There are no invalid states present.

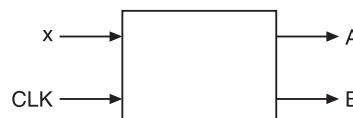


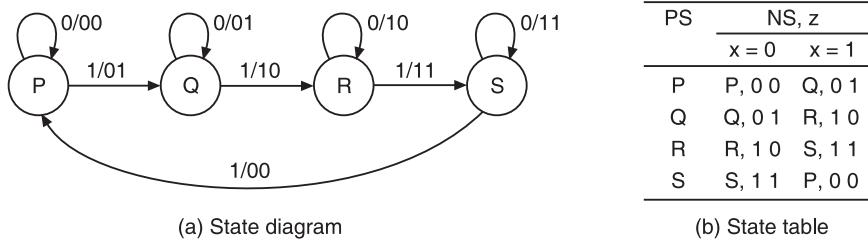
Figure 13.35 Example 13.5: Sequential machine.

*Steps 2 and 3. State diagram and state table:* Let the machine be initially in state P.

The state diagram and the state table of the sequential machine with state assignment of

$$P \rightarrow 00, Q \rightarrow 01, R \rightarrow 10, \text{ and } S \rightarrow 11$$

are shown in Figure 13.36.



(a) State diagram

(b) State table

**Figure 13.36** Example 13.5: Sequential machine.

*Step 4. Reduced standard form state table:* The machine is already in this form. So no need to do anything.

*Step 5. State assignment and transition and output table:* The transition and output table with this assignment is shown in Table 13.12.

**Table 13.12** Example 13.5: Transition and output table

PS		NS ( $Y_1 Y_2$ )		O/P (z)	
		x = 0	x = 1	x = 0	x = 1
$y_1$	$y_2$	0 0	0 1	0 0	0 1
0	0	0 0	0 1	0 0	0 1
0	1	0 1	1 0	0 1	1 0
1	0	1 0	1 1	1 0	1 1
1	1	1 1	0 0	1 1	0 0

*Step 6. Choose type of flip-flops and form the excitation table:* Let us select T flip-flops as memory elements. With T FFs, the excitation table is shown in Table 13.13.

**Table 13.13** Example 13.5: Excitation table

PS		I/P	NS		Required excitations		
$y_1$	$y_2$		x	$Y_1$	$Y_2$	$T_1$	$T_2$
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	1
0	1	0	0	0	1	0	0
0	1	1	1	0	0	1	1
1	0	0	1	0	0	0	0
1	0	1	1	1	0	0	1
1	1	0	1	1	0	0	0
1	1	1	0	0	0	1	1

**Step 7. K-maps and minimal expressions:** The K-maps, their minimization, and the minimal expressions for excitations obtained from them are shown in Figure 13.37a. The outputs of the machine are the same as the outputs of the flip-flops.

**Step 8. Implementation:** The logic diagram based on those minimal expressions is shown in Figure 13.37b.

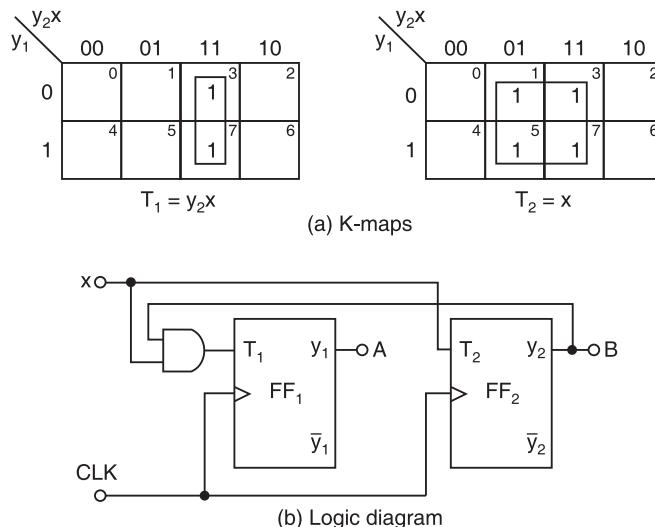


Figure 13.37 Example 13.5: K-maps and logic diagram.

**EXAMPLE 13.6** Design a circuit that will function as prescribed by the state diagram shown in Figure 13.38. Use S-R flip-flops for implementation.

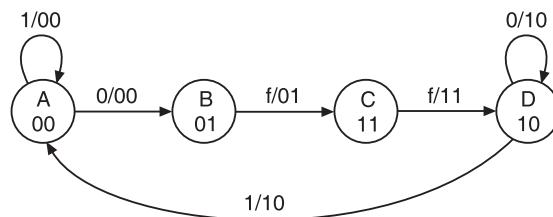


Figure 13.38 Example 13.6: State diagram.

### Solution

The state diagram is already given and is as shown in Figure 13.38. The states are already assigned. Based on that state assignment, write the transition and output table (Table 13.14). Select the memory elements as S-R flip-flops and write the excitation table (Table 13.15). Two state variables are required. Two state variables can have a maximum of four states. So there are no invalid states.

Draw the K-maps, simplify them and obtain the minimal expressions for  $S_1$ ,  $R_1$ ,  $S_2$ ,  $R_2$  and  $z_1$ ,  $z_2$  in terms of  $y_1$ ,  $y_2$  and  $x$  as shown in Figure 13.39. From the excitation table, we can see that  $z_1 = y_1$  and  $z_2 = y_2$  (because the entries are the same).

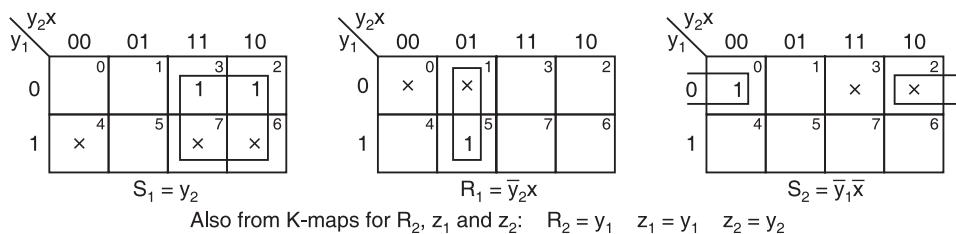
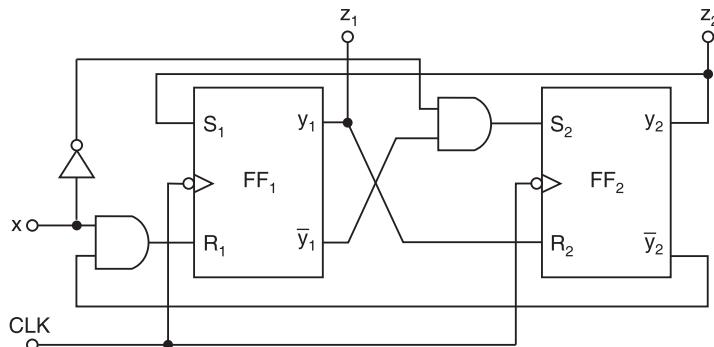
The logic circuit using S-R flip-flops based on those minimal expressions is shown in Figure 13.40.

**Table 13.14** Example 13.6: Transition and output table

PS		NS ( $Y_1 Y_2$ )		O/P ( $z_1 z_2$ )	
		x = 0	x = 1	x = 0	x = 1
$y_1$	$y_2$				
0	0	0 1	0 0	0 0	0 0
0	1	1 1	1 1	0 1	0 1
1	1	1 0	1 0	1 1	1 1
1	0	1 0	0 0	1 0	1 0

**Table 13.15** Example 13.6: Excitation table

PS		I/P	NS		Required excitations				O/P			
			$y_1$	$y_2$	x	$Y_1$	$Y_2$	$S_1$	$R_1$	$S_2$	$R_2$	
0	0	0	0	0	0	0 1	0 0	0	x	1	0	0 0
0	0	1	0	0	0	0 0	0 0	0	x	0	x	0 0
0	1	0	1	1	0	1 1	1 1	1	0	x	0	0 1
0	1	1	1	1	0	1 1	1 1	1	0	x	0	0 1
1	1	0	1	0	0	1 0	1 0	x	0	0	1	1 1
1	1	1	1	0	0	1 0	1 0	x	0	0	1	1 1
1	0	0	1	0	0	1 0	1 0	x	0	0	x	1 0
1	0	1	0	0	0	0 0	0 0	0	1	0	x	1 0

**Figure 13.39** Example 13.6: K-maps.**Figure 13.40** Example 13.6: Logic diagram of the sequential machine.

**EXAMPLE 13.7** Design a 2-input 2-output synchronous sequential circuit which produces an output  $z = 1$ , whenever any of the following input sequences 1100, 1010, or 1001 occurs. The circuit resets to its initial state after a 1 output has been generated.

**Solution**

**Step 1. Word statement of the problem:** We have to design a 2-input 2-output synchronous sequential circuit (i.e. a circuit with one input terminal and one output terminal) which produces an output  $z = 1$ , whenever any of the following input sequences 1100, 1010, or 1001 occurs. The circuit resets to its initial state after a 1 output has been generated. Assume that overlapping is permitted.

**Steps 2 and 3. State diagram and state table:** Let the sequential machine be initially in state A. The state diagram and the state table are shown in Figure 13.41.

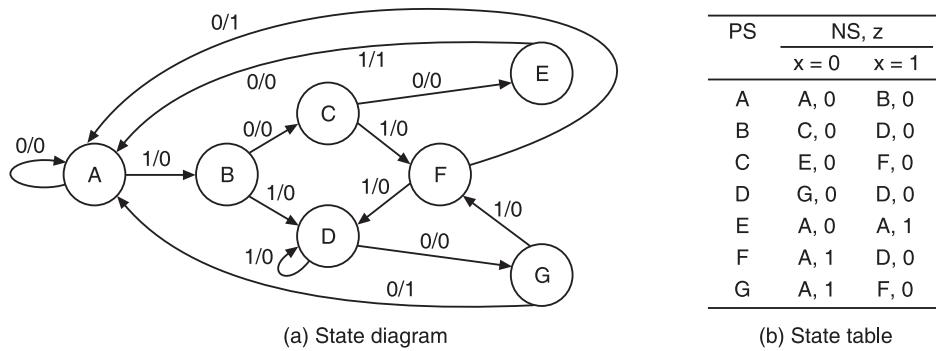


Figure 13.41 Example 13.7.

**Step 4. Reduced standard form state table:** The machine is already in this form. So no need to do anything.

**Step 5. State assignment and transition and output table:** There are seven states. So, three state variables are required which can give a maximum of eight possible states. Since only seven states are utilized, one is invalid and the corresponding excitations to the flip-flops are don't cares. The state assignment is arbitrary. Let it be A  $\rightarrow$  000, B  $\rightarrow$  010, C  $\rightarrow$  011, D  $\rightarrow$  111, E  $\rightarrow$  110, F  $\rightarrow$  100, and G  $\rightarrow$  101. Three flip-flops are used. Draw the transition and output table as shown in Table 13.16.

Table 13.16 Example 13.7: Transition and output table

	PS			NS ( $Y_1 Y_2 Y_3$ )			O/P (z)	
	$y_1$	$y_2$	$y_3$	$x = 0$	$x = 1$	$x = 0$	$x = 1$	
A $\rightarrow$	0	0	0	0 0 0	0 1 0	0	0	0
B $\rightarrow$	0	1	0	0 1 1	1 1 1	0	0	0
C $\rightarrow$	0	1	1	1 1 0	1 0 0	0	0	0
D $\rightarrow$	1	1	1	1 0 1	1 1 1	0	0	0
E $\rightarrow$	1	1	0	0 0 0	0 0 0	0	1	1
F $\rightarrow$	1	0	0	0 0 0	1 1 1	1	0	0
G $\rightarrow$	1	0	1	0 0 0	1 0 0	1	0	0

*Step 6. Choose type of flip-flops and form the excitation table:* Select J-K FFs as memory elements. Draw the excitation table (Table 13.17).

**Table 13.17** Example 13.7: Excitation table

<b>PS</b>			<b>I/P</b>	<b>NS</b>			<b>Present excitations required</b>				<b>O/P</b>		
$y_1$	$y_2$	$y_3$	$x$	$Y_1$	$Y_2$	$Y_3$	$J_1$	$K_1$	$J_2$	$K_2$	$J_3$	$K_3$	$z$
0	0	0	0	0	0	0	0	$\times$	0	$\times$	0	$\times$	0
0	0	0	1	0	1	0	0	$\times$	1	$\times$	0	$\times$	0
0	1	0	0	0	1	1	0	$\times$	$\times$	0	1	$\times$	0
0	1	0	1	1	1	1	1	$\times$	$\times$	0	1	$\times$	0
0	1	1	0	1	1	0	1	$\times$	$\times$	0	$\times$	1	0
0	1	1	1	1	0	0	1	$\times$	$\times$	1	$\times$	1	0
1	1	1	0	1	0	1	$\times$	0	$\times$	1	$\times$	0	0
1	1	1	1	1	1	1	$\times$	0	$\times$	0	$\times$	0	0
1	1	0	0	0	0	0	$\times$	1	$\times$	1	0	$\times$	0
1	1	0	1	0	0	0	$\times$	1	$\times$	1	0	$\times$	1
1	0	0	0	0	0	0	$\times$	1	0	$\times$	0	$\times$	1
1	0	0	1	1	1	1	$\times$	0	1	$\times$	1	$\times$	0
1	0	1	0	0	0	0	$\times$	1	0	$\times$	$\times$	1	1
1	0	1	1	1	0	0	$\times$	0	0	$\times$	$\times$	1	0

*Step 7. K-maps and minimal expressions:* Draw the K-maps using the entries of the excitation table and obtain the minimal expressions for  $J_3$ ,  $K_3$ ,  $J_2$ ,  $K_2$ ,  $J_1$ ,  $K_1$ , and  $z$  in terms of  $y_1$ ,  $y_2$ ,  $y_3$ , and  $x$  as follows:

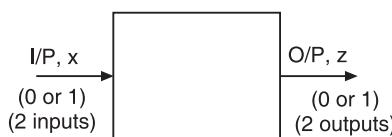
$$\begin{aligned} J_1 &= y_2x + y_3, & K_1 &= \bar{y}_2\bar{x} + y_2\bar{y}_3; \\ J_2 &= \bar{y}_3x, & K_2 &= y_1\bar{y}_3 + y_1\bar{x} + \bar{y}_1y_3x; \\ J_3 &= \bar{y}_1y_2 + y_1\bar{y}_2x, & K_3 &= \bar{y}_1 + \bar{y}_2; \\ z &= \bar{y}_2y_3\bar{x} + y_1\bar{y}_2\bar{y}_3\bar{x} + y_1y_2\bar{y}_3x \end{aligned}$$

*Step 8. Implementation:* The logic diagram can now be realized using these minimal expressions.

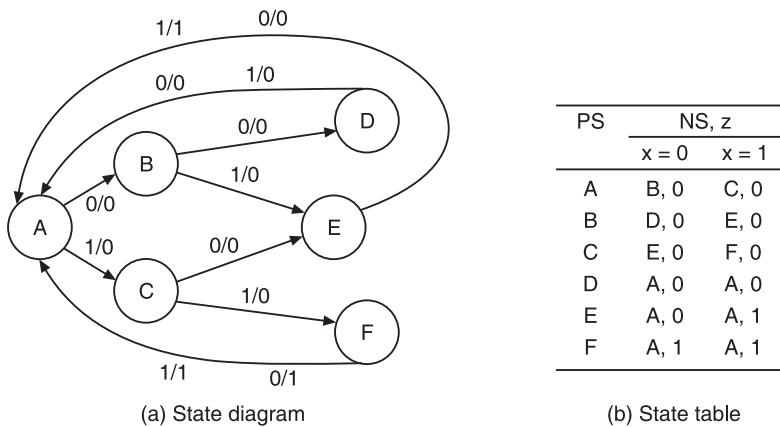
**EXAMPLE 13.8** Draw the state diagram and the state table of a 2-input 2-output synchronous sequential circuit which examines the input sequence in non-overlapping strings of three inputs each and produces a 1 output coincident with the last input of the string if and only if the string consists of either two or three 1s. For example, if the input sequence is 010101110, the required output sequence is 000001001.

### Solution

The block diagram of the sequential machine is shown in Figure 13.42. The state diagram and the state table of the sequential circuit are shown in Figure 13.43.



**Figure 13.42** Example 13.8: Block diagram of the sequential machine.



**Figure 13.43** Example 13.8: State diagram and state table.

Assume that initially the machine is in state A. While at A it may receive the first bit either as a 0 or as a 1. Both are valid for the sequence. So, the machine may go to state B (if the first bit is a 0) or to state C (if the first bit is a 1). While at B, the machine may receive the next bit as a 0 or a 1. If it is a 1, the last two bits are 01, which are part of the valid string. So, the machine goes to state E. If it is a 0, the last two bits are 00, which are not part of the valid string. But since the machine cannot start processing the next string from here (since it is non overlapping type) it goes to state D just to provide a time delay and then goes to A whether the third bit received is a 0 or a 1. While at E, if the next bit is a 0, the string is 010 which is invalid and so the output is a 0; if it is a 1, the string is 011 which is valid and, so, the machine outputs a 1. In both the cases, the machine goes to A. While at C, the machine may receive a 0 or a 1. If it is a 0, the machine goes to E because the first two bits are 10. If it is a 1, it goes to state F. The first two bits are now 11. While at F, whether the next bit is a 0 or a 1 the machine outputs a 1 because both the sequences 110 and 111 are valid and goes to state A because overlapping is not permitted.

**EXAMPLE 13.9** A clocked sequential circuit with single input x and single output z produces an output  $z = 1$  whenever the input x completes the sequence 1011 and overlapping is allowed.

(a) Obtain the state diagram.

(b) Obtain its minimum state table and design the circuit with D flip-flops.

### Solution

**Step 1. Word statement of the problem:** A single input (x), single output (z) synchronous sequential circuit to detect the sequence 1011 and to produce an output  $z = 1$ , whenever the input x completes the sequence is to be designed. It is given that overlapping is permitted.

*Steps 2 and 3. State diagram and state table:* The state diagram is drawn as shown in Figure 13.44. Let the machine be initially in state A. While at A it receives the first bit which may be a 0 or a 1. If it is a 0, the detection does not start. So the machine remains in state A itself and outputs a 0. If the first bit is a 1, it outputs a 0 and goes to state B. While at B, it receives the next bit which may be either a 0 or a 1. If it is a 0, the last two bits will be 10 which is a part of the valid sequence. So it outputs a 0 and goes to the next state C. If it is a 1, the last two bits become 11 which is not a part of the valid sequence. So it outputs a 0 and since overlapping is permitted it remains at state B (to utilize the last 1 bit). While at C it receives the third bit which may be a 0 or a 1. If it is a 0 the last 3 bits will be 100 which is not a part of the valid sequence and so it outputs a 0 and goes to state A because the detection has to start afresh. If it is a 1, the last 3 bits will be 101 which is part of the valid sequence. So the machine outputs a 0 and goes to state D. While at D it receives the fourth bit which may be either a 0 or a 1. If it is a 0, the last four bits become 1010 which is not a valid sequence. So it outputs a zero and since overlapping is permitted it goes to state C to utilize the last two bits 10. If it is a 1, the last 4 bits become 1011 which is a valid sequence. So the machine outputs a 1 and goes to state B to utilize the last bit 1. The state table is also shown in Figure 13.44.

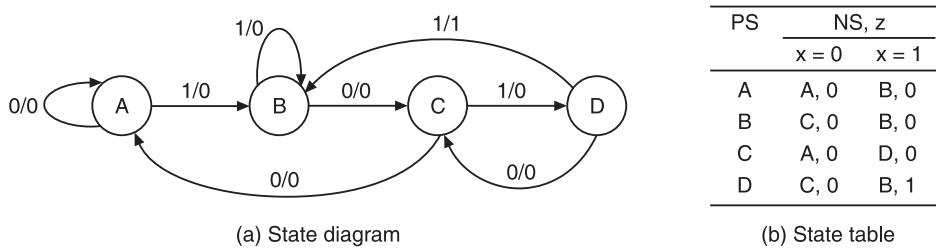


Figure 13.44 Example 13.9.

*Step 4. Reduced standard form state table:* Since there are no redundant states, the state table shown in Figure 13.44 itself is the minimal.

*Step 5. State assignment and transition and output table:* There are four states. So two state variables are required. Two state variables can have a maximum of four states. So all the states are utilized and there are no invalid states. Hence there are no don't cares. Assign the states arbitrarily. Let A  $\rightarrow$  00, B  $\rightarrow$  01, C  $\rightarrow$  10, and D  $\rightarrow$  11 be the state assignment. With this state assignment, draw the transition and output Table 13.18.

Table 13.18 Example 13.9: Transition and output table

PS	NS ( $Y_1 Y_2$ )		O/P (z)	
	$y_1 y_2$	x = 0	x = 1	x = 0
A $\rightarrow$ 0 0	0 0	0 1	0	0
B $\rightarrow$ 0 1	1 0	0 1	0	0
C $\rightarrow$ 1 0	0 0	1 1	0	0
D $\rightarrow$ 1 1	1 0	0 1	0	1

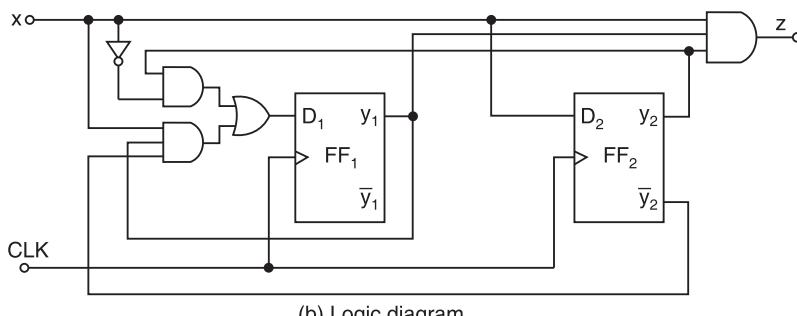
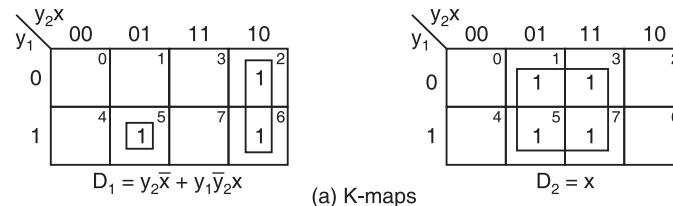
*Step 6. Choose type of flip-flops and form the excitation table:* It is given that the memory elements are D flip-flops. With D flip-flops as memory elements draw the excitation Table 13.19.

**Table 13.19** Example 13.9: Excitation table

PS		I/P	NS		I/P to FFs		O/P
y <sub>1</sub>	y <sub>2</sub>	x	Y <sub>1</sub>	Y <sub>2</sub>	D <sub>1</sub>	D <sub>2</sub>	z
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	1	0
1	0	0	0	0	0	0	0
1	0	1	1	1	1	1	0
1	1	0	1	0	1	0	0
1	1	1	0	1	0	1	1

*Step 7. K-maps and minimal expressions:* Draw the K-maps and simplify them to obtain the minimal expressions for excitations D<sub>2</sub>, and D<sub>1</sub> in terms of present state outputs y<sub>1</sub> and y<sub>2</sub> and the input x as shown in Figure 13.45a. The expression for the output of the machine z in terms of y<sub>1</sub>, y<sub>2</sub> and x can be obtained directly from the excitation table.

*Step 8. Implementation:* The logic diagram based on those minimal expressions is shown in Figure 13.45b.



**Figure 13.45** Example 13.9: K-maps and logic diagram.

**EXAMPLE 13.10** A clocked sequential circuit is provided with a single input  $x$  and a single output  $z$ . whenever the input produces a string of pulses 111 or 000 and at the end of the sequence it produces an output  $z = 1$  and overlapping is also allowed.

- Obtain the state diagram.
- Obtain the state table.
- Design the sequence detector.

**Solution**

*Step 1. Word statement of the problem:* A sequential circuit with one input terminal and one output terminal to detect the sequence 111 or 000 is to be designed. Overlapping is permitted.

*Steps 2 and 3. State diagram and state table:* Let the machine be initially in state A. While at A the machine may receive the first bit as a 0 or as a 1. Both are valid and the machine outputs a 0 and will go to state B if the first bit is a 0 and to state C if the first bit is a 1. While at B the machine may receive the next bit as a 0 or as a 1. If the bit is a 0, the first two bits become 00 which is a part of the valid string and so it outputs a 0 and goes to next state D. If the bit is a 1, the first two bits become 01 which is not a part of the valid string. It outputs a 0 and since overlapping is permitted, to utilize the last bit 1 the machine will go to state C. While at C the machine may receive the next bit as a 0 or as a 1. If the bit is a 0, the last two bits will be 10 which is not a part of the valid string. It outputs a 0 and since overlapping is permitted to utilize the last 0, the machine will go to state B. If the bit is a 1, the last two bits will be 11 which is a part of the valid sequence. So the machine goes to state E and outputs a 0. While at D the machine may receive the next bit as a 0 or as a 1. If the bit is a 0, the last three bits become 000 which is a valid string. So the machine outputs a 1 and since overlapping is permitted it remains on state D to utilize the last two zeros. If the bit is a 1, the last three bits become 001 which is not a valid string. So it outputs a 0 and since overlapping is permitted it goes to state C to utilize the last 1. While at E the machine may receive the next bit as a 0 or as a 1. If the bit is a 0, the last three bits become 110 which is not a valid string. So it outputs a 0 and since overlapping is permitted it goes to state B to utilize the last 0. If the bit is a 1, the last three bits will be 111 which is a valid sequence. So the machine outputs a 1 and remains in the same state to utilize the last two 1s. The state diagram and the state table are as shown in Figure 13.46.

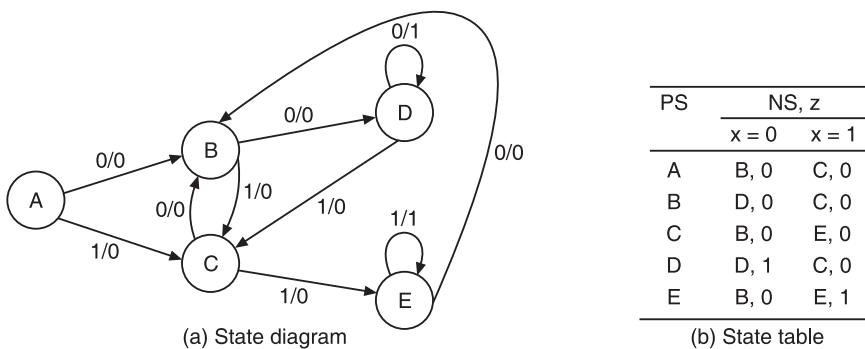


Figure 13.46 Example 13.10.

*Step 4. Reduced standard form state table:* Since there are no redundant states the state table shown in Figure 13.46 itself is the minimal.

*Step 5. State assignment and transition and output table:* There are five states. So three state variables are required. The three state variables can have a maximum of eight states. Out of those, five are valid and three are invalid. The invalid states can be treated as don't cares. Assign the states arbitrarily, as

$$A \rightarrow 000, B \rightarrow 001, C \rightarrow 010, D \rightarrow 011, \text{ and } E \rightarrow 100$$

States 101, 110 and 111 are invalid. With this state assignment, the transition and output table is as shown in Table 13.20.

**Table 13.20** Example 13.10: Transition and output table

PS $y_1\ y_2\ y_3$	NS ( $Y_1 Y_2 Y_3$ )		O/P (z)	
	x = 0	x = 1	x = 0	x = 1
0 0 0	0 0 1	0 1 0	0	0
0 0 1	0 1 1	0 1 0	0	0
0 1 0	0 0 1	1 0 0	0	0
0 1 1	0 1 1	0 1 0	1	0
1 0 0	0 0 1	1 0 0	0	1

*Step 6. Choose type of flip-flops and form the excitation table:* The circuit is to be designed using D flip-flops. So selecting D flip-flops as memory elements the excitation table is as shown in Table 13.21.

**Table 13.21** Example 13.10: Excitation table

PS			I/P	NS			I/P			O/P
$y_1$	$y_2$	$y_3$	x	$Y_1$	$Y_2$	$Y_3$	$D_1$	$D_2$	$D_3$	z
0	0	0	0	0	0	1	0	0	1	0
0	0	0	1	0	1	0	0	1	0	0
0	0	1	0	0	1	1	0	1	1	0
0	0	1	1	0	1	0	0	1	0	0
0	1	0	0	0	0	1	0	0	1	0
0	1	0	1	1	0	0	1	0	0	0
0	1	1	0	0	1	1	0	1	1	1
0	1	1	1	0	1	0	0	1	0	0
1	0	0	0	0	0	1	0	0	1	0
1	0	0	1	1	0	0	1	0	0	1

*Step 7. K-maps and minimal expressions:* The K-maps, their minimization, and the minimal expressions for excitations  $D_1$ ,  $D_2$ ,  $D_3$  and output z obtained from them are shown in Figure 13.47.

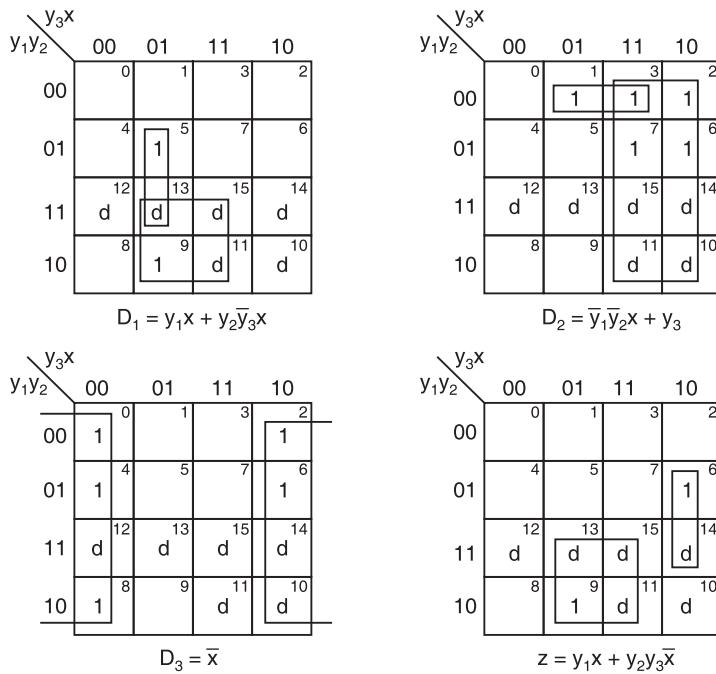


Figure 13.47 Example 13.10: K-maps.

*Step 8. Implementation:* A logic diagram can be drawn based on those minimal expressions.

**EXAMPLE 13.11** The synchronous circuit shown in Figure 13.48 where  $D$  denotes a unit delay produces a periodic binary output sequence. Assume that initially  $x_1 = 1, x_2 = 1, x_3 = 0$ , and  $x_4 = 0$  and that the initial output sequence is 1100101000. Thereafter, the sequence repeats itself. Find a minimal expression for the combinational circuit  $f(x_1, x_2, x_3, x_4)$ . The clock need not be included in the expression although it is implicit.

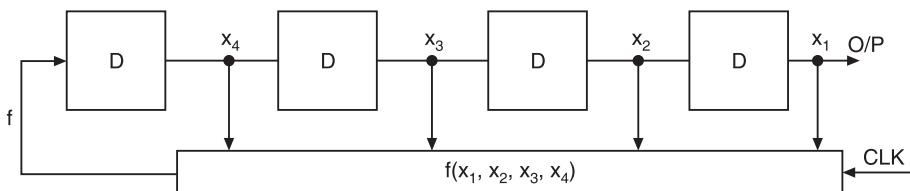


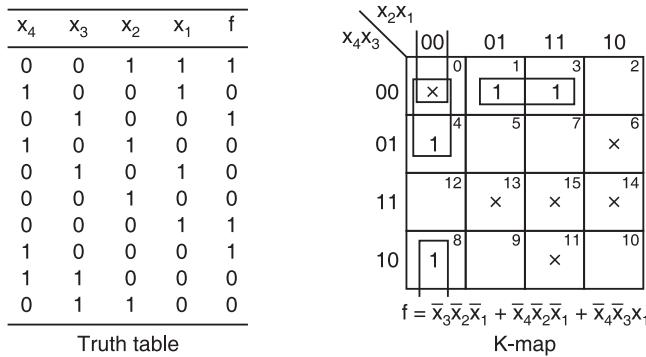
Figure 13.48 Example 13.11: Synchronous machine.

### Solution

The output  $x_1$  of the last delay unit is the output bit of the synchronous circuit. There are four delay units arranged in the form of a shift register. It is given that the output sequence 1100101000 repeats itself. That means, the input to the first flip-flop which is the output of the combinational circuit must also be the same sequence. Since initially  $x_1 = 1, x_2 = 1, x_3 = 0$ , and  $x_4 = 0$ , the sequence of the bits at the input of the first delay element, i.e. bits of  $f$  must come in the order 1010001100 to get the given output sequence. There are four state

variables which can give a maximum of 16 possible states. Only 10 of them are utilized, so, there are six invalid states. The values of f corresponding to those states are the don't cares.

The initial state is 0011 and f = 1. The first clock pulse shifts each bit to the right by one place. So,  $x_1$  is shifted out, and f is shifted to  $x_4$ 's place, and the next f becomes the input to the first flip-flop. Thus, the next state is 1001 and f = 0, and so on. After 10 clock pulses the sequence repeats itself. The truth table and K-map are shown in Figure 13.49.



**Figure 13.49** Example 13.11: Truth table and K-map.

### SHORT QUESTIONS AND ANSWERS

1. What is a sequential machine?  
A. A sequential machine is nothing but a sequential circuit represented by a block or a module with inputs and outputs indicated. Digital computers and digital communication systems are examples of synchronous sequential machines.
2. What are finite state machines?  
A. Finite state machines are those machines whose past histories can affect their future behaviour only in a finite number of ways, i.e. they are machines with a finite number of states. They are abstract models describing the synchronous sequential machines.
3. List the various memory elements used in sequential machines.  
A. The various memory elements used in sequential machines are—D flip-flop, T flip-flop, SR flip-flop, and J-K flip-flop.
4. How is the state of the memory element specified?  
A. The memory element has two states. Its state is specified by the value of its output which may assume either a 0 or a 1. It is represented by a state variable.
5. What do you mean by the term 'state diagram'? What do the vertices, the directed arcs, and the labels on the arcs of a state diagram represent?  
A. The state diagram or state graph is a pictorial representation of the relationships between the present state, the input, the next state, and the output of the finite state sequential machine. The vertices (nodes) of the graph represent the states of the machine. The directed arcs emanating from each vertex indicate the state transitions caused by various input symbols (i.e. the direction of arrows point to the next state that the machine will go after the input is applied). The label on the directed arc indicates the input symbol that causes the transition, and the output symbol that is to be generated.

## 736 FUNDAMENTALS OF DIGITAL CIRCUITS

6. What is state assignment?
  - A. The process of assigning the states of a physical device to the states of a sequential machine is known as state assignment.
7. What do you mean by the term ‘state table’? What does each row, column and entry of the state table represent?
  - A. The state table is a tabular representation of the relationship between the present state, the input, the next state and the output. Each column of the state table corresponds to one input symbol, and each row of the state table corresponds to one state. The entries corresponding to each combination of the input symbols and the present state specify the output that will be generated and the next state to which the machine will go.
8. Compare the state diagram and the state table.
  - A. Both the state diagram and the state table contain the same information and the choice between the two representations is a matter of convenience. Both have the advantage of being precise, unambiguous, and thus more suitable for describing the operation of a sequential machine than that by any verbal description. The succession of states through which a sequential machine passes and the output sequence which it produces in response to a known input sequence are specified uniquely by the state diagram or by the state model and the initial state.
9. What do you mean by initial state and final state?
  - A. The initial state refers to the state of the machine prior to the application of the input sequence and the final state refers to the state of the machine after the application of the input sequence.
10. What is an excitation table? What information does it give?
  - A. An excitation table is a table which lists the present states, the excitations and the next states. It gives information about the excitations or inputs required to be applied to the memory elements in the sequential circuit to bring the sequential machine from the present state to the next state. It also gives information about the outputs of the machine after application of the present inputs.
11. What is a transition and output table?
  - A. The transition and output table of a sequential machine is a table which lists the present state, the next state to which it will go and the output it produces. It can be obtained from the state table by modifying the entries of the state table to correspond to the states of the machine, in accordance with the selected state assignment. In this table the next state and output entries are separated into two sections. The next state part of the state table is called the transition table.
12. Define an input alphabet and an output alphabet.
  - A. The set of all possible combinations of inputs is called an input alphabet and the set of all possible combinations of outputs is called an output alphabet.
13. What are state variables?
  - A. The output values of physical devices are referred to as state variables.
14. What is the Mealy model of the state diagram of a memory element?
  - A. In the Mealy model of the state diagram each node in the state diagram represents a particular state of the FF (0 or 1). The labels on the arcs indicate the input/output, i.e. the input that is given when the FF is in a particular state and the corresponding output. The directions of the arrows point to the next state the FF will go after the input is applied.
15. What is the Moore model of the state diagram of a memory element?
  - A. In the Moore model of the state diagram, the state code and the value of the output are written inside the circle. The directed line joining one node to the other, or looping back to the same node has the value of the input written beside the line.

- 16.** What is a serial binary adder?  
A. A serial binary adder is a sequential circuit which adds two binary numbers serially.
- 17.** What is a sequence detector?  
A. A sequence detector is a sequential machine which produces an output 1 every time the desired sequence is detected, and an output 0 at all other times.
- 18.** What is a serial parity-bit generator?  
A. A serial parity-bit generator is a two-terminal circuit which receives coded messages, so that the resulting outcome is an error-detecting coded message.

### REVIEW QUESTIONS

1. Write the main steps in the synthesis of synchronous sequential circuits.
2. What is a serial adder? Explain its working with the help of a state diagram.
3. What is a sequence detector? Explain its working with the help of a state diagram to detect any arbitrary sequence.
4. What do you mean by a parity-bit generator? Explain its working with the help of a state diagram to generate odd parity.

### FILL IN THE BLANKS

1. The \_\_\_\_\_ of the state diagram represents the states of the machine.
2. The state of the machine after the application of the input sequence is called the \_\_\_\_\_.
3. The output values of physical devices are referred to as \_\_\_\_\_.
4. An up-down counter is also called a \_\_\_\_\_ counter or a \_\_\_\_\_ counter.
5. Synchronous counters have the advantages of \_\_\_\_\_ and \_\_\_\_\_, but the disadvantage of having \_\_\_\_\_ than that of asynchronous counters.
6. Presetting a counter is also referred to as \_\_\_\_\_ a counter.
7. State diagram can also be called the \_\_\_\_\_ diagram.
8. The next state part of the state table is called the \_\_\_\_\_ table.
9. The process of assigning the states of a physical device to the states of a sequential machine is known as \_\_\_\_\_.

### OBJECTIVE TYPE QUESTIONS

1. The design of a clocked sequential circuit requires
 

(a) the state reduction	(b) the state assignment
(c) the design of the next state decoder	(d) all of the above

## 738 FUNDAMENTALS OF DIGITAL CIRCUITS

2. In a sequential circuit design, state reduction is done for designing the circuit with
  - (a) a minimum number of flip-flops
  - (b) a minimum number of gates
  - (c) a minimum number of gates and memory elements
  - (d) none of the above
3. A sequential circuit with ten states will have
  - (a) 10 flip-flops
  - (b) 5 flip-flops
  - (c) 4 flip-flops
  - (d) 0 flip-flops
4. The output of a clocked sequential circuit is independent of the input. The circuit can be represented by
  - (a) Mealy model
  - (b) Moore model
  - (c) either Mealy or Moore model
  - (d) neither Mealy nor Moore model
5. For designing a finite state machine K-maps can be used for minimizing the
  - (a) excitation expressions of flip-flops
  - (b) number of flip-flops
  - (c) output logic expressions
  - (d) excitation and output logic expressions.
6. While constructing a state diagram of sequential circuit from the set of given statements,
  - (a) a minimum number of states must only be used
  - (b) redundant states may be used
  - (c) redundant states must be avoided
  - (d) none of the above is relevant
7. A finite state machine
  - (a) is the same as a clocked sequential circuit
  - (b) consists of combinational logic circuits only
  - (c) consists of electrical motors
  - (d) does not exist in practice
8. A serial adder can be designed
  - (a) using only gates
  - (b) using only flip-flops
  - (c) as a combinational circuit
  - (d) as a sequential circuit
9. The number of directed arcs emanating from any state in a state diagram is
  - (a)  $2^n$ , where  $n$  is the number of inputs
  - (b) independent of the number of inputs
  - (c) an arbitrary number
  - (d)  $2^n$ ,  $n$  is the number of flip-flops
10. The number of directed arcs terminating on any state of a state diagram is
  - (a)  $2^n$  where  $n$  is the number of inputs
  - (b)  $2^n$  where  $n$  is the number of flip-flops in the circuit
  - (c) independent of the number of inputs
  - (d) dependent on the number of outputs

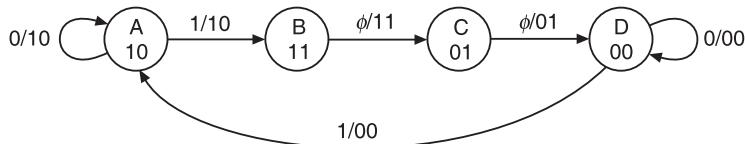
### PROBLEMS

- 13.1 A long sequence of pulses enters a 2-input 2-output synchronous sequential circuit which is required to produce an output  $z = 1$ , whenever the sequence 1101 occurs. Overlapping sequences are accepted. Design the circuit.

- 13.2** Design a 3-bit counter which counts in the following sequence.

$0 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 0 \rightarrow 2 \rightarrow \dots$ , etc.

- 13.3** Design a circuit using S-R FFs that will function as per the state diagram shown below.



- 13.4** Design a 2-input 2-output synchronous sequential circuit which produces an output  $z = 1$ , whenever any of the following input sequences—1101, 1011, or 1001—occurs. The circuit resets to the initial state after a 1 output is generated.
- 13.5** A long sequence of pulses enters a 2-input 2-output synchronous sequential circuit which produces an output pulse  $z = 1$ , whenever the sequence 10010 occurs. The overlapping sequences are accepted. Draw the state diagram, select an assignment and show the excitation table.
- 13.6** Design a sequence detector which generates an output  $z = 1$ , whenever the string is 0110, and generates a 0 at all other times. The overlapping sequences are detected. Implement the circuit using D FFs.
- 13.7** Design a 3-bit up/down counter which counts up when the control signal  $M = 1$  and counts down when  $M = 0$ .
- 13.8** Draw the state diagram and the state table for a 4-bit odd-parity generator.
- 13.9** Construct the state diagram and the state table for a 2-input machine, which produces an output  $z = 1$ , whenever the last string of five inputs contains exactly four zeros and the string starts with three zeros. Analysis of the next string does not start until the end of this string of five inputs, whether or not it produces a 1 output.

## VHDL PROGRAMS

### 1. VHDL PROGRAM FOR FOUR-BIT EVEN PARITY GENERATOR USING DATA FLOW MODELING

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity EVEN_PARITY_GENERATOR is
    Port ( DATA : in STD_LOGIC_VECTOR (3 downto 0);
           EVEN_PARITY : out STD_LOGIC);
end EVEN_PARITY_GENERATOR;

architecture Behavioral of EVEN_PARITY_GENERATOR is
begin
EVEN_PARITY <= (DATA(0) XOR DATA(1) XOR DATA(2) XOR DATA(3));
end Behavioral;

```

#### SIMULATION OUTPUT:



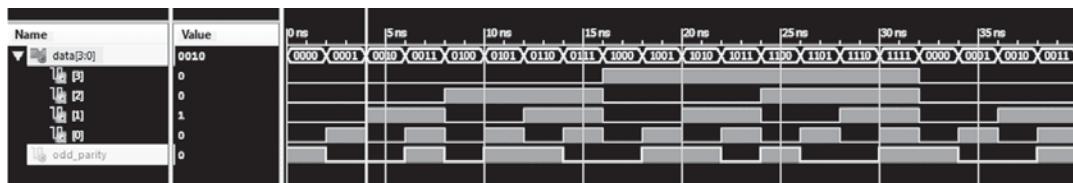
### 2. VHDL PROGRAM FOR FOUR-BIT ODD PARITY GENERATOR USING DATA FLOW MODELING

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ODD_PARITY_GENERATOR is
    Port ( DATA : in STD_LOGIC_VECTOR (3 downto 0);
           ODD_PARITY : out STD_LOGIC);
end ODD_PARITY_GENERATOR;
architecture Behavioral of ODD_PARITY_GENERATOR is
begin
ODD_PARITY <= (NOT (DATA(0) XOR DATA(1) XOR DATA(2) XOR DATA(3)));
end Behavioral;

```

#### SIMULATION OUTPUT:



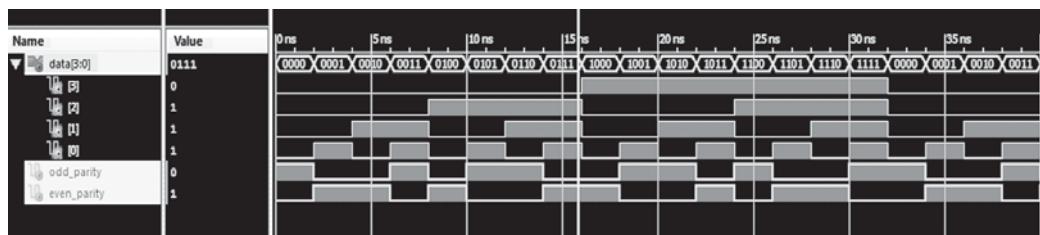
### 3. VHDL PROGRAM FOR FOUR-BIT EVEN AND ODD PARITY GENERATOR USING DATA FLOW MODELING

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity PARITY_GENERATOR is
    Port ( DATA : in STD_LOGIC_VECTOR (3 downto 0);
           ODD_PARITY : out STD_LOGIC;
           EVEN_PARITY : out STD_LOGIC);
end PARITY_GENERATOR;
architecture Behavioral of PARITY_GENERATOR is
begin
EVEN_PARITY <= (DATA(0) XOR DATA(1) XOR DATA(2) XOR DATA(3));
ODD_PARITY <= (NOT (DATA(0) XOR DATA(1) XOR DATA(2) XOR
DATA(3)));
end Behavioral;

```

#### SIMULATION OUTPUT:



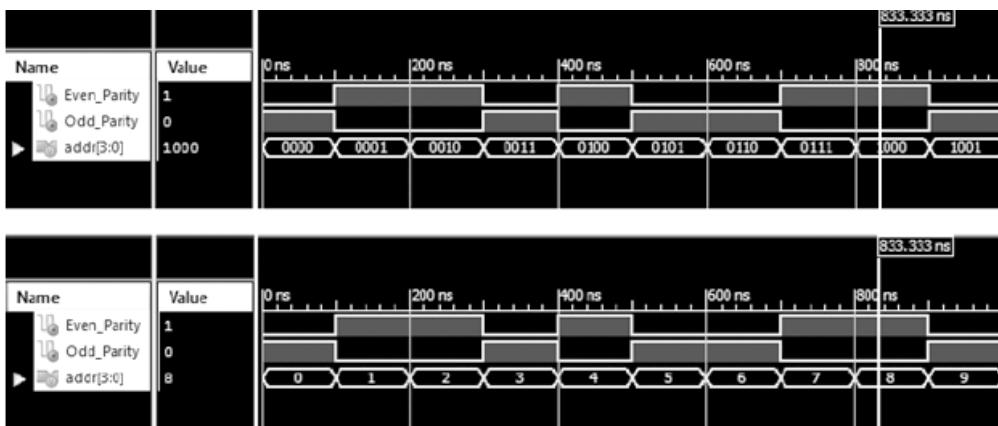
## VERILOG PROGRAMS

### 1. VERILOG PROGRAM TO FIND EVEN & ODD PARITY IN THE GIVEN SEQUENCE

```

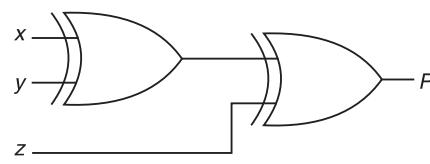
module parity (addr, Even_Parity,Odd_Parity);
input [3:0] addr;
output reg Even_Parity;
output reg Odd_Parity;
always@(addr)
begin
Even_Parity = calc_parity(addr);
Odd_Parity = (~(calc_parity(addr)));
end
function calc_parity;
input [3:0] address;
begin
calc_parity = ^address;
end
endfunction
endmodule

```

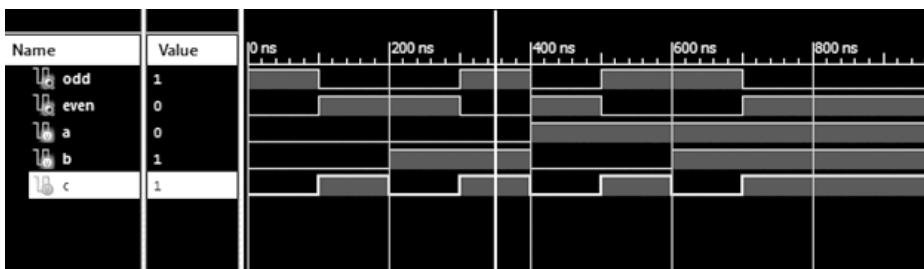
**SIMULATION OUTPUT:****2. VERILOG PROGRAM FOR 3-BIT EVEN & ODD PARITY GENERATOR USING DATA FLOW MODELING**

```
module parity_gen (a,b,c,odd,even);
input a,b,c;
output odd,even;
assign even = a ^ b ^ c;
assign odd = ~(a ^ b ^ c);
endmodule
```

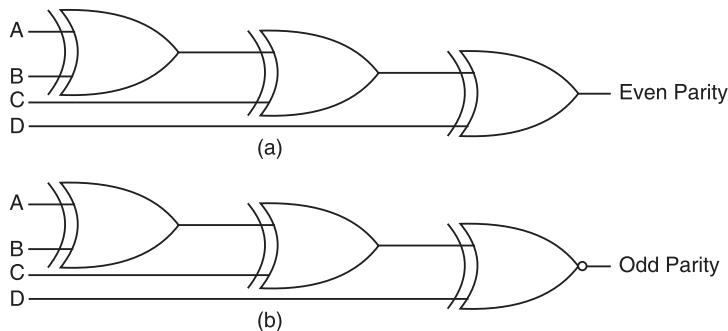
Message			
x	y	z	Parity
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



3-bit even parity generator

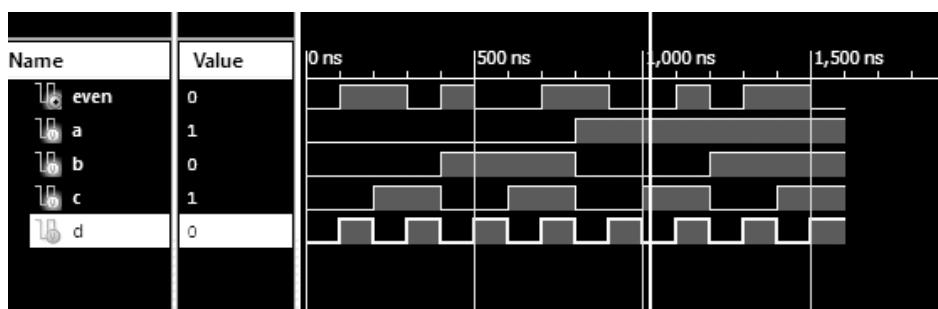
**SIMULATION OUTPUT:**

### 3. VERILOG PROGRAM FOR 4-BIT EVEN PARITY GENERATOR USING DATA GATE LEVEL MODELING



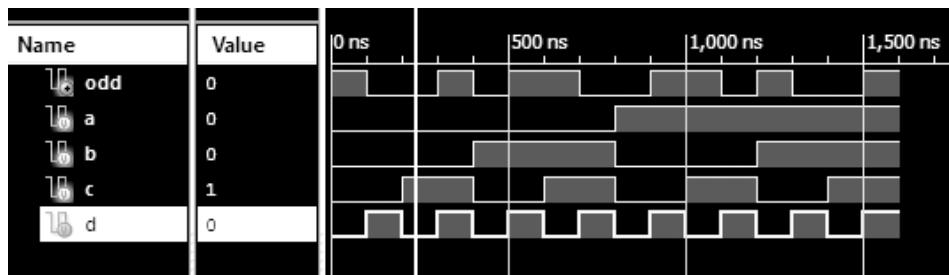
```
module even_parity_gen (a,b,c,d,even);
input a,b,c,d;
output even;
wire N1,N2;
xor x1 (N1,a,b);
xor x2 (N2,N1,c);
xor x3 (even,N2,d);
endmodule
```

#### SIMULATION OUTPUT:



### 4. VERILOG PROGRAM FOR 4-BIT ODD PARITY GENERATOR USING DATA GATE LEVEL MODELING

```
module odd_parity_gen (a,b,c,d,odd);
input a,b,c,d;
output odd;
wire N1,N2;
xor x1 (N1,a,b);
xor x2 (N2,N1,c);
xnor x3 (odd,N2,d);
endmodule
```

**SIMULATION OUTPUT:**

# 14

## SEQUENTIAL CIRCUITS-II

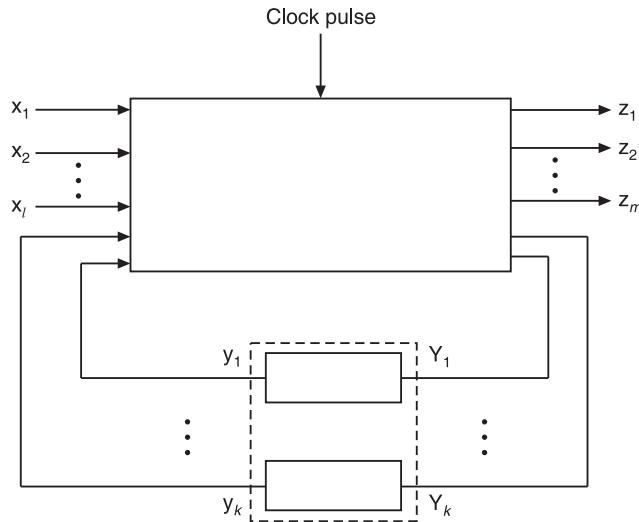
### 14.1 FINITE STATE MACHINE

The most general model of a sequential circuit has inputs, outputs and internal states. A sequential circuit is referred to as a finite state machine (FSM). A finite state machine is an abstract model that describes the synchronous sequential machine. Since in a sequential circuit the output depends on the present input as well as on the past inputs, i.e. on the past histories and since a machine might have an infinite varieties of possible histories, it would need an infinite capacity for storing them. Since it is impossible to implement machines which have infinite storage capabilities, we consider only finite state machines. Finite state machines are sequential circuits whose past histories can affect their future behaviour in only a finite number of ways, i.e. they are machines with a fixed number of states. These machines can distinguish among a finite number of classes of input histories. These classes of input histories are referred to as the internal states of the machine. Every finite state machine therefore contains a finite number of memory devices.

Figure 14.1 shows the block diagram of a finite state model.  $x_1, x_2, \dots, x_l$  are inputs.  $z_1, z_2, \dots, z_m$  are outputs.  $y_1, y_2, \dots, y_k$  are state variables, and  $Y_1, Y_2, \dots, Y_k$  represent the next state.

### 14.2 CAPABILITIES AND LIMITATIONS OF FINITE STATE MACHINES

**1. Periodic sequence of finite states:** With an  $n$ -state machine, we can generate a periodic sequence of  $n$  states or smaller than  $n$  states. For example, in a 6-state machine, we can have a maximum periodic sequence as 0, 1, 2, 3, 4, 5, 0, 1, ....



**Figure 14.1** Block diagram of a finite state model.

**2. No infinite sequence:** Consider an infinite sequence such that the output is 1 when and only when the number of inputs received so far is equal to  $P(P + 1)/2$  for  $P = 1, 2, 3, \dots$ , i.e. the desired input-output sequence has the following form:

Input:	x   x
Output:	1   0   1   0   0   1   0   0   0   1   0   0   0   0   1   0   0   0   0   0   1

Such an infinite sequence cannot be produced by a finite state machine.

**3. Limited memory:** The finite state machine has a limited memory and due to limited memory, it cannot produce certain outputs. Consider a binary multiplier circuit for multiplying two arbitrarily large binary numbers. If we implement this with a finite state machine capable of performing serial multiplication, we can find that it is not possible to multiply certain numbers. Such a limitation does occur due to the limited memory available to the machine. This memory is not sufficient to store arbitrarily large partial products resulted during multiplication.

Finite state machines are of two types. They differ in the way the output is generated. They are:

1. **Mealy type model:** In this model, the output is a function of the present state and the present input.
2. **Moore type model:** In this model, the output is a function of the present state only.

### 14.3 MATHEMATICAL REPRESENTATION OF SYNCHRONOUS SEQUENTIAL MACHINE

We know that the next state of a sequential machine depends upon the present state and the present input. The relation between the present state  $S(t)$ , present input  $x(t)$ , and next state  $S(t + 1)$  can be given as

$$S(t + 1) = f\{S(t), x(t)\}$$

The value of output  $z(t)$  can be given as

$$\begin{aligned} z(t) &= g\{S(t), x(t)\} && \text{for Mealy model} \\ z(t) &= g\{S(t)\} && \text{for Moore model} \end{aligned}$$

because in a Mealy machine, the output depends on the present state and input, whereas in a Moore machine, the output depends only on the present state. Table 14.1 shows a comparison between the Moore machine and Mealy machine.

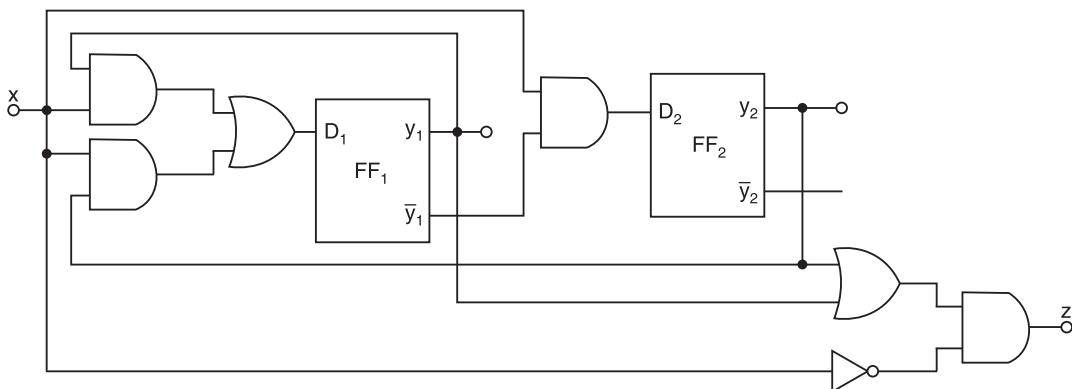
**Table 14.1** Comparison between the Moore machine and Mealy machine

Moore machine	Mealy machine
1. Its output is a function of present state only. $z(t) = g\{S(t)\}$	1. Its output is a function of present state as well as present input. $z(t) = g\{S(t), x(t)\}$
2. Input changes do not affect the output.	2. Input changes may affect the output of the circuit.
3. It requires more number of states for implementing same function.	3. It requires less number of states for implementing same function.

#### 14.4 MEALY MODEL

When the output of the sequential circuit depends on both the present state of the flip-flops and on the inputs, the sequential circuit is referred to as Mealy circuit or Mealy machine.

Figure 14.2 shows the logic diagram of a Mealy model. Notice that the output depends upon the present state as well as the present inputs. Looking at the figure, we can easily realize that changes in the input during the clock pulse cannot affect the state of the flip-flop. However, they can affect the output of the circuit. Due to this, if the input variations are not synchronized with a clock, the derived output will also not be synchronized with the clock and we get false outputs. The false outputs can be eliminated by allowing input to change only at the active transition of the clock.



**Figure 14.2** Logic diagram of a Mealy model.

The behaviour of a clocked sequential circuit can be described algebraically by means of state equations. A state equation (also called transition equation) specifies the next state as a function of the present state and inputs. The Mealy model shown in the figure consists of two D flip-flops, an

input  $x$ , and an output  $z$ . Since the D input of a flip-flop determines the value of the next state, the state equations for the model can be written as

$$y_1(t+1) = y_1(t)x(t) + y_2(t)x(t)$$

$$y_2(t+1) = \bar{y}_1(t)x(t)$$

and the output equation is

$$z(t) = \{y_1(t) + y_2(t)\}\bar{x}(t)$$

where  $y(t+1)$  is the next state of the flip-flop one clock edge later,  $x(t)$  is the present input, and  $z(t)$  is the present output. If  $y_1(t+1)$  and  $y_2(t+1)$  are represented by  $Y_1(t)$  and  $Y_2(t)$ , in more compact form, the equations are

$$y_1(t+1) = Y_1 = y_1x + y_2x$$

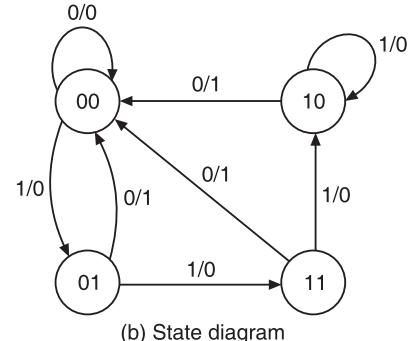
$$y_2(t+1) = Y_2 = \bar{y}_1x$$

$$z = (y_1 + y_2)\bar{x}$$

The state table of the Mealy model based on the above state equations and output equation is shown in Figure 14.3a. The state diagram based on the state table is shown in Figure 14.3b.

PS	NS		O/P	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
$y_1$	$y_2$	$Y_1$	$Y_2$	$z$
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1

(a) State table



(b) State diagram

Figure 14.3 Mealy model.

In general form, the Mealy circuit can be represented with its block schematic as shown in Figure 14.4.

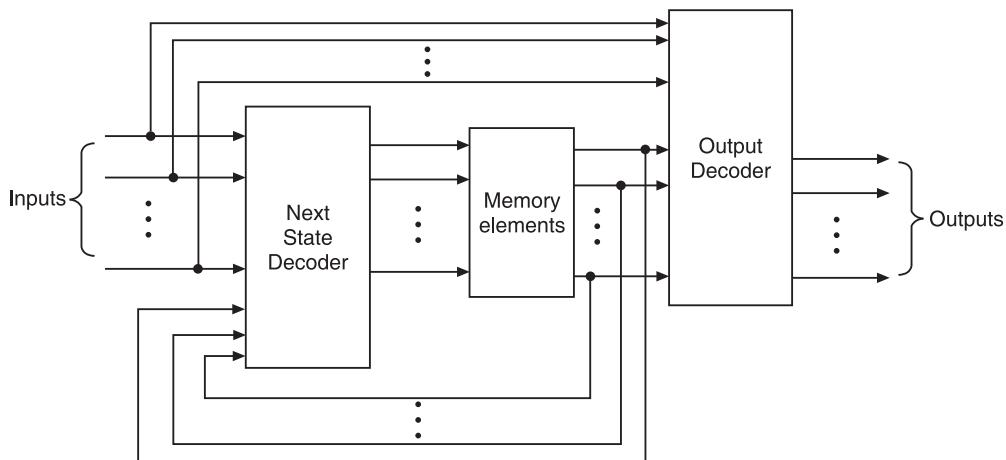


Figure 14.4 Mealy circuit model.

## 14.5 MOORE MODEL

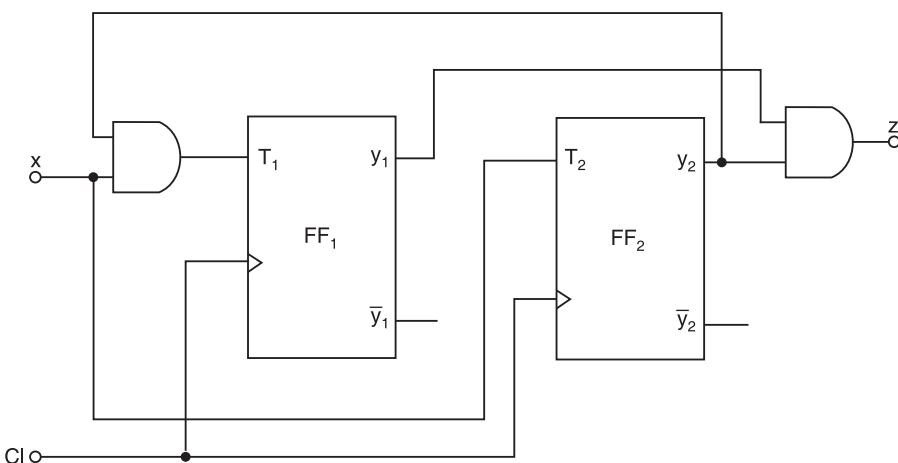
As mentioned earlier, when the output of the sequential circuit depends only on the present state of the flip-flop, the sequential circuit is referred to as the Moore circuit or the Moore machine. Figure 14.5 shows the logic diagram of a Moore circuit.

Notice that the output depends only on the present state. It does not depend upon the input at all. The input is used only to determine the inputs of flip-flops. It is not used to determine the output. The circuit shown has two T flip-flops, one input  $x$ , and one output  $z$ . It can be described algebraically by two input equations and an output equation.

$$T_1 = y_2 x$$

$$T_2 = x$$

$$z = y_1 y_2$$



**Figure 14.5** Logic diagram of a Moore model.

The characteristic equation of a T flip-flop is

$$Q(t+1) = T\bar{Q} + \bar{T}Q$$

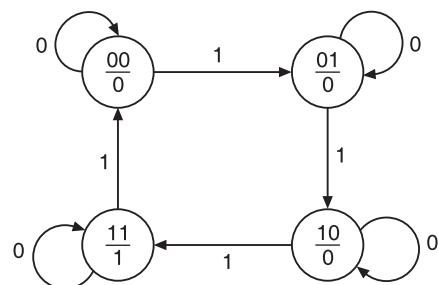
The values for the next state can be derived from the state equations by substituting  $T_1$  and  $T_2$  in the characteristic equation yielding

$$\begin{aligned} y_1(t+1) &= Y_1 = (y_2 x) \oplus y_1 = (\bar{y}_2 \bar{x}) y_1 + (y_2 x) \bar{y}_1 \\ &= y_1 \bar{y}_2 + y_1 \bar{x} + \bar{y}_1 y_2 x \\ y_2(t+1) &= x \oplus y_2 = x \bar{y}_2 + \bar{x} y_2 \end{aligned}$$

The state table of the Moore model based on the above state equations and output equation is shown in Figure 14.6a. The state diagram based on the state table is shown in Figure 14.6b.

PS	NS				O/P	
	$x = 0$		$x = 1$			
$y_1$	$y_2$	$Y_1$	$Y_2$	$Y_1$	$Y_2$	$z$
0	0	0	0	0	1	0
0	1	0	1	1	0	0
1	0	1	0	1	1	0
1	1	1	1	0	0	1

(a) State table



(b) State diagram

Figure 14.6 Moore model.

In general form, the Moore circuit can be represented with its block schematic as shown in Figure 14.7. Figure 14.8 shows the Moore circuit model with an output decoder.

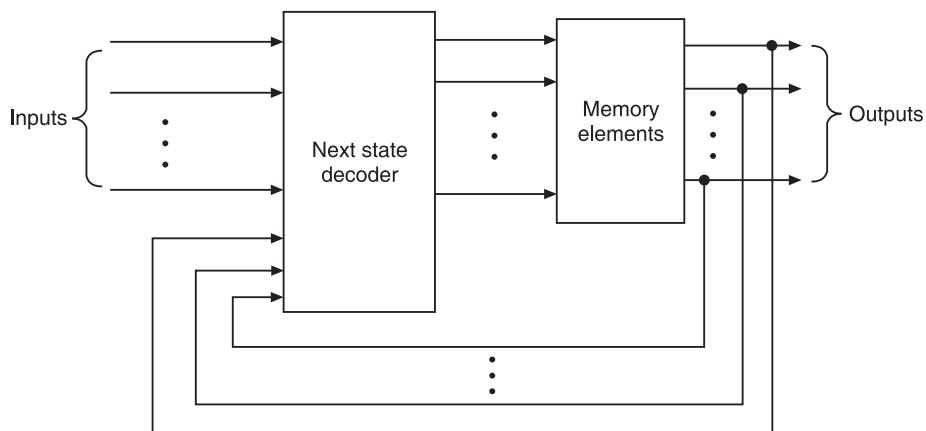


Figure 14.7 Moore circuit model.

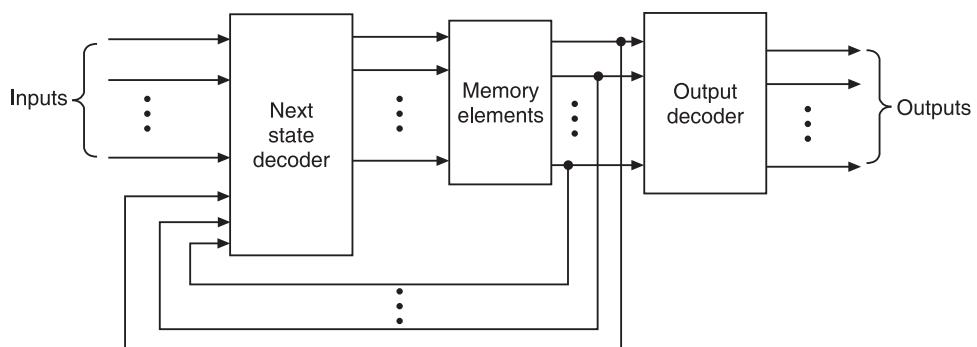


Figure 14.8 Moore circuit model with an output decoder.

## 14.6 IMPORTANT DEFINITIONS AND THEOREMS

### 14.6.1 Finite State Machine—Definitions

Consider the state diagram of a finite state machine shown in Figure 14.9. It is a five-state machine with one input variable and one output variable.

$$S = \{A, B, C, D, E\} \quad I = \{0, 1\} \quad O = \{0, 1\}$$

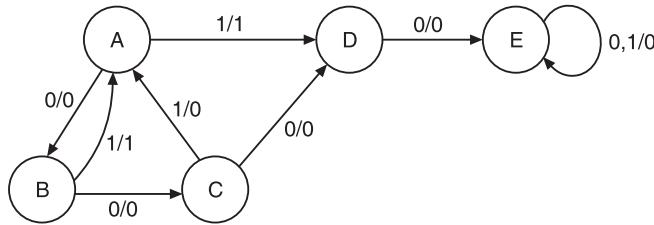


Figure 14.9 State diagram.

**Successor:** Looking at the state diagram in Figure 14.9 we can say that, when present state is A and input is 1, the next state is D. In other words, this condition is specified as D is the 1-successor of A. Similarly, we can say that A is the 1 successor of B and C, D is the 11 successor of B and C, C is the 00 successor of A, D is the 000 successor of A, E is the 10 successor of A or 0000 successor of A and so on. In general, we can say that, if an input sequence X takes a machine from state  $S_i$  to  $S_j$ , then  $S_j$  is said to be the X successor of  $S_i$ .

**Terminal state:** Looking at the state diagram of Figure 14.9, we observe that no such input sequence exists which can take the sequential machine out of state E and thus state E is said to be a terminal state.

In general, we can say that a state is a terminal state when there are no outgoing arcs which start from it and terminate in other states.

**Strongly-connected machine:** In sequential machines many times certain subsets of states may not be reachable from other subsets of states, even if the machine does not contain any terminal state. However, if for every pair of states  $S_i, S_j$  of a sequential machine there exists an input sequence which takes the machine M from  $S_i$  to  $S_j$ , then the sequential machine is said to be strongly connected.

### 14.6.2 State Equivalence and Machine Minimization

In realizing the logic diagram from a state table or state diagram many times we come across redundant states. Redundant states are states whose functions can be accomplished by other states. The elimination of redundant states reduces the total number of states of the machine which in turn results in reduction of the number of flip-flops and logic gates, reducing the cost of the final circuit.

Two states are said to be equivalent, if every possible set of inputs applied to the machine while in this state generate exactly the same output and take the machine exactly to the same next state. When two states are equivalent, one of them can be removed without altering the input-output relationship. Here we will discuss what is meant by state equivalence and how to find equivalent states for machine minimization.

**State equivalence theorem:** It states that two states  $S_1$  and  $S_2$  are equivalent if for every possible input sequence applied, the machine goes to the same next state and generates the same output. That is, if  $S_1(t+1) = S_2(t+1)$  and  $Z_1 = Z_2$ , then  $S_1 = S_2$ .

#### 14.6.3 Distinguishable States and Distinguishing Sequences

Two states  $S_A$  and  $S_B$  of a sequential machine are distinguishable, if and only if there exists at least one finite input sequence which when applied to the sequential machine causes different output sequences depending on whether  $S_A$  or  $S_B$  is the initial state. The sequence which distinguishes these states is called a distinguishing sequence of the pair  $(S_A, S_B)$ .

Consider states A and B in the state table shown in Table 14.2. When input X is 0, their outputs are 0 and 1 respectively and therefore, states A and B are called 1-distinguishable. Now consider states A and E. The output sequence is as follows:

$$X = 0 \quad \left\{ \begin{array}{l} A \rightarrow C, 0 \text{ and } E \rightarrow D, 0: \text{outputs are the same} \\ C \rightarrow E, 0 \text{ and } D \rightarrow B, 1: \text{outputs are different} \end{array} \right.$$

Here the outputs are different after 2-state transitions and hence states A and E are 2-distinguishable.

**Table 14.2** Distinguishable states

PS	NS, Z	
	X = 0	X = 1
A	C, 0	F, 0
B	D, 1	F, 0
C	E, 0	B, 0
D	B, 1	E, 0
E	D, 0	B, 0
F	D, 1	B, 0

Again consider states A and C. The output sequence is as follows:

$$X = 0 \quad \left\{ \begin{array}{l} A \rightarrow C, 0 \text{ and } C \rightarrow E, 0: \text{outputs are the same} \\ C \rightarrow E, 0 \text{ and } E \rightarrow D, 0: \text{outputs are the same} \\ E \rightarrow D, 0 \text{ and } D \rightarrow B, 1: \text{outputs are different} \end{array} \right.$$

Here the outputs are different after 3-state transitions and hence states A and B are 3-distinguishable. In general, we can say that, if two states have a distinguishable sequence of length K, the states are said to be K-distinguishable. The concept of K-distinguishability leads directly to the definition of K-equivalence. States that are not K distinguishable are said to be K-equivalent.

#### 14.7 MINIMIZATION OF COMPLETELY SPECIFIED SEQUENTIAL MACHINES USING PARTITION TECHNIQUE

A procedure for the state minimization and determination of  $n$  equivalence is discussed in the following example. Consider the same state table given in Table 14.2.

*Step 1.* Partition the states into subsets such that all states in the same subset are 1-equivalent.

The first partition  $P_1$  can be obtained by placing those states having the same outputs under all inputs, in the same block. This partitioning gives two subsets.

1. (A, C, E): Their outputs under 0 and 1 inputs are 0 and 0 respectively.

2. (B, D, F): Their outputs under 0 and 1 inputs are 1 and 0 respectively.

∴

$$P_1 = (A, C, E)(B, D, F)$$

*Step 2.* Partition the states into subsets such that all states in the same subset are 2-equivalent.

This can be accomplished by observing that two states are 2-equivalent if and only if they are 1-equivalent and their  $I_i$ -successors of all possible  $I_i$  are also 1-equivalent. In other words, we can say that, two states are placed in the same block of partition  $P_2$  if and only if they are in the same block of  $P_1$  and for each possible  $I_i$ , their  $I_i$  successors are also contained in the same block of  $P_1$ .

1. The 0-successors of (A, C, E) are (C, E, D): They are in different blocks of  $P_1$ . So the block (A, C, E) must be split into (A, C) and (E).

2. The 1-successors of (B, D, F) are (F, E, B): They are in different blocks of  $P_1$ . Therefore, (B, D, F) must be split into (B, F) and (D).

∴

$$P_2 = (A, C)(E)(B, F)(D)$$

3. The 1-successors of (A, C, E) are (F, B, B): They are in the same block of  $P_1$ . The 0-successors of (B, D, F) are (D, B, D). They are in the same block of  $P_1$ . So, no partitioning is possible.

*Step 3.* Partition the states into subsets such that all states in the same block are 3-equivalent. For this, consider the states which are 2-equivalent, i.e. blocks in  $P_2$ .

1. The 0-successors of (A, C) are (C, E): They are in different blocks of  $P_2$ . So partition (A, C) into (A) and (C).

2. The 1-successors of (A, C) are (F, B): They are in the same block of  $P_2$ .

∴

$$P_3 = (A)(C)(E)(B, F)(D)$$

Further partitioning of states is not possible because we find that the 0- and 1-successors of (B, F), i.e. (D, D) and (B, F) are in the same block of  $P_3$ .

The states in the same blocks of  $P_3$  are equivalent. So states B and F are equivalent. One of them is redundant and can be eliminated. Let us remove F and replace F by B in the other places in the table. The minimized state table is shown in Table 14.3.

**Table 14.3** Minimized state table

PS	NS, Z	
	X = 0	X = 1
A	C, 0	F, 0
B	D, 1	F, 0
C	E, 0	B, 0
D	B, 1	E, 0
E	D, 0	B, 0

In general, we can say that the  $P_{k+1}$  partition is obtained from the  $P_k$  partition by placing in the same block of  $P_{k+1}$ , those states which are in the same block of  $P_k$  and whose  $I_i$  successors for every possible  $I_i$  are also in a common block of  $P_k$ .

When  $P_{k+1} = P_k$ , the partitioning process terminates and  $P_k$  defines the sets of equivalent states of the sequential machine. The  $P_k$  is thus called the *equivalence partition* and the partitioning procedure discussed above is referred to as the *Moore reduction procedure*.

*Note:*

1. The equivalent partition is unique.
2. If two states,  $S_i$  and  $S_j$  of sequential machine M are distinguishable, then they are distinguishable by a sequence of length  $n-1$  or less, where  $n$  is the number of states in M.

**Machine equivalence:** Two machines,  $M_1$  and  $M_2$  are said to be equivalent if and only if for every state in  $M_1$ , there is a corresponding equivalent state in  $M_2$  and vice versa.

**EXAMPLE 14.1** For the machine given in Table 14.4, find the equivalence partition and a corresponding reduced machine in standard form and also explain the procedure.

**Table 14.4** Example 14.1: State table

PS	NS, Z	
	X = 0	X = 1
A	B, 0	E, 0
B	E, 0	D, 0
C	D, 1	A, 0
D	C, 1	E, 0
E	B, 0	D, 0

### *Solution*

1. For equivalence partition, group the states having the same output under all input conditions (i.e. for  $X = 0$  and  $X = 1$ ) into blocks.

In the given table, states (A, B, E) and states (C, D) have same outputs under all input conditions.

$$\therefore P_1 = (A, B, E) (C, D)$$

2. See whether the 0 and 1-successors of states in each block of  $P_1$  are in the same block of  $P_1$  or not. If they are in different blocks partition the states.

Here the 0-successors of (A, B, E), i.e. (B, E, B) are in the same block, but 1-successors of (A, B, E), i.e. (E, D, D) are in different blocks of  $P_1$ . So, partition (A, B, E) into (A) and (B, E). 0-successors of (C, D), i.e. (D, C) are in the same block. Also 1-successors of (C, D), i.e. (A, E) are in the same block. So, no partitioning is possible.

$$\therefore P_2 = (A) (B, E) (C, D)$$

3. See whether the 0 and 1-successors of states in each block of  $P_2$  are in the same blocks of  $P_2$  or not. If they are in different blocks of  $P_2$ , partition them.

Here, the 0- and 1-successors of (B, E), i.e. (E, B) and (D, D) are in same blocks of  $P_2$ . The 0-successors of (C, D), i.e. (D, C) are also in one block, but the 1-successors of (C, D), i.e. (A, E) are in different blocks. So, partition (C, D) into (C) and (D).

$$\therefore P_3 = (A) (B, E) (C) (D)$$

4. See whether the 0 and 1-successors of (B, E) are in same blocks of  $P_3$ . If they are in different blocks of  $P_3$  partition them.

The 0 and 1-successors of (B, E), i.e. (E, B) and (D, D) are in the same blocks of  $P_3$ . So, no further partitioning is possible.

$$\therefore P_4 = (A) (B, E) (C) (D)$$

Thus, equivalent states are

$$B = E$$

So, state E is redundant and can be removed. Also state E can be replaced by state B in the table. A corresponding reduced machine in standard form is shown in Table 14.5.

**Table 14.5** Example 14.1: Reduced state table

PS	NS, Z	
	X = 0	X = 1
A	B, 0	B, 0
B	B, 0	D, 0
C	D, 1	A, 0
D	C, 1	B, 0

### EXAMPLE 14.2

- (a) Explain the limitations of finite state machines.  
 (b) Find the equivalence partition and a corresponding reduced machine in standard form for the machine given in Table 14.6.

**Table 14.6** Example 14.2: State table

PS	NS, Z	
	X = 0	X = 1
A	E, 0	D, 1
B	F, 0	D, 0
C	E, 0	B, 1
D	F, 0	B, 0
E	C, 0	F, 1
F	B, 0	C, 0

### Solution

1. States having the same output under all input conditions can be grouped as

$$P_1 = (A, C, E)(B, D, F)$$

2. The 0- and 1-successors of (A, C, E), i.e. (E, E, C) and (D, B, F) are in the same block of  $P_1$ . 0-successors of (B, D, F), i.e. (F, F, B) are also in the same block of  $P_1$ . So, no partitioning is required, but 1-successors of (B, D, F), i.e. (D, B, C) are in different blocks of  $P_1$ . So, partition (B, D, F) into (B, D) and (F).

$$\therefore P_2 = (A, C, E)(B, D)(F)$$

3. 0-successors of (A, C, E), i.e. (E, E, C) and the 0- and 1-successors of (B, D), i.e. (F, F) and (D, B) are in same blocks of  $P_2$ . So, no partitioning is required. The 1-successors of (A, C, E), i.e. (D, B, F) are in different blocks of  $P_2$ . So, partition (A, C, E) into (A, C) and (E).

$$\therefore P_3 = (A, C)(E)(B, D)(F)$$

4. The 0- and 1-successors of (A, C), and (B, D) i.e. (E, E), (D, B) and (F, F), (D, B) are in the same blocks of  $P_3$ . So, no partitioning is required.

$$\therefore P_4 = (A, C)(E)(B, D)(F)$$

Thus, equivalent states are

$$A = C \text{ and } B = D$$

So, states C and D are redundant and can be removed. C and D can be replaced by A and B respectively in the rest of the table. The resultant minimized state table is as shown in Table 14.7.

**Table 14.7** Example 14.2: Reduced state table

PS	NS, Z	
	X = 0	X = 1
A	E, 0	B, 1
B	F, 0	B, 0
E	A, 0	F, 1
F	B, 0	A, 0

**EXAMPLE 14.3** What are the conditions for two machines to be equivalent? For the machine given in Table 14.8, find the equivalence partition and a corresponding reduced machine in standard form.

**Table 14.8** Example 14.3: State table

PS	NS, Z	
	X = 0	X = 1
A	F, 0	B, 1
B	G, 0	A, 1
C	B, 0	C, 1
D	C, 0	B, 1
E	D, 0	A, 1
F	E, 1	F, 1
G	E, 1	G, 1

**Solution**

1. States having the same output under all input conditions can be grouped as

$$P_1 = (A, B, C, D, E)(F, G)$$

2. 1-successors of  $(A, B, C, D, E)$ , i.e.  $(B, A, C, B, A)$  and the 0- and 1-successors of  $(F, G)$ , i.e.  $(E, E)$ , and  $(F, G)$  are in the same blocks of  $P_1$ . So, no partitioning is required. The 0-successors of  $(A, B, C, D, E)$ , i.e.  $(F, G, B, C, D)$  are in different blocks of  $P_1$ . So, partition  $(A, B, C, D, E)$  into  $(A, B)$  and  $(C, D, E)$ .

$$\therefore P_2 = (A, B)(C, D, E)(F, G)$$

3. The 0- and 1-successors of  $(A, B)$  and  $(F, G)$ , i.e.  $(F, G)$ ,  $(B, A)$  and  $(E, E)$ ,  $(F, G)$  are in the same blocks of  $P_2$ . So, no partitioning is required.

The 1-successors of  $(C, D, E)$ , i.e.  $(C, B, A)$  are in different blocks of  $P_2$ . Also the 0-successors of  $(C, D, E)$ , i.e.  $(B, C, D)$  are in different blocks of  $P_2$ . So, partition  $(C, D, E)$  into  $(C)$  and  $(D, E)$ .

$$\therefore P_3 = (A, B)(C)(D, E)(F, G)$$

4. The 0- and 1-successors of  $(A, B)$  and  $(F, G)$ , i.e.  $(F, G)$ ,  $(B, A)$  and  $(E, E)$ ,  $(F, G)$  and the 1-successors of  $(D, E)$ , i.e.  $(B, A)$  are in the same blocks of  $P_3$ . So, no partitioning is possible. But the 0-successors of  $(D, E)$ , i.e.  $(C, D)$  are in different blocks of  $P_3$ . So, partition  $(D, E)$  into  $(D)$  and  $(E)$ .

$$\therefore P_4 = (A, B)(C)(D)(E)(F, G)$$

5. The 0- and 1-successors of  $(A, B)$  and  $(F, G)$ , i.e.  $(F, G)$ ,  $(B, A)$  and  $(E, E)$ ,  $(F, G)$  are in the same blocks of  $P_4$ . So, no further partitioning is possible.

Thus, equivalent states are

$$A = B \text{ and } F = G$$

So, states B and G are redundant and can be removed. In the rest of the table, B and G are replaced by A and F respectively. The resultant minimized state table is shown in Table 14.9.

**Table 14.9** Example 14.3: Reduced state table

PS	NS, Z	
	X = 0	X = 1
A	F, 0	A, 1
C	A, 0	C, 1
D	C, 0	A, 1
E	D, 0	A, 1
F	E, 1	F, 1

**EXAMPLE 14.4**

- (a) Define the state equivalence and machine equivalence with reference to sequential machines.  
 (b) Reduce the number of states in the state table given in Table 14.10, and tabulate the reduced state table and give proper assignment.

**Table 14.10** Example 14.4: State table

PS	NS, Z	
	X = 0	X = 1
A	F, 0	B, 0
B	D, 0	C, 0
C	F, 0	E, 0
D	G, 1	A, 0
E	D, 0	C, 0
F	F, 1	B, 1
G	G, 0	H, 0
H	G, 1	A, 0

**Solution**

1. States having the same output under all input conditions can be grouped as

$$P_1 = (A, B, C, E, G) (D, H) (F)$$

2. The 1-successors of (A, B, C, E, G), i.e. (B, C, E, C, H) are in different blocks. So, split (A, B, C, E, G) into (A, B, C, E) and (G). The 0-successors of (A, B, C, E, G), i.e. (F, D, F, D, G) are in different blocks. So, partition (A, B, C, E, G) into (A, C) (B, E) and (G).

$$\therefore P_2 = (A, C) (B, E) (G) (D, H) (F)$$

3. The 0-successors of (A, C), i.e. (F, F) and the 1-successors of (A, C), i.e. (B, E) are in the same blocks of  $P_2$ . The 0-successors of (B, E), i.e. (D, D) and the 1-successors of (D, H), i.e. (A, A) are also in the same block of  $P_2$ . So, no further partitioning is possible.

$$\therefore P_3 = (A, C) (B, E) (G) (D, H) (F)$$

Thus, the equivalent states are

$$A = C, B = E, \text{ and } D = H$$

So, states C, E and H are redundant and can be removed from the state table. Also states C, E and H can be replaced by A, B and D in the rest of the table. The resultant minimized state table is shown in Table 14.11.

**Table 14.11** Example 14.4: Reduced state table

PS	NS, Z	
	X = 0	X = 1
A	F, 0	B, 0
B	D, 0	A, 0
D	G, 1	A, 0
F	F, 1	B, 1
G	G, 0	D, 0

There are five states. So three state variables are required. The state assignment can be

$$A \rightarrow 000, B \rightarrow 001, D \rightarrow 011, F \rightarrow 101, G \rightarrow 111.$$

**EXAMPLE 14.5** For the machine described by Table 14.12 obtain

- The corresponding reduced machine table in standard form.
- Find a minimum length that distinguishes state A from state B.

**Table 14.12** Example 14.5: State table

PS	NS, Z	
	X = 0	X = 1
A	B, 1	H, 1
B	F, 1	D, 1
C	D, 0	E, 1
D	C, 0	F, 1
E	D, 1	C, 1
F	C, 1	C, 1
G	C, 1	D, 1
H	C, 0	A, 1

### Solution

(a)

- States having the same output under all input conditions are grouped as

$$P_1 = (A, B, E, F, G) (C, D, H)$$

- The 1-successors of (A, B, E, F, G), i.e. (H, D, C, C, D) are in the same block of  $P_1$ . Also the 0- and 1-successors of (C, D, H), i.e. (D, C, C) and (E, F, A) are in the same blocks of  $P_1$ . So, no partitioning is possible. The 0-successors of (A, B, E, F, G), i.e. (B, F, D, C, C) are in different blocks of  $P_1$ . So, partition (A, B, E, F, G) into (A, B) and (E, F, G).

$$\therefore P_2 = (A, B) (E, F, G) (C, D, H)$$

- The 1-successors of (A, B), i.e. (H, D) and the 0-successors of (C, D, H), i.e. (D, C, C) are in the same blocks of  $P_2$ . So, no partitioning is possible.

The 0-successors of (A, B), i.e. (B, F) are in different blocks of  $P_2$ . So, partition (A, B) into (A) and (B). The 1-successors of (C, D, H), i.e. (E, F, A) are in different blocks of  $P_2$ . So, partition (C, D, H) into (C, D) and (H).

$$\therefore P_3 = (A) (B) (E, F, G) (C, D) (H)$$

- The 0- and 1-successors of (E, F, G) and (C, D), i.e. (D, C, C), (C, C, D) and (D, C), (E, F) are in the same blocks of  $P_3$ . So, no further partitioning is possible.

$$\therefore P_4 = (A) (B) (E, F, G) (C, D) (H)$$

Thus, equivalent states are

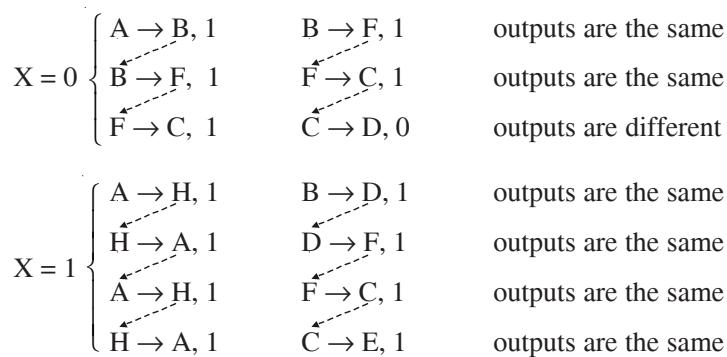
$$E = F = G \quad \text{and} \quad C = D$$

So, states F, G and D are redundant and can be removed. Also states F, G and D can be replaced by states E, E and C respectively in the remaining table. The corresponding reduced machine table in the standard form is shown in Table 14.13.

**Table 14.13** Example 14.5: Reduced state table

PS	NS, Z	
	X = 0	X = 1
A	B, 1	H, 1
B	E, 1	C, 1
C	C, 0	E, 1
E	C, 1	C, 1
H	C, 0	A, 1

(b) Distinguishability of states A and B:



So, the minimum length that distinguishes state A from state B is 3, i.e. states A and B are 3-distinguishable.

#### EXAMPLE 14.6

- (a) Obtain a minimal state table using partition technique for the state table shown in Table 14.14.
- (b) Find a minimal length sequence that distinguishes state  $q_1$  from state  $q_2$ .

**Table 14.14** Example 14.6: State table

PS	NS, Z	
	X = 0	X = 1
$q_1$	$q_2, 0$	$q_8, 1$
$q_2$	$q_6, 0$	$q_4, 1$
$q_3$	$q_4, 1$	$q_5, 0$
$q_4$	$q_3, 1$	$q_6, 0$
$q_5$	$q_4, 0$	$q_5, 1$
$q_6$	$q_3, 0$	$q_5, 1$
$q_7$	$q_3, 0$	$q_4, 1$
$q_8$	$q_3, 1$	$q_1, 0$

**Solution**

(a)

1. States having the same output under all input conditions can be grouped as

$$P_1 = (q_1, q_2, q_5, q_6, q_7) (q_3, q_4, q_8)$$

2. The 0-successors of  $(q_3, q_4, q_8)$ , i.e.  $(q_4, q_3, q_3)$  and the 1-successors of  $(q_3, q_4, q_8)$ , i.e.  $(q_5, q_6, q_1)$  are in the same blocks of  $P_1$ . So, no partitioning is required.

The 0-successors of  $(q_1, q_2, q_5, q_6, q_7)$ , i.e.  $(q_2, q_6, q_4, q_3, q_3)$  are in different blocks of  $P_1$ . So, partition  $(q_1, q_2, q_5, q_6, q_7)$  into  $(q_1, q_2)$  and  $(q_5, q_6, q_7)$ . The 1-successors of  $(q_1, q_2, q_5, q_6, q_7)$ , i.e.  $(q_8, q_4, q_5, q_5, q_4)$  are in different blocks. So, partition  $(q_1, q_2, q_5, q_6, q_7)$  into  $(q_1, q_2, q_7)$  and  $(q_5, q_6)$ .

$\therefore$

$$P_2 = (q_1, q_2) (q_5, q_6) (q_7) (q_3, q_4, q_8)$$

3. The 1-successors of  $(q_1, q_2)$ , i.e.  $(q_8, q_4)$  are in the same block of  $P_2$  but the 0-successors of  $(q_1, q_2)$ , i.e.  $(q_2, q_6)$  are in different blocks of  $P_2$ . So, partition  $(q_1, q_2)$  into  $(q_1)$  and  $(q_2)$ . The 0-successors of  $(q_5, q_6)$ , i.e.  $(q_4, q_3)$  are in the same block of  $P_2$ . Also the 1-successors of  $(q_5, q_6)$ , i.e.  $(q_5, q_5)$  are in the same block of  $P_2$ . So, no partitioning of  $(q_5, q_6)$  is possible. The 0-successors of  $(q_3, q_4, q_8)$ , i.e.  $(q_4, q_3, q_3)$  are in the same block of  $P_2$ . The 1-successors of  $(q_3, q_4, q_8)$ , i.e.  $(q_5, q_6, q_1)$  are in different blocks of  $P_2$ . So, partition  $(q_3, q_4, q_8)$  into  $(q_3, q_4)$  and  $(q_8)$ .

$\therefore$

$$P_3 = (q_1) (q_2) (q_5, q_6) (q_7) (q_3, q_4) (q_8)$$

4. The 0- and 1-successors of  $(q_5, q_6)$  i.e.  $(q_4, q_3)$  and  $(q_5, q_5)$ , and the 0- and 1-successors of  $(q_3, q_4)$ , i.e.  $(q_4, q_3)$  and  $(q_5, q_6)$  are in the same blocks of  $P_3$ . So, no further partitioning is possible.

$\therefore$

$$P_4 = (q_1) (q_2) (q_5, q_6) (q_7) (q_3, q_4) (q_8)$$

Thus, equivalent states are

$$q_5 = q_6 \text{ and } q_3 = q_4$$

So, states  $q_6$  and  $q_4$  are redundant and can be removed. In the remaining table  $q_6$  can be replaced by  $q_5$ , and  $q_4$  can be replaced by  $q_3$ . So, the resultant minimized state table is as shown in Table 14.15.

**Table 14.15** Example 14.6: Reduced state table

PS	NS, Z	
	X = 0	X = 1
$q_1$	$q_2, 0$	$q_8, 1$
$q_2$	$q_5, 0$	$q_3, 1$
$q_3$	$q_3, 1$	$q_5, 0$
$q_5$	$q_3, 0$	$q_5, 1$
$q_7$	$q_3, 0$	$q_3, 1$
$q_8$	$q_3, 1$	$q_1, 0$

- (b) The minimum length sequence that distinguishes state  $q_1$  from state  $q_2$  is determined as follows:

$X = 0$	$q_1 \rightarrow q_2, 0$	$q_2 \rightarrow q_6, 0$	outputs are the same
	$q_2 \rightarrow q_6, 0$	$q_6 \rightarrow q_3, 0$	outputs are the same
	$q_6 \rightarrow q_3, 0$	$q_3 \rightarrow q_4, 1$	outputs are different
$X = 1$	$q_1 \rightarrow q_8, 1$	$q_2 \rightarrow q_4, 1$	outputs are the same
	$q_8 \rightarrow q_1, 0$	$q_4 \rightarrow q_6, 0$	outputs are the same
	$q_1 \rightarrow q_8, 1$	$q_6 \rightarrow q_5, 1$	outputs are the same
	$q_8 \rightarrow q_1, 0$	$q_5 \rightarrow q_5, 1$	outputs are different

So, the minimum length of the sequence that distinguishes state  $q_1$  from state  $q_2$  is 3.

**EXAMPLE 14.7** Determine the minimal state equivalent of the state table shown in Table 14.16 using partition technique. Also determine the minimum length of sequence that distinguishes state B from state C.

**Table 14.16** Example 14.7: State table

PS	NS, Z	
	X = 0	X = 1
A	A, 0	E, 1
B	A, 1	E, 1
C	B, 1	F, 1
D	B, 1	F, 1
E	C, 0	G, 0
F	C, 0	G, 0
G	D, 0	H, 0
H	D, 0	H, 0

### Solution

- States having the same output under all input conditions can be grouped as

$$P_1 = (A) (B, C, D) (E, F, G, H)$$

- The 1-successors of (B, C, D), i.e. (E, F, F) are in the same block of  $P_1$ . Also, the 0- and 1-successors of (E, F, G, H), i.e. (C, C, D, D) and (G, G, H, H) are in the same blocks of  $P_1$ . So, no partitioning is possible.

The 0-successors of (B, C, D), i.e. (A, B, B) are in different blocks of  $P_1$ . So, partition (B, C, D) into (B) and (C, D).

$$\therefore P_2 = (A) (B) (C, D) (E, F, G, H)$$

- The 0- and 1-successors of (C, D), i.e. (B, B) and (F, F) are in the same blocks of  $P_2$ . Also, the 0- and 1-successors of (E, F, G, H), i.e. (C, C, D, D) and (G, G, H, H) are in the same blocks of  $P_2$ . So, no further partitioning is possible.

$$\therefore P_3 = (A) (B) (C, D) (E, F, G, H)$$

Thus, the equivalent states are

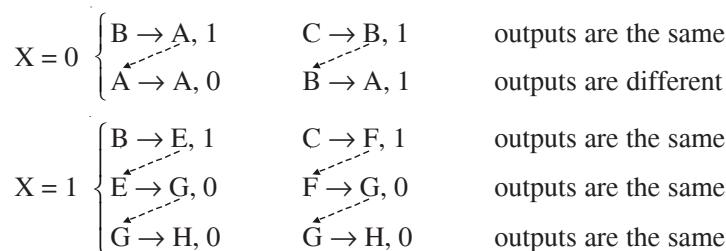
$$C = D \quad \text{and} \quad E = F = G = H$$

So, state D and states F, G, H are redundant and can be removed. In the reduced table, D may be replaced by C and F, G and H may be replaced by E. The resultant minimized state table is shown in Table 14.17.

**Table 14.17** Example 14.7: Reduced state table

PS	NS, Z	
	X = 0	X = 1
A	A, 0	E, 1
B	A, 1	E, 1
C	B, 1	E, 1
E	C, 0	E, 0

The minimum length of sequence:



So, the minimum length of the sequence that distinguishes state B from state C is 2.

**EXAMPLE 14.8** Determine a minimal state table equivalent to the state table given in Table 14.18 using the partition technique.

**Table 14.18** Example 14.8: State table

PS	NS, Z	
	X = 0	X = 1
S <sub>1</sub>	S <sub>1</sub> , 1	S <sub>1</sub> , 0
S <sub>2</sub>	S <sub>1</sub> , 1	S <sub>6</sub> , 1
S <sub>3</sub>	S <sub>2</sub> , 0	S <sub>5</sub> , 0
S <sub>4</sub>	S <sub>1</sub> , 0	S <sub>7</sub> , 0
S <sub>5</sub>	S <sub>4</sub> , 1	S <sub>3</sub> , 1
S <sub>6</sub>	S <sub>2</sub> , 0	S <sub>5</sub> , 0
S <sub>7</sub>	S <sub>4</sub> , 1	S <sub>3</sub> , 1

**Solution**

1. States having the same output under all input conditions can be grouped as

$$P_1 = (S_1) (S_2, S_5, S_7) (S_3, S_4, S_6)$$

2. The 1-successors of  $(S_2, S_5, S_7)$ , i.e.  $(S_6, S_3, S_3)$  are in the same block of  $P_1$ . Also the 1-successors of  $(S_3, S_4, S_6)$ , i.e.  $(S_5, S_7, S_5)$  are in the same block of  $P_1$ . So, no partitioning is required.

The 0-successors of  $(S_2, S_5, S_7)$ , i.e.  $(S_1, S_4, S_4)$  are in different blocks of  $P_1$ . So, partition  $(S_2, S_5, S_7)$  into  $(S_2)$  and  $(S_5, S_7)$ . Also, the 0-successors of  $(S_3, S_4, S_6)$ , i.e.  $(S_2, S_1, S_2)$  are in different blocks of  $P_1$ . So, split  $(S_3, S_4, S_6)$  into  $(S_4)$  and  $(S_3, S_6)$ .

∴

$$P_2 = (S_1) (S_2) (S_5, S_7) (S_4) (S_3, S_6)$$

3. The 0- and 1-successors of  $(S_5, S_7)$  and  $(S_3, S_6)$ , i.e.  $(S_4, S_4)$ ,  $(S_3, S_3)$  and  $(S_2, S_2)$ ,  $(S_5, S_5)$  are in the same blocks of  $P_2$ . So, no further partitioning is possible.

∴

$$P_3 = (S_1) (S_2) (S_5, S_7) (S_4) (S_3, S_6)$$

Thus, the equivalent states are

$$S_5 = S_7 \text{ and } S_3 = S_6$$

So, states  $S_7$  and  $S_6$  are redundant and can be removed. Also,  $S_7$  can be replaced by  $S_5$  and  $S_6$  can be replaced by  $S_3$  in the remaining table. The resultant minimized state table is shown in Table 14.19.

**Table 14.19** Example 14.8: Reduced state table

PS	NS, Z	
	X = 0	X = 1
$S_1$	$S_1, 1$	$S_1, 0$
$S_2$	$S_1, 1$	$S_3, 1$
$S_3$	$S_2, 0$	$S_5, 0$
$S_4$	$S_1, 0$	$S_5, 0$
$S_5$	$S_4, 1$	$S_3, 1$

## 14.8 SIMPLIFICATION OF INCOMPLETELY SPECIFIED MACHINES

In designing combinational logic circuits, we often encountered situations where the truth table was incompletely specified, which resulted in don't care terms. The same thing can happen in sequential circuits if the state transitions or output variables are not completely specified. The machines in which the state transitions or output variables are not completely specified are called incompletely specified machines.

When the state transitions are not specified, the sequential machine is not predictable. It is desirable to avoid such situations either by specifying the input sequence so that no next state is unspecified or by assigning next states that are not contrary to the desired results. This is illustrated in Table 14.20 and Table 14.21. Table 14.20 gives the incomplete description of a sequential machine. However Table 14.21 adds state T to describe specified behaviour of the sequential

machine in which all state transitions are specified and only the outputs are partially specified. Unspecified outputs can be assigned without any effect on the state sequence. However, it is advantageous to leave outputs unspecified as long as possible during the state reduction process. This provides additional flexibility in the state reduction process.

**Table 14.20** Incompletely specified state table

PS	NS, Z	
	X = 0	X = 1
S <sub>1</sub>	S <sub>3</sub> , 0	–
S <sub>2</sub>	–, 1	S <sub>3</sub> , 0
S <sub>3</sub>	S <sub>1</sub> , 0	S <sub>2</sub> , 1

**Table 14.21** State table

PS	NS, Z	
	X = 0	X = 1
S <sub>1</sub>	S <sub>3</sub> , 0	T, –
S <sub>2</sub>	T, 1	S <sub>3</sub> , 0
S <sub>3</sub>	S <sub>1</sub> , 0	S <sub>2</sub> , 1
T	T, –	T, –

When reducing incompletely specified state table we use the term state compatibility instead of state equivalence. The state compatibility is defined as:

States  $S_i$  and  $S_j$  are said to be compatible states, if and only if for every input sequence that affects the two states, the same output sequence occurs whenever both the outputs are specified and regardless of whether  $S_i$  or  $S_j$  is the initial state.

Consider the state table shown in Table 14.22. In this table the following compatible states are there.

1.  $S_0 = S_3$ , if the output Z for  $S_3$  when  $X = 0$  and  $X = 1$  is changed from a don't care to a 0.
2.  $S_2 = S_4$ , if the output Z for  $S_2$  and  $S_4$  when  $X = 0$  is changed from a don't care to either a 0 or a 1.

**Table 14.22** State table

PS	NS, Z	
	X = 0	X = 1
S <sub>0</sub>	S <sub>1</sub> , 0	S <sub>2</sub> , 0
S <sub>1</sub>	S <sub>0</sub> , –	S <sub>2</sub> , 0
S <sub>2</sub>	S <sub>3</sub> , –	S <sub>4</sub> , 1
S <sub>3</sub>	S <sub>1</sub> , –	S <sub>2</sub> , –
S <sub>4</sub>	S <sub>2</sub> , –	S <sub>4</sub> , 1

## 14.9 MERGER CHART METHODS

### 14.9.1 Merger Graphs

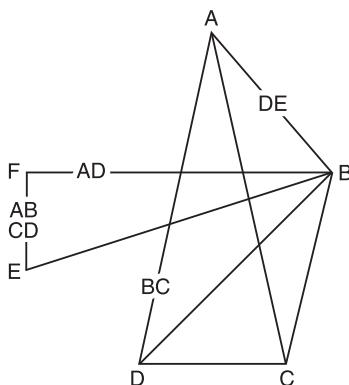
The merger graph is a state reducing tool used to reduce states in the incompletely specified machine. The merger graph is defined as follows.

1. Each state in the state table is represented by a vertex in the merger graph. So it contains the same number of vertices as the state table contains states.
2. Each compatible state pair is indicated by an unbroken line drawn between the two state vertices.
3. Every potentially compatible state pair with non-conflicting outputs but with different next states is connected by a broken line. The implied states are written in the line break between the two potentially compatible states.
4. If two states are incompatible no connecting line is drawn.

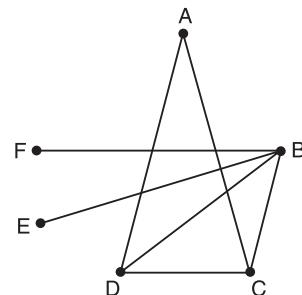
Consider a state table of an incompletely specified machine shown in Table 14.23. The corresponding merger graph shown in Figure 14.10a is drawn like this:

**Table 14.23** State table

PS	NS, Z			
	I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	I <sub>4</sub>
A	-	E, 1	B, 1	-
B	-	D, 1	-	F, 1
C	F, 1	-	-	-
D	-	-	C, 1	-
E	C, 0	-	A, 0	F, 1
F	D, 0	A, 1	B, 0	-



(a) Merger graph



(b) Simplified merger graph

**Figure 14.10** Merger graphs.

States A and B have non-conflicting outputs, but the successors (next states) under input I<sub>2</sub> are compatible only if implied states D and E are compatible. So, draw a broken line from A to B

with DE written in between. States A and C are compatible because the next states and output entries of states A and C are not conflicting. Therefore, a line is drawn between nodes A and C. States A and D have non-conflicting outputs but the successors under input  $I_3$  are B and C. Hence join A and D by a broken line with BC entered in between. States A and E have conflicting outputs under input  $I_3$ . So states A and E are non-compatible and hence no line is drawn between A and E. States A and F also have conflicting outputs under input  $I_3$ . So states A and F are non-compatible and hence no line is drawn between A and F. In a similar way, the merger graph is drawn for all possible pairs of states. We can see that the merger graph displays all possible pairs of states and their implied pairs. We know that a pair of states is compatible only if its implied pair is compatible. Therefore, it is necessary to check whether the implied pairs are indeed compatible.

Two states are said to be incompatible if no line is drawn between them. If implied states are incompatible, they are crossed and the corresponding line is ignored. For example, implied states D and E are incompatible, so states A and B are also incompatible. Next, it is necessary to check whether the incompatibility of A and B does not invalidate any other broken line. Observe that states E and F also become incompatible because the implied pair AB is incompatible. The broken lines which remain in the graph after all the implied pairs have been verified to be compatible are regarded as complete lines.

After checking all possibilities of incompatibility, the merger graph gives the following seven compatible pairs.

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

These compatible pairs are further checked for further compatibility. For example, pairs (B, C) (B, D) (C, D) are compatible. So (B, C, D) is also compatible. Also pairs (A, C) (A, D) (C, D) are compatible. So, (A, C, D) is also compatible. In this way the entire set of compatibles of sequential machine can be generated from its compatible pairs.

To find the minimal set of compatibles for state reduction, it is useful to find what are called the maximal compatibles. A set of compatible state pairs is said to be maximal, if it is not completely covered by any other set of compatible state pairs. The maximum compatibles can be found by looking at the merger graph for polygons which are not contained within any higher order complete polygons. For example, in Figure 14.10b only triangles (A, C, D) and (B, C, D) are of higher order.

The set of maximal compatibles for this sequential machine is given as

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

**EXAMPLE 14.9** Draw the merger graph and obtain the set of maximal compatibles for the incompletely specified sequential machine whose state table is given in Table 14.24.

**Table 14.24** Example 14.9: State table

PS	NS, Z	
	$I_1$	$I_2$
A	E, 0	B, 0
B	F, 0	A, 0
C	E, -	C, 0
D	F, 1	D, 0
E	C, 1	C, 0
F	D, -	B, 0

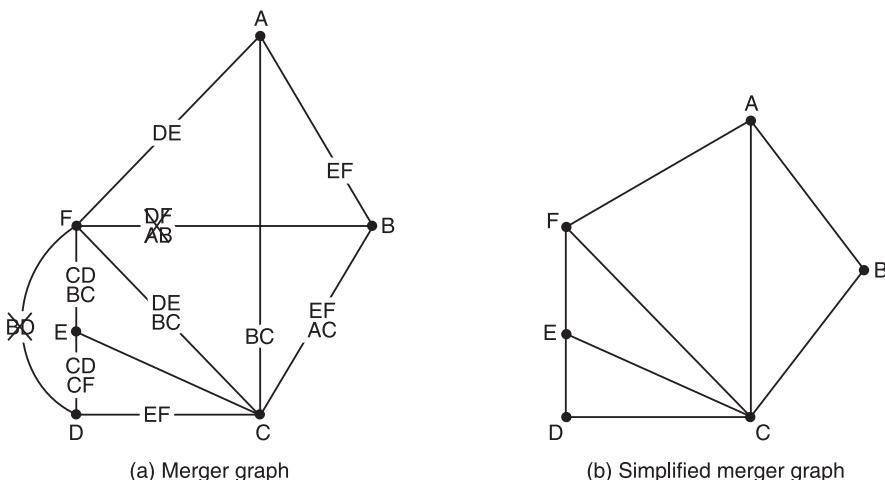
**Solution**

The merger graph corresponding to the given state table is drawn as shown in Figure 14.11a. In the merger graph shown states B and D are not connected. So the pair (B, D) is not compatible. Notice that the pair (D, F) is compatible only if the implied pair (B, D) is compatible. Since (B, D) is not compatible, (D, F) is also not compatible. Also notice that the pair (B, F) is compatible only if the implied pairs (D, F) and (A, B) are compatible. Since the pair (D, F) is not compatible, the pair (B, F) is also not compatible. Removing the broken lines corresponding to non-compatible pairs, i.e. (D, F) and (B, F) and replacing the broken lines of other pairs by unbroken lines (indicating that the corresponding pairs of states are compatible) the merger graph is redrawn as shown in Figure 14.11b.

After checking all possibilities of incompatibility, the merger graph gives the following nine compatible pairs:

$$\begin{aligned}
 & (A, B) (A, C) (A, F) (B, C) (C, D) (C, E) (C, F) (D, E) (E, F) \\
 & = (A, B) (A, C) (B, C) (A, F) (C, F) (C, D) (C, E) (D, E) (C, F) (E, F) \\
 & = (A, B, C) (A, C, F) (C, D, E) (C, E, F)
 \end{aligned}$$

Also looking at the merger graph in Figure 14.11b we can find the maximal compatibles corresponding to the triangles as (A, B, C) (A, C, F) (C, D, E) (C, E, F).



**Figure 14.11** Example 14.9.

### 14.9.2 Merger Table

The merger table method is also called Paull–Unger method or implication chart method. This method is more convenient than the merger graph method to find the compatible pairs and their implications while performing the state reduction of machines having a large number of states. Table 14.25 shows the state table for an incompletely specified sequential machine. Table 14.26 shows the corresponding merger table.

Each cell of the merger table corresponds to the compatible pair represented by the intersection of the row and column headings. The incompatibility of two states is indicated by placing a cross

mark  $\times$  in the corresponding cell. For example, states B and C are incompatible because their outputs are conflicting and hence the cell corresponding to them contains a cross mark  $\times$ . Similarly states B, E; D, E; E, F are incompatible. Hence put a  $\times$  mark in the corresponding cells. On the other hand, states A and B are compatible and hence the cell corresponding to them contains the check mark  $\checkmark$ . Similarly, cells corresponding to states A, D; A, E; A, G; B, G; C, F; D, F; D, G are also compatible. So a check mark is put in those cells also. The implied pairs or pairs corresponding to the state pair are written within the cell as shown in Table 14.26. For example, states A and C are compatible only when implied states E and F are compatible. Therefore, EF is written in the cell corresponding to states A and C. States C and E are compatible only when implied states A and B, and D and F are compatible. So AB and DF are written in the cell corresponding to states C and E. In a similar way, the entire merger table is written. Now it is necessary to check whether the implied pairs are compatible or not by observing the merger table. The implied states are incompatible if the corresponding cell contains a  $\times$ . For example, implied pair E, F is incompatible because cell EF contains a  $\times$ . Similarly, implied pairs EF, AF are incompatible because EF contains a  $\times$ . It is indicated by a  $\times$ .

**Table 14.25** State table

PS	NS, Z			
	00	01	11	10
A	E, 0	-	-	-
B	-	F, 1	E, 1	A, 1
C	F, 0	-	A, 0	F, 1
D	-	-	A, 1	-
E	-	C, 0	B, 0	D, 1
F	C, 0	C, 1	-	-
G	E, 0	-	-	A, 1

**Table 14.26** Merger table

	A	B	C	D	E	F
B	$\checkmark$					
C	$\times$		$\times$			
D	$\checkmark$	AE	$\times$			
E	$\checkmark$	$\times$	AB DF	$\times$		
F	CE	CF	$\checkmark$	$\checkmark$	$\times$	
G	$\checkmark$	$\checkmark$	EF <del>AF</del>	$\checkmark$	AD	CE

Once the merger table is completed, the set of all maximal compatibles can be formed by procedure which is the counterpart to that of finding a complete polygon in the merger graph. The procedure is as follows:

1. Begin with the rightmost column in the merger table and proceed left until a column containing a compatible pair is encountered. For example, in the table, the pair FG is the starting one.
2. Proceed left to the next column containing at least one compatible pair. If the state to which this column corresponds is compatible with all the states in the set of previously determined compatible states, then add this state to that set of compatible states to form a larger compatible. If this state is not compatible with all states of previously determined set, but is compatible with some of them and / or with some other state(s), form a new set of compatible states. For example, state E is not compatible with both F and G which are the previously determined set of states. However, it is compatible with state G. Therefore, it is not added to the previous set (FG) but a new set (EG) is formed. On the other hand, state D is compatible with states F and G of the previous set. So form a set (DFG) but D is not compatible with states E and G of the other previous set. So leave it as it is.
3. Repeat step 2 until the leftmost column is reached.

After application of the above mentioned procedure, the merger table in Table 14.26 gives the following set of maximal compatibles.

Column F: (F, G)

Column E: (F, G) (E, G)

Column D: (D, F, G) (E, G)

Column C: (C, E) (C, F) (E, G) (D, F, G)

Column B: (B, D, F, G) (C, E) (C, F) (E, G)

Column A: (A, B, D, F, G) (A, E) (C, E) (C, F) (C, G)

The complete procedure for the considered merger table is as follows:

The rightmost column is F. It indicates that states F and G are compatible, so, form a compatible set (F, G). Hence at column F we have (F, G). Column E indicates that state G is compatible only with state E. So include a new compatible set (E, G) in the list. So at column E we have (E, G) (F, G). Column D shows that states G and F are compatible with state D. Since (G, F) is already a previously formed compatible set, expand it into a bigger compatible set (D, F, G). So at column D, we have (D, F, G) (E, G) . Column C indicates that state C is compatible with states E and F. So form new compatible pairs (C, E) and (C, F). So at column C we have (C, E) (C, F) (D, F, G) (E, G). Column B indicates that state B is compatible with the states D, F, and G. Since we have a previously formed set (D, F, G), expand it into a compatible set (B, D, F, G). So at column B we have (B, D, F, G) (C, E) (C, F) (E, G). Column A indicates that state A is compatible with states B, D, E, F, G. Out of these, since we already have a compatible set (B, D, F, G), expand it into a new compatible set (A, B, D, F, G) and form another new set (A, E). So, at column A, we have (A, B, D, F, G) (A, E) (C, E) (C, F) (C, G) as the set of compatibles.

Therefore, the set of maximal compatibles is (A, B, D, F, G) (A, E) (C, E) (C, F) (C, G).

**EXAMPLE 14.10** Obtain the set of maximal compatibles for the sequential machine whose state table is shown in Table 14.27 using the merger table method.

**Table 14.27** Example 14.10: State table

PS	NS, Z			
	I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	I <sub>4</sub>
A	—	C, 1	E, 1	B, 1
B	E, 0	—	—	—
C	F, 0	F, 1	—	—
D	—	—	B, 1	—
E	—	F, 0	A, 0	D, 1
F	C, 0	—	B, 0	C, 1

**Table 14.28** Example 14.10: Merger table

B	✓				
C	CF	EF			
D	BE	✓	✓		
E	✗	✓	✗	✗	
F	✗	✗	✓	✗	
	A	B	C	D	E

**Solution**

Table 14.28 shows the merger table for the sequential machine whose state table is given in Table 14.27. A ✗ is put in the cells corresponding to pairs (A, E), (A, F), (C, E), (D, E), and (D, F) because they are non-compatible. A ✓ is put in cells corresponding to pairs (A, B), (B, D), (B, E), (C, D), and (C, F) because they are compatible. In other cells the implied pairs are written. Since pair (C, E) is not compatible put a ✗ in cell (B, F) which has this implied pair.

The merger table in Table 14.28 gives the following set of maximal compatibles.

Column E: (E, F)

Column D: (E, F)

Column C: (C, D) (C, F) (E, F)

Column B: (B, C, D) (B, E) (C, F) (E, F)

Column A: (A, B, C, D) (B, E) (C, F) (E, F)

The rightmost column is E. It indicates that state E is compatible with state F resulting in a compatible pair (E, F). Column D indicates that there is no compatible pair. So at column D also we have only (E, F). Column C indicates that state C is compatible with states D and F. So, add new pairs (C, D) and (C, F). So at column C, we have (C, D) (C, F) (E, F). Column B has 3 compatibilities. Since it is compatible with both C and D in the previous group, make a bigger group (B, C, D). Also, add a new compatible pair (B, E). So, at column B we have (B, C, D) (B, E) (C, F) (E, F). In column A, state A has compatibility with states (B, C, D).

Since (B, C, D) is already a group, we can form a bigger group (A, B, C, D). So at column A, we have (A, B, C, D) (B, E) (C, F) (E, F) as the set of maximal compatibles.

**EXAMPLE 14.11** Obtain the set of maximal compatibles for the state table given in Table 14.29 using the merger table method.

**Table 14.29** Example 14.11: State table

PS	NS, O/P			
	00	01	11	10
A	B, -	D, -	-	C, -
B	F, -	I, -	-	-
C	-	-	G, -	H, -
D	B, -	A, -	F, -	E, -
E	-	-	-	F, -
F	A, 0	-	B, -	-, 1
G	E, 1	B, -	-	-
H	E, -	-	-	A, 0
I	E, -	C, -	-	-

**Table 14.30** Example 14.11: Merger table

B	BF DI						
C	CH	✓					
D	DE	BF AF	FG EH				
E	CF	✓	X	EF			
F	AB	AF	BG	AB BF	✓		
G	BE BD	EF BI	✓	BE AB	✓	✗	
H	BE AC	EF	AH	BE AE	AF	✗	✓
I	BE CD	EF CI	✓	BE AC	✓	AE	BC
	A	B	C	D	E	F	G
							H

### Solution

The merger table for the sequential machine described by the state table of Table 14.29 is shown in Table 14.30. The maximal compatibles are obtained as follows:

Column H: (H, I)

Column G: (G, H, I)

Column F: (F, I) (G, H, I)

Column E: (E, G, H, I) (E, F, I)

Column D: (D, E, G, H, I) (D, E, F, I)

Column C: (D, E, G, H, I) (C, F) (C, G) (C, H) (D, E, F, I) (C, I)

Column B: (D, E, G, H, I) (B, C, I) (B, C, H) (B, C, G) (B, C, F) (D, E, F, I)

Column A: (D, E, G, H, I) (B, C, I) (A, B, C, H) (B, C, G) (D, F) (A, B, C, F) (D, E, F, I)

**EXAMPLE 14.12** Obtain the set of maximal compatibles for the sequential machine whose state table is given in Table 14.31 using (a) Merger table method and (b) Merger graph method.

**Table 14.31** Example 14.12: State table

PS	NS, Z		
	I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>
A	C, 0	E, 1	—
B	C, 0	E, —	—
C	B, —	C, 0	A, —
D	B, 0	C, 0	E, —
E	—	E, 0	A, —

**Table 14.32** Example 14.12: Merger table

B	✓		
C	✗	CE	
D	✗	BC CE	✗ AE
E	✗	✓	✓ ✗ AE CE
	A	B	C

### Solution

(a) *Merger table method:* The merger table for the given state table of Table 14.31 is shown in Table 14.32.

From the table we see that states (A, E) are not compatible. So cross all cells which contain AE. So states (D, E) and (C, D) are also not compatible. Therefore, the maximal compatibles are as follows:

Column D: NIL

Column C: (C, E)

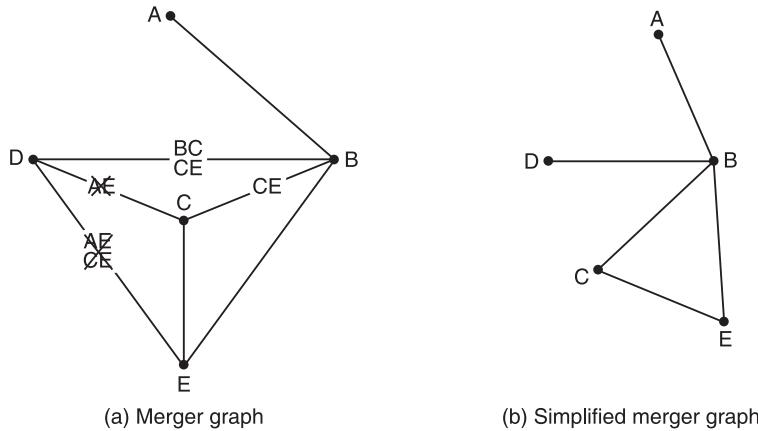
Column B: (B, C, E) (B, D)

Column A: (A, B) (B, C, E) (B, D)

Therefore, the set of maximal compatibles is (A, B) (B, C, E) (B, D).

(b) *Merger graph method:* The merger graph for the sequential machine corresponding to the given state table of Table 14.31 is shown in Figure 14.12a. From the merger graph we observe that states A and E are not connected. So they are not a compatible pair. So put a  $\times$  mark on all broken lines which have AE, i.e. cross the lines CD and DE. The rest of the graph can be drawn as shown in Figure 14.12b with unbroken lines indicating that those pairs are compatible. From the revised graph, we observe that the maximal set of compatibles is

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



**Figure 14.12** Example 14.12.

**EXAMPLE 14.13** Obtain the set of maximal compatibles for the sequential machine whose state table is given in Table 14.33 using (a) the merger table method and (b) merger chart method.

**Table 14.33** Example 14.13: State table

PS	NS, Z			
	00	01	11	10
A	C, 0	-	C, 0	-
B	A, -	B, 1	D, -	-
C	-	E, 1	-, 0	D, 0
D	E, 0	-	F, 1	C, -
E	F, 0	-	B, -	A, 1
F	-	B, 1	-, 0	C, 0

**Table 14.34** Example 14.13: Merger table

B	<del>AC</del> <del>CD</del>			
C	✓	<del>DE</del>		
D	✗	<del>AE</del> <del>DF</del>	✗	
E	<del>CF</del> <del>BC</del>	<del>AF</del> <del>BD</del>	✗	<del>EF</del> <del>BF</del> <del>AC</del>
F	✓	✓	<del>BE</del> <del>CD</del>	✗

**Solution**

(a) *Merger table method:* The merger table for the given state table of Table 14.33 is shown in Table 14.34. From the table we see that states (E, F) are not compatible. So states (D, E) are also not compatible. States (C, D) are not compatible. So states (C, F) and (A, B) are also not compatible. If (C, F) are not compatible, then states (A, E) are not compatible. States (D, F) are not compatible. So (B, D) are also not compatible. If states (B, D) are not compatible states, (B, E) are also not compatible. If states (B, E) are not compatible, then states (B, C) are also not compatible. So we are left with only compatible pairs (A, C) (A, F) and (B, F):

Column E: Nil

Column D: Nil

Column C: Nil

Column B: (B, F)

Column A: (A, C)(A, F)(B, F)

Therefore, the set of maximal compatibles is

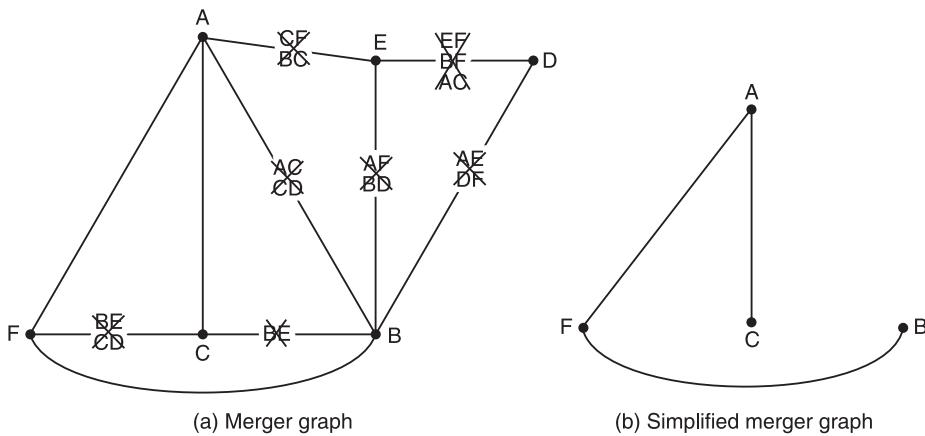
$$(A, C)(A, F)(B, F)$$

(b) *Merger graph method:* The merger graph for the sequential machine corresponding to the given state table of Table 14.33 is shown in Figure 14.13a. From the merger graph, we observe that E and F are not connected. So they are not a compatible pair. So put a ✗ mark on all broken lines which have EF, i.e. cross line DE. A and E are not connected. So cross line BD. Since line BD is crossed, cross line BE. Since line BE is crossed, cross lines BC and CF. So we are left with the reduced merger graph shown in Figure 14.13b. From the revised graph, we observe that the maximal set of compatibles is

$$(A, C)(A, F)(B, F)$$

## 14.10 CONCEPT OF MINIMAL COVER TABLE

In the previous sections we have seen how to use the merger graph and merger table for state reduction. Both the merger graph and merger table give us the maximal compatible pairs and we



**Figure 14.13** Example 14.13.

can cover (implement) the sequential machine using the number of states equal to the number of maximal compatible pairs. In this section, we discuss the compatibility graph which can be used to get minimal closed covering giving further reduction of states if possible.

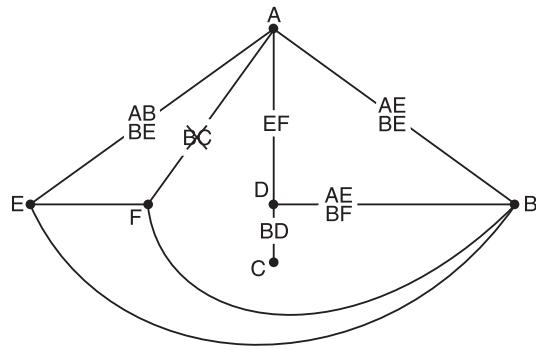
#### 14.10.1 Compatibility Graph

The compatibility graph is a directed graph with the number of vertices equal to the number of compatible pairs. Each of its vertices correspond to one compatible pair. A directed arc leads from vertex  $(S_i, S_j)$  to vertex  $(S_p, S_q)$  if and only if  $(S_i, S_j)$  implies  $(S_p, S_q)$ . The compatibility graph can be easily drawn from the merger graph or merger table.

**Steps to construct compatibility graph from merger graph:** Consider a sequential machine specified by the state table shown in Figure 14.14a. The merger graph for this machine is shown in Figure 14.14b.

PS	NS, Z			
	00	01	11	10
A	A, 0	—	E, —	B, 1
B	E, —	C, 1	B, —	—
C	—	B, 0	—, 1	D, 0
D	A, 0	—	F, 1	B, —
E	B, 0	—	B, 0	—
F	—	C, 1	—, 0	C, 1

(a) State table



**Figure 14.14** Sequential machine.

*Step 1. Identify and draw vertices:* Refer to the reduced merger graph shown in Figure 14.15a. Mark the vertices corresponding to all compatible pairs. In the merger graph, each compatible state pair is indicated by a line drawn between the two state vertices. For example, in the merger

graph shown in Figure 14.15a, there are lines between A and B, A and D, A and E, B and D, B and E, B and F, C and D, and E and F. This gives us eight vertices—AB, AD, AE, BD, BE, BF, CD, and EF in the compatibility graph shown in Figure 14.15b.

*Step 2. Draw arcs:* Draw the arcs lead from vertex  $(S_i, S_j)$  to vertex  $(S_p, S_q)$  if and only if the compatible pair  $(S_i, S_j)$  implies the pair  $(S_p, S_q)$ . The implied pairs are shown in the merger graph of Figure 14.14a. The compatible pair AB implies (AE) and (BE), AD implies (EF), AE implies (AB) and (BE), BD implies (AE) and (BF), CD implies (BD), and so on.

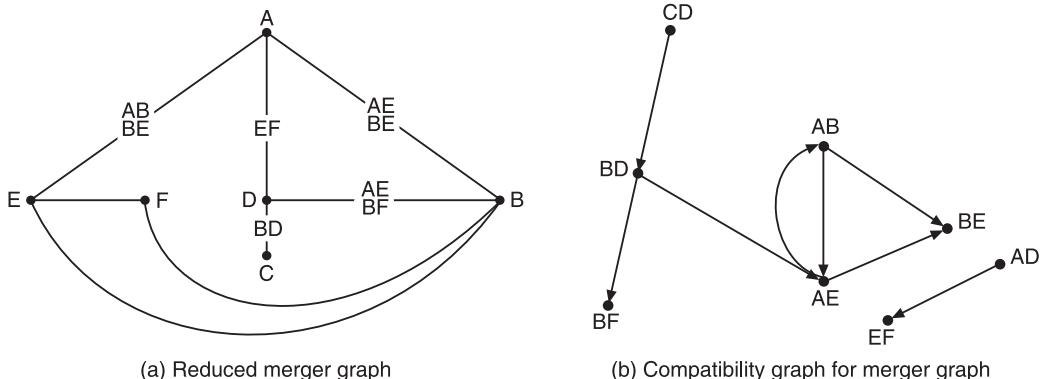


Figure 14.15 Reduced merger graph and compatibility graph.

**Steps to construct compatibility graph from merger table:** For the given state table of Figure 14.14a, first draw the merger table shown in Figure 14.16a.

*Step 1. Identify and draw vertices:* Refer to the merger table of the sequential machine shown in Figure 14.16a. Mark the vertices corresponding to all compatible pairs in the merger table. In the merger table, each cell of the table except those marked (x) corresponds to the compatible pair defined by the intersection of the row and column headings. If we observe the merger table from right to left and top to bottom the compatible pairs are EF, CD, BD, BE, BF, AB, AD, and AE. This gives us eight vertices in the compatibility graph as shown in Figure 14.16b.

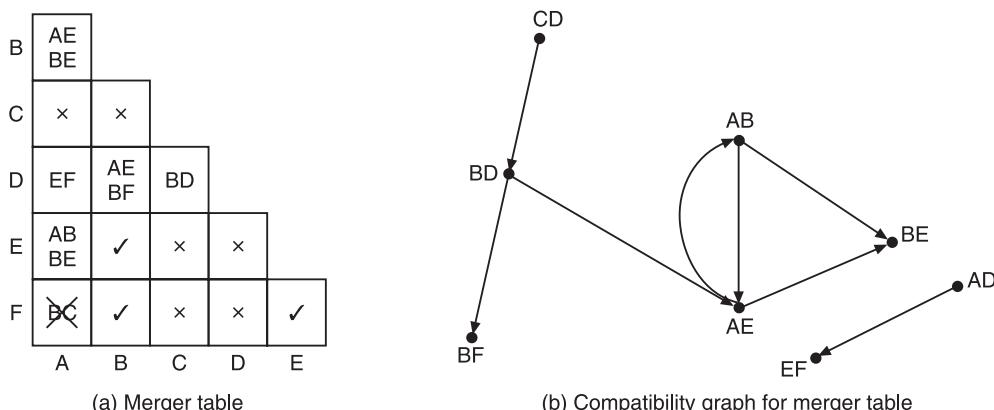


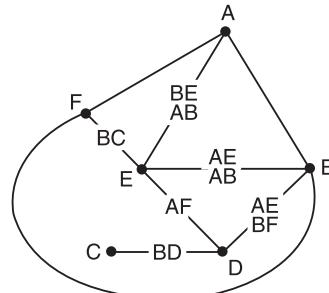
Figure 14.16 Merger table and compatibility graph from merger table.

**Step 2. Draw arcs:** Draw the arcs lead from vertex  $(S_i, S_j)$  to vertex  $(S_p, S_q)$  if and only if the compatible pair  $(S_i, S_j)$  implies  $(S_p, S_q)$ . The entries in cell  $(S_i, S_j)$  of the merger table are the pairs implied by  $(S_i, S_j)$ . For example, compatible pair CD implies BD, pair BD implies AE and BF, pair AB implies AE and BE, and so on.

**EXAMPLE 14.14** For the incompletely specified sequential machine whose state table is given in Figure 14.17a, obtain the compatibility graph from its (a) merger graph and (b) merger table.

PS	NS, Z			
	I <sub>0</sub>	I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>
A	B, 0	—	B, 0	—
B	A, —	C, 1	B, —	—
C	—	B, 0	—, 1	D, 0
D	E, 0	—	F, 1	B, —
E	E, 0	—	A, —	B, 1
F	—	C, 1	—, 0	C, 1

(a) State table



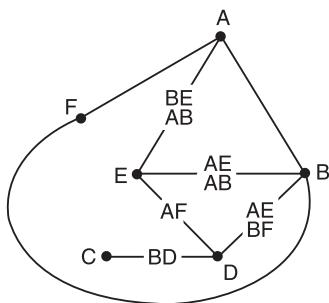
(b) Merger graph

Figure 14.17 Example 14.14.

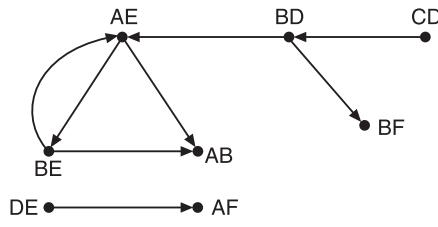
### Solution

(a) The merger graph of the sequential machine described by the given state table is shown in Figure 14.17b. The reduced merger graph is shown in Figure 14.18a. The compatibility graph based on the merger graph is shown in Figure 14.18b.

In the merger graph there are eight compatible pairs. So there are eight vertices—AB, AE, AF, BD, BE, BF, CD, and DE in the compatibility graph.



(a) Reduced merger graph



(b) Compatibility graph from merger graph

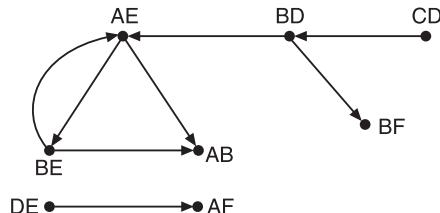
Figure 14.18 Example 14.14a.

(b) The merger table of the sequential machine described by the given state table is shown in Figure 14.19a. In the merger table there are eight compatible pairs. So there are eight vertices—AB, AE, AF, BD, BE, BF, CD, and DE in the compatibility graph.

The compatibility graph based on the merger table is shown in Figure 14.19b.

		✓		
B		x	x	
C	x			
D	x	AE BF	BD	
E	BE AB	AE AB	x	AF
F	✓	✓	x	x
	A	B	C	D
				E

(a) Merger table



(b) Compatibility graph from merger table

Figure 14.19 Example 14.14b.

#### 14.10.2 Subgraph of Compatibility Graph

Any part of a compatibility graph is called the subgraph of the compatibility graph. A subgraph of a compatibility graph is said to be closed if for every vertex in the subgraph, all outgoing arcs and their terminating vertices also belong to the subgraph. Each vertex in the subgraph belongs to one state. Such a subgraph forms a closed covering for the corresponding machine.

Figure 14.20 shows the subgraph in the compatibility graph. It includes three vertices AB, AE, and BE. Therefore, the corresponding sequential machine can be covered by a three-state machine and this is a minimal closed covering.

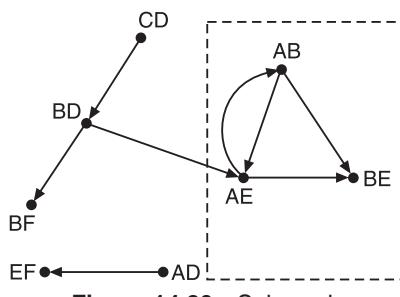


Figure 14.20 Subgraph.

The set of maximal compatibles derived from the merger graph given in Figure 14.18a contains four members  $\{(A, B, D), (A, B, E), (B, E, F), (C, D)\}$ . Therefore, the merger graph reduces the number of states required to implement the machine up to four. On the other hand, the same machine can be implemented using only three states as indicated by the subgraph of the compatibility graph.

#### 14.10.3 Minimal Cover Table

Once we obtain the minimal closed covering, then we have to derive the minimal cover table. This table consists of states of the minimal state machine (Table 14.36) corresponding to the sequential machine of Figure 14.14a reproduced as Table 14.35. For the case under consideration there are three states and we assign them as X, Y and Z. Therefore,

$$(AB) \rightarrow X \quad (AE) \rightarrow Y \quad (BE) \rightarrow Z$$

Using these assignments, we can write the minimal cover table as shown in Table 14.36.

**Table 14.35** State table

PS	NS, Z			
	I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	I <sub>4</sub>
A	A, 0	–	E, –	B, 1
B	E, –	C, 1	B, –	–
C	–	B, 0	–, 1	D, 0
D	A, 0	–	F, 1	B, –
E	B, 0	–	B, 0	–
F	–	C, 1	–, 0	C, 1

**Table 14.36** Minimal cover table

PS	NS, Z			
	I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	I <sub>4</sub>
(A, B) → X	Y, 0	–	Z, –	Z/X, 1
(A, E) → Y	X, 0	–	Z, 0	Z/X, 1
(B, E) → Z	Z, 0	–	Z/X, 0	–

**EXAMPLE 14.15** Construct the compatibility graph and obtain the minimal cover table for the sequential machine described by the state table given in Table 14.37.

**Table 14.37** Example 14.15: State table

PS	NS, Z	
	I <sub>1</sub>	I <sub>2</sub>
S <sub>1</sub>	S <sub>5</sub> , 1	S <sub>2</sub> , 1
S <sub>2</sub>	S <sub>6</sub> , 1	S <sub>1</sub> , 1
S <sub>3</sub>	S <sub>5</sub> , –	S <sub>3</sub> , 1
S <sub>4</sub>	S <sub>6</sub> , 0	S <sub>4</sub> , 1
S <sub>5</sub>	S <sub>3</sub> , 0	S <sub>3</sub> , 1
S <sub>6</sub>	S <sub>4</sub> , –	S <sub>2</sub> , 1

### **Solution**

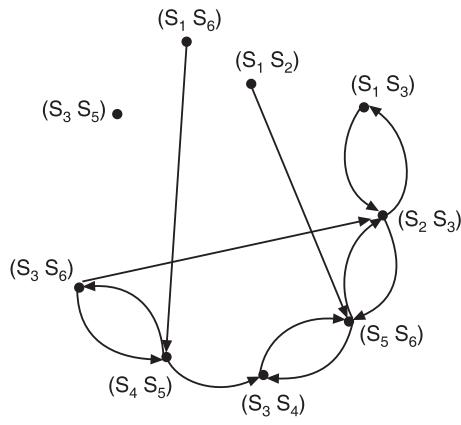
The merger table for the given state table is constructed as shown in Figure 14.21a. From the merger table the compatibility graph shown in Figure 14.21b can be constructed.

To construct the minimal cover table first we have to consider the closed subgraph. There may be more than one subgraph for the given compatibility graph. So, there may be more than one solution. One subgraph which can be easily visualized from the compatibility graph contains four vertices (S<sub>1</sub> S<sub>3</sub>), (S<sub>2</sub> S<sub>3</sub>), (S<sub>3</sub> S<sub>4</sub>), (S<sub>5</sub> S<sub>6</sub>) as shown in Figure 14.22a. It indicates

that the machine can be covered by a four state machine. This is one of the solutions. However, we should try to find a larger closed subgraph and check whether the added vertices can be used to merge compatible pairs to give larger compatibles. In the present case, if we add vertex  $(S_1 S_2)$  to the preceding subgraph as shown in Figure 14.22b, we obtain a set which consists of five compatible pairs  $\{(S_1 S_2), (S_1 S_3), (S_2 S_3), (S_3 S_4), (S_5 S_6)\}$  and is reduced to the closed covering  $\{(S_1 S_2, S_3), (S_3 S_4), (S_5 S_6)\}$ . Now there are three states and the machine can be covered by a three-state machine.

$S_2$	$S_5 S_6$				
$S_3$	$S_2 S_3$		$S_1 S_3$	$S_5 S_6$	
$S_4$	x	x		$S_5 S_6$	
$S_5$	x	x	✓	$S_3 S_4$	$S_3 S_6$
$S_6$	$S_4 S_5$	<del><math>S_4 S_2</math></del> <del><math>S_4 S_6</math></del>	$S_2 S_3$	$S_4 S_5$	<del><math>S_3 S_4</math></del>
$S_1$	$S_2$	$S_3$	$S_4$	$S_5$	

(a) Merger table



(b) Compatibility graph

Figure 14.21 Example 14.15.

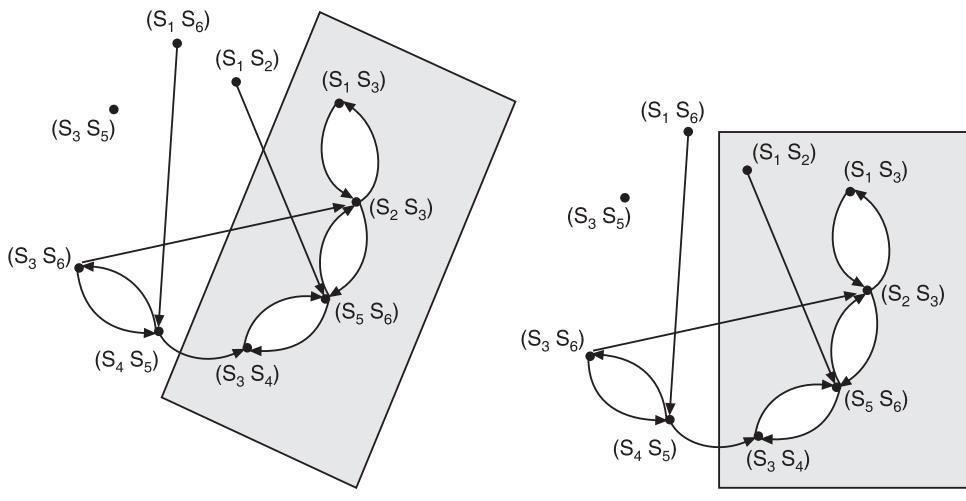


Figure 14.22 Example 14.15.

Now by assigning A, B, C such that  $(S_1 S_2 S_3) \rightarrow A$ ,  $(S_3 S_4) \rightarrow B$ ,  $(S_5 S_6) \rightarrow C$  we can write the minimal cover table as shown in Table 14.38.

**Table 14.38** Example 14.15: Minimal cover table

PS	NS, Z	
	I <sub>1</sub>	I <sub>2</sub>
(S <sub>1</sub> S <sub>2</sub> S <sub>3</sub> ) → A	C, 1	A, 1
(S <sub>3</sub> S <sub>4</sub> ) → B	C, 1	B, 1
(S <sub>5</sub> S <sub>6</sub> ) → C	B, 1	A, 0

**EXAMPLE 14.16** Construct the compatibility graph and obtain the minimal cover table for the sequential machine described by the state table given in Table 14.39.

**Table 14.39** Example 14.16: State table

PS	NS, Z	
	I <sub>1</sub>	I <sub>2</sub>
A	—	F, 0
B	B, 0	C, 0
C	E, 0	A, 1
D	B, 0	D, 0
E	F, 1	D, 0
F	A, 0	—

**Table 14.40** Example 14.16: Merger table

B	CF				
C	x	x			
D	DF	✗	x		
E	DF	x	x	x	
F	✓	AB	AE	AB	x
	A	B	C	D	E

### Solution

The merger table for the given state table is constructed as shown in Table 14.40. From the merger table the compatibility graph shown in Figure 14.23a may be constructed. Looking at the compatibility graph we can consider the closed subgraph shown in Figure 14.23b.

From the closed covering we can derive the minimal cover table as follows: There are four states (A B), (A E), (C F), (D F) in the subgraph. Assign them as

$$(A B) \rightarrow P, \quad (A E) \rightarrow Q, \quad (C F) \rightarrow R, \quad (D F) \rightarrow S$$

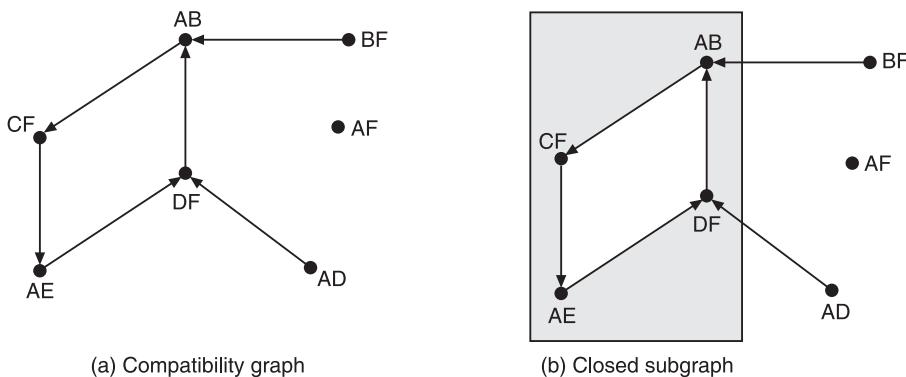


Figure 14.23 Example 14.16.

Using these assignments we can write the minimal cover table as shown in Table 14.41.

Table 14.41 Example 14.16: Minimal cover table

PS	NS, Z	
	I <sub>1</sub>	I <sub>2</sub>
AB → P	P, 0	R, 0
AE → Q	R/S, 1	S, 0
CF → R	Q, 0	P/Q, 1
DF → S	P, 0	S, 0

### SHORT QUESTIONS AND ANSWERS

1. By how many models are synchronous sequential circuits represented? Name them.  
A. Synchronous or clocked sequential circuits are represented by two models. They are: 1. Moore circuit (or model) and 2. Mealy circuit (or model).
2. What is a sequential machine?  
A. A sequential machine is another name of a sequential circuit.
3. What is the Mealy machine?  
A. The Mealy machine (or model or circuit) is a sequential circuit in which the output depends on both the present state of the flip-flops and on the inputs.
4. What is the Moore machine?  
A. The Moore machine (or model or circuit) is a sequential circuit in which the output depends only on the present state of the flip-flops.
5. Compare the Moore and Mealy machines?  
A. The Moore and Mealy machines are compared as follows.

Moore machine	Mealy machine
1. Its output is a function of present state only. $z(t) = g\{S(t)\}$	1. Its output is a function of present state as well as present input. $z(t) = g\{S(t), x(t)\}$
2. Input changes do not affect the output.	2. Input changes may affect the output of the circuit.
3. It requires more number of states for implementing same function.	3. It requires less number of states for implementing same function.

**6. What are the capabilities and limitations of finite state machines?**

A. The capabilities and limitations of finite state machines are as follows:

1. With an  $n$ -state machine, we can generate a periodic sequence of  $n$  states or less than  $n$  states.
2. Certain infinite sequences cannot be produced by a finite state machine.
3. The finite state machine has limited memory and due to limited memory it cannot produce certain outputs.

**7. What do you mean by a successor?**

A. If an input sequence  $X$  takes a machine from state  $S_i$  to state  $S_j$ , then  $S_j$  is said to be the  $X$ -successor of  $S_i$ .

**8. What do you mean by a terminal state?**

A. A terminal state is a state with no incoming arcs which start from other states and terminate on it.

**9. What is a strongly connected machine?**

A. A sequential machine  $M$  is said to be strongly connected, if for every pair of states  $S_i, S_j$  of the sequential machine, there exists an input sequence which takes the machine  $M$  from  $S_i$  to  $S_j$ .

**10. What are redundant states?**

A. Redundant states are states whose functions can be accomplished by other states.

**11. What are equivalent states?**

A. Two states are said to be equivalent if for every possible set of inputs they generate exactly the same output and the same next state.

**12. What is the advantage of having equivalent states?**

A. When equivalent states are there, one of them can be retained and all others can be removed without altering the input-output relationship because they are redundant. This results in reduction of states which in turn reduces the number of required flip-flops and logic gates reducing the cost of the final circuit.

**13. State 'state equivalence theorem'.**

A. The state equivalence theorem states that two states  $S_A$  and  $S_B$  of a sequential machine are equivalent if and only if for every possible input sequence  $X$ , the outputs are the same and the next states are equivalent. That is, if  $S_A(t+1) = S_B(t+1)$  and  $Z_A = Z_B$ , then  $S_A = S_B$ .

**14. What is a distinguishing sequence?**

A. An input sequence which distinguishes two states is called a distinguishing sequence.

**15. What are distinguishable states?**

A. Two states  $S_A$  and  $S_B$  of a sequential machine are distinguishable if and only if there exists at least one finite input sequence which when applied to the sequential machine causes different output sequences depending on whether  $S_A$  or  $S_B$  is the initial state.

**16.** What are K-distinguishable states?

A. Two states are said to be K-distinguishable, if they have a distinguishable sequence of length K.

**17.** Write the Moore reduction procedure for minimization of completely specified sequential machines using the partition technique.

A. The procedure for state minimization and determination of  $n$ -equivalence using the partition technique is:

*Step 1.* Partition the states into subsets such that all states in the same subset are 1-equivalent.

*Step 2.* Partition the states into subsets such that all states in the same subset are 2-equivalent.

*Step 3.* Partition the states into subsets such that all states in the same subset are 3-equivalent and so on till further partitioning of states is not possible. The last partition called the equivalence partition defines the sets of equivalent states of the sequential machine.

**18.** What is machine equivalence?

A. Two machines  $M_1$  and  $M_2$  are said to be equivalent if and only if for every state in  $M_1$ , there is a corresponding equivalent state in  $M_2$  and vice versa.

**19.** What are incompletely specified machines?

A. The sequential circuits whose state transitions or output variables are not completely specified are called incompletely specified machines.

**20.** What is the advantage of unspecified outputs?

A. The advantage of leaving outputs unspecified as long as possible during the state reduction process is, this provides additional flexibility in state reduction process.

**21.** Define state compatibility.

A. Two states  $S_i$  and  $S_j$  of a sequential machine are said to be compatible states, if and only if for every input sequence that affects the two states, the same output sequence occurs whenever both outputs are specified and regardless of whether  $S_i$  or  $S_j$  is the initial state.

**22.** What is a merger graph?

A. The merger graph is a state reducing tool used to reduce states in the incompletely specified machine. The merger graph is defined as follows:

1. It contains the same number of vertices as the state table contains states.
2. Each compatible state pair is indicated by an unbroken line drawn between the two state vertices.
3. Every potentially compatible state pair with outputs not in conflict but whose next states are different is connected by a broken line. The implied states are written in the line break between the two potentially compatible states.
4. If two states are incompatible, then no connecting line is drawn.

**23.** What are maximal compatibles?

A. A set of compatible state pairs which is not completely covered by any other set of compatible state pairs is called a set of maximal compatibles.

**24.** What are the other names of the merger table method?

A. The merger table method is also called the Paull–Unger method or implication chart method.

**25.** What is a merger table?

A. A merger table is a table in which each cell shows the compatible pairs and their implications.

**26.** How do you obtain the set of all maximal compatibles from the merger table?

A. From the merger table, the set of all maximal compatibles is obtained as follows:

1. Begin with the rightmost column in the merger table and proceed left until a column containing a compatible pair is encountered. Write the set of all compatible pairs in this column.

2. Proceed left to the next column containing at least one compatible pair. If the state to which this column corresponds is compatible with all the states in the set of previously determined compatible states, then add this state to that set of compatible states to form a larger compatible. If the state is not compatible with all the states of previously determined set, but is compatible with some other state, form a new set of compatible states.
  3. Repeat step 2 until the leftmost column is reached. The sets in the leftmost column give the set of maximal compatibles.
- 27.** What is compatibility graph?
- A. The compatibility graph is a directed graph whose vertices correspond to all compatible pairs, and an arc leads from vertex  $(S_i, S_j)$  to vertex  $(S_p, S_q)$  if and only if  $(S_i, S_j)$  implies  $(S_p, S_q)$ . The compatibility graph can be easily drawn from the merger graph or merger table.
- 28.** What is subgraph of a compatibility graph?
- A. Any part of a compatibility graph is called the subgraph of the compatibility graph.
- 29.** What is a closed subgraph?
- A. A subgraph of a compatibility graph is said to be closed if for every vertex in the subgraph all outgoing arcs and their terminating vertices also belong to the subgraph. Each vertex in the subgraph belongs to one state. Such a subgraph forms a closed covering for the corresponding machine.
- 30.** What is a minimal cover table?
- A. A minimal cover table is a table which consists of the states of a minimal state machine.

### REVIEW QUESTIONS

1. What are the Moore and Mealy machines? Compare them.
2. What are the capabilities and limitations of finite state machines?
3. Define: Successor, terminal state, strongly connected machine, state equivalence, machine equivalence, compatibility graph, subgraph of compatibility graph, minimal cover table, merger graph, merger table.
4. Explain the procedure of state minimization using the partition technique.
5. Explain the procedure of state minimization using the merger graph and merger table.

### FILL IN THE BLANKS

1. The sequential circuit in which the output depends only on the present state of the flip-flops is called a \_\_\_\_\_ circuit.
2. The sequential circuit in which the output depends on the present state of the flip-flops as well as on the present inputs is called a \_\_\_\_\_ circuit.
3. A sequential machine is another name of \_\_\_\_\_.
4. A state which has no outgoing arcs is called a \_\_\_\_\_ state.
5. If the outputs of two states are different after P-state transitions, they are said to be \_\_\_\_\_.

6. \_\_\_\_\_ outputs provide additional flexibility in state reduction.
7. The merger table method of state reduction is also called \_\_\_\_\_ method or \_\_\_\_\_ method.
8. The compatibility graph can be easily drawn from the \_\_\_\_\_ or \_\_\_\_\_.
9. Each vertex in the subgraph belongs to \_\_\_\_\_.
10. A table which consists of the states of a minimal state machine is called a \_\_\_\_\_.

### PROBLEMS

- 14.1** For the state tables of the machines given below, find the equivalence partition and a corresponding reduced machine in standard form.

(a)	PS	NS, Z	
		X = 0	X = 1
A	D, 0	A, 1	
B	F, 1	C, 1	
C	D, 0	F, 1	
D	C, 0	E, 1	
E	C, 1	D, 1	
F	D, 1	D, 1	

(b)	PS	NS, Z	
		X = 0	X = 1
A	B, 1	H, 1	
B	F, 1	D, 1	
C	D, 0	E, 1	
D	C, 1	F, 1	
E	D, 1	C, 1	
F	C, 1	C, 1	
G	C, 1	D, 1	
H	C, 0	A, 1	

(c)	PS	NS, Z	
		X = 0	X = 1
A	E, 0	D, 1	
B	F, 0	D, 1	
C	E, 0	B, 0	
D	F, 0	B, 1	
E	C, 0	F, 0	
F	B, 0	C, 0	
G	D, 1	C, 1	
H	B, 1	A, 1	

(d)	PS	NS, Z	
		X = 0	X = 1
A	F, 0	B, 1	
B	F, 0	A, 1	
C	D, 0	C, 1	
D	C, 0	B, 1	
E	D, 0	A, 1	
F	E, 1	F, 1	
G	E, 1	G, 1	

- 14.2 For the state tables of the incompletely specified sequential machines given below, find the set of maximal compatibles using (a) the merger graph method and (b) the merger table method.

(a)	PS	NS, Z			
		I <sub>0</sub>	I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>
A	C, 0	—		C, 0	—
B	A, —	B, 1		C, —	—
C	—	C, 0	—, 1	D, 0	
D	F, 0	—	E, 1	C, —	
E	F, 0	—	A, —	C, 1	
F	—	B, 1	—, 0	B, 1	

(b)

PS	NS, Z			
	00	11	12	13
A	A, -	-	F, -	C, 1
B	F, -	B, 1	C, -	-
C	-	C, 0	-, 1	D, 0
D	A, 0	-	E, 1	C, -
E	C, 0	-	C, 0	-
F	-	B, 1	-, 0	B, 1

(c)

PS	NS, Z			
	00	01	11	10
A	F, 0	-	-	-
B	-	E, 1	F, 1	A, 1
C	E, 0	-	A, 0	E, 1
D	-	-	A, 1	-
E	-	C, 0	B, 0	D, 1
F	E, 0	-	-	A, 1

(d)

PS	NS, Z			
	I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	I <sub>4</sub>
A	-	C, -	-	-, 1
B	A, 1	-	B, 0	-
C	-	-	-	D, 1
D	C, -	A, -	C, -	F, 0
E	B, -	B, -	A, -	-, 0
F	-, 0	C, 1	-	H, 1
G	-, 1	E, 1	F, 1	D, 1
H	-, 1	G, -	-	F, 1

## VHDL PROGRAMS

### 1. VHDL PROGRAM FOR MEALY TYPE STATE MACHINE MODEL

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mealy_machine is
    Port ( x : in STD_LOGIC;
            clk : in STD_LOGIC;
            z : out STD_LOGIC := '0' );
end mealy_machine;

architecture Behavioral of mealy_machine is
    type state_type is (A,B,C,D);
    signal ps,ns:state_type;

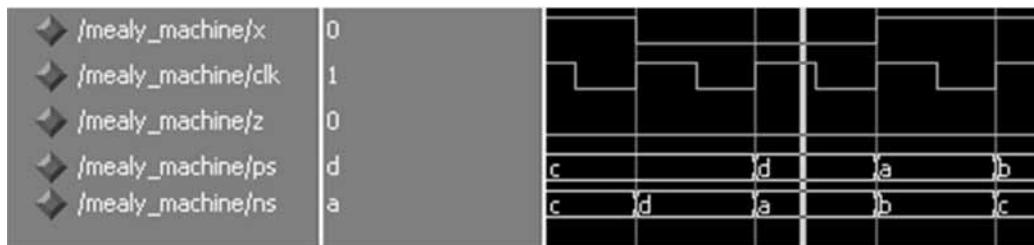
begin
    p1: process(ps,x)
    begin
        case ps is
            when A=>
                if x='1' then ns<= B;
                else ns<=A;
                end if;
            when B=>
                if x='1' then ns<=C;
                else ns<=A;
                end if;
            when C=>
                if x='1' then ns<=C;
                else ns<=D;
                end if;
            when D=>
                if x='1' then
                    z<='1';
                    ns<=B;
                else
                    z<='0';
                    ns<=A;
                end if;
            when OTHERS=> z<='0';
        end case;
    end process;
end Behavioral;

```

```

end process p1;
p2: process(clk)
begin
  if (clk ='1' and clk'event) then
    ps<= ns;
    end if;
  end process p2;
end Behavioral;

```

**SIMULATION OUTPUT:****2. VHDL PROGRAM FOR MOORE TYPE STATE MACHINE MODEL**

Example: Design of BCD Counter

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MOOREBCD is
  Port ( clk : in STD_LOGIC;
         count : out STD_LOGIC_VECTOR (3 downto 0));
end MOOREBCD;

architecture Behavioral of MOOREBCD is
Type state_type is (S0,S1,S2,S3,S4,S5,S6,S7,S8,S9);
signal ps,ns : state_type;

begin
  process(clk)
  begin
    if clk='1' and clk' event then
      ps<=ns;
    end if;
  end process;

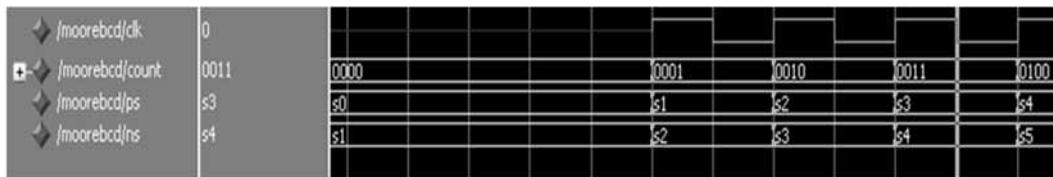
  process(ps)

```

```

begin
  case ps is
    when S0=> count <= "0000"; ns<=S1;
    when S1=> count <= "0001"; ns<=S2;
    when S2=> count <= "0010"; ns<=S3;
    when S3=> count <= "0011"; ns<=S4;
    when S4=> count <= "0100"; ns<=S5;
    when S5=> count <= "0101"; ns<=S6;
    when S6=> count <= "0110"; ns<=S7;
    when S7=> count <= "0111"; ns<=S8;
    when S8=> count <= "1000"; ns<=S9;
    when S9=> count <= "1001"; ns<=S0;
    when OTHERS => NULL;
  end case;
end process;
end Behavioral;

```

**SIMULATION OUTPUT:****VERILOG PROGRAMS****1. VERILOG PROGRAM FOR DESIGN OF MOORE MODEL USING STATE MACHINE APPROACH**

```

module moore_fsm(z, a, clk);
  output z;
  input a;
  input clk;
  reg z;
  parameter st0=0,st1=1,st2=2,st3=3;
  reg [0:1]moore_state;
  initial
  begin
    moore_state=st0;
  end
  always
  @ (posedge (clk))
  case (moore_state)

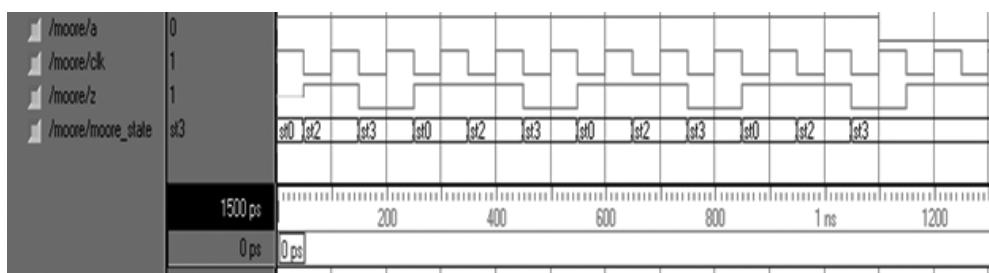
```

```

st0:
begin
z=1;
if(a)
moore_state=st2;
end
st1:
begin
z=0;
if(a)
moore_state=st3;
end
st2:
begin
z=0;
if(~a)
moore_state=st1;
else
moore_state=st3;
end
st3:
begin
z=1;
if(a)
moore_state=st0;
end
endcase
endmodule

```

### SIMULATION OUTPUT:



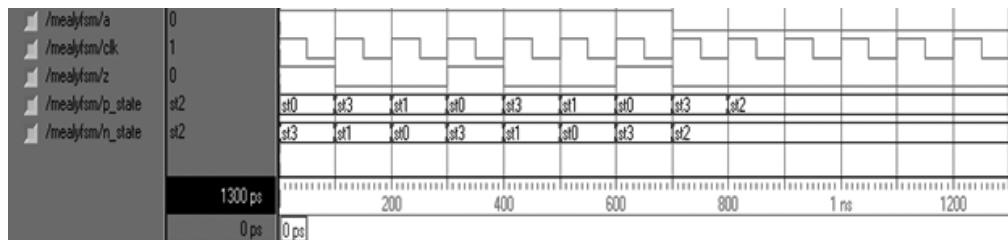
**2. VERILOG PROGRAM FOR MEALY MODEL USING STATE MACHINE APPROACH**

```
module mealy_fsm(z, a, clk);
    output z;
    input a;
    input clk;
    reg z;
    parameter st0=0,st1=1,st2=2,st3=3;
    reg [1:2]p_state,n_state;
    initial
    begin
        n_state=st0;
    end
    always
        @(posedge(clk))
        p_state=n_state;
    always
        @(p_state or a) begin:comb_part
        case(p_state)
        st0:
            if(a)
                begin
                    z=1;
                    n_state=st3;
                end
            else
                z=0;
        st1:
            if(a)
                begin
                    z=0;
                    n_state=st0;
                end
            else
                z=1;
        st2:
            if(~a)
                z=0;
            else
                begin
                    z=1;
                    n_state=st1;
                end
        end
```

```

st3:
begin
    z=0;
    if(~a)
        n_state=st2;
    else
        n_state=st1;
end
endcase
end
endmodule

```

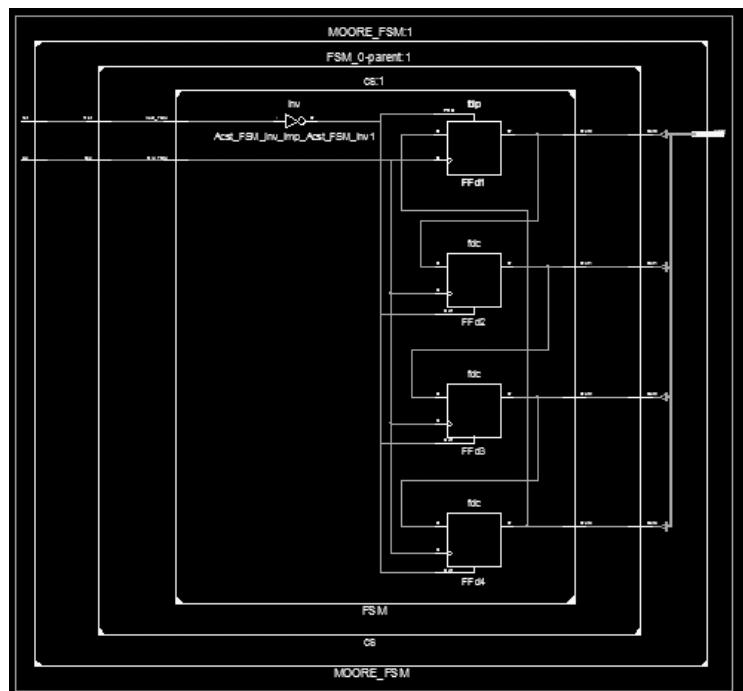
**SIMULATION OUTPUT:****3. VERILOG PROGRAM FOR 4-BIT RING COUNTER IMPLEMENTED AS A MOORE STATE MODEL**

```

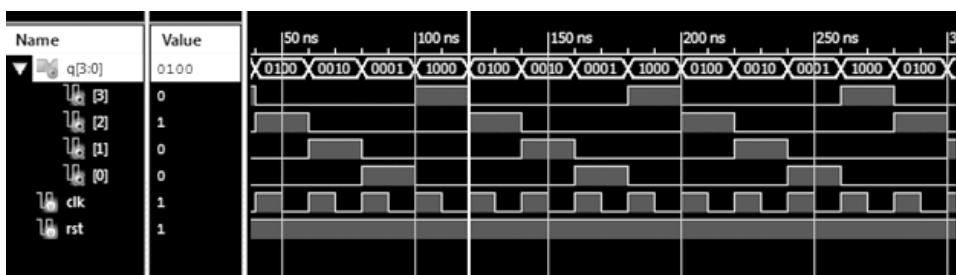
module MOORE_FSM(clk, rst, q);
input clk, rst;
output [3:0] q;
reg [3:0] cs, ns1;
always @(posedge clk or negedge rst)
begin
if(!rst) cs = 4'b1000;
else cs = ns1;
end
always @(cs)
begin
case(cs)
4'b1000: ns1 = 4'b0100;
4'b0100: ns1 = 4'b0010;
4'b0010: ns1 = 4'b0001;
4'b0001: ns1 = 4'b1000;
default: ns1 = 4'b1000;
endcase
end
assign q = cs;
endmodule

```

## RTL SCHEMATIC:



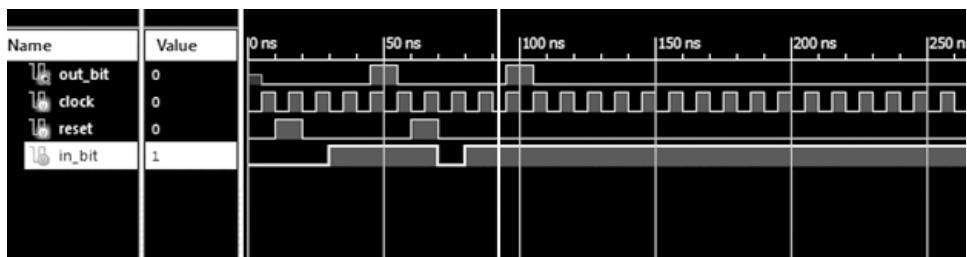
## SIMULATION OUTPUT:



## 4. VERILOG PROGRAM TO DETECT A 011 SEQUENCE USING MEALY STATE MODEL

```
module MEALY_FSM_SEQUENCE_011(clock, reset, in_bit, out_bit);
input clock, reset, in_bit;
output out_bit;
reg [2:0] state_reg, next_state;
// State declaration
parameter    reset_state      = 3'b000;
parameter    read_zero        = 3'b001;
```

```
parameter      read_0_one        =  3'b010;
parameter      read_zero_one_one = 3'b011;
// state register
always @ (posedge clock or posedge reset)
if (reset == 1)
state_reg <= reset_state;
else
state_reg <= next_state;
// next-state logic
always @ (state_reg or in_bit)
case (state_reg)
reset_state:
if (in_bit == 0)
next_state = read_zero;
else if (in_bit == 1)
next_state = reset_state;
else next_state = reset_state;
read_zero:
if (in_bit == 0)
next_state = read_zero;
else if (in_bit == 1)
next_state = read_0_one;
else next_state = reset_state;
read_0_one:
if (in_bit == 0)
next_state = read_zero;
else if (in_bit == 1)
next_state = read_zero_one_one;
else next_state = reset_state;
read_zero_one_one:
if (in_bit == 0)
next_state = read_zero;
else if (in_bit == 1)
next_state = reset_state;
else next_state = reset_state;
default: next_state = reset_state;
endcase
assign out_bit = (state_reg == read_zero_one_one)? 1 : 0;
endmodule
```

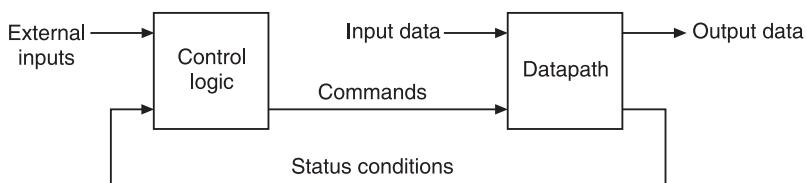
**SIMULATION OUTPUT:**

# 15

## ALGORITHMIC STATE MACHINES

### 15.1 INTRODUCTION

The binary information stored in a digital system can be classified as either data or control information. Data are discrete elements of information that are manipulated to perform arithmetic, logic, shift, and other similar data processing tasks. These operations are implemented with digital components such as adders, multiplexers, counters and shift registers. Control information provides command signals that supervise the various operations in the data section in order to accomplish the desired data processing tasks. The logic design of a digital system can be divided into two distinct parts. One part is concerned with the design of the digital circuits that perform the data processing operations. The other part is concerned with the design of the control circuits that determine the sequence in which the various actions are performed. Figure 15.1 shows the relationship between the control logic and the datapath in a digital system. The data processing path, commonly referred to as the datapath, manipulates data in registers, according to the system's requirements. The control logic initiates a sequence of commands to the datapath. The control logic uses status conditions from the datapath to serve as decision variables for determining the sequence of control signals.



**Figure 15.1** Interaction between control logic and datapath.

From the above discussion it is clear that to design a digital system, we have to design two subsystems—datapath subsystem and control subsystem. A datapath subsystem consists of digital circuits which are required to perform specified operations on the data information. A control subsystem is a sequential circuit whose internal states decide the control commands for the subsystem. At any given time, the state of the sequential circuit initiates a prescribed set of commands. The sequential circuit considers the status conditions and the other external inputs to determine the next state to initiate other operations.

The control sequence and datapath tasks of a digital system are represented by means of a hardware algorithm. An algorithm consists of a finite number of procedural steps that specify how to obtain a solution to a problem. A hardware algorithm is a step-by-step procedure to implement the desired tasks with selected circuit components.

A state machine is another term for a sequential circuit which is the basic structure of a digital system. Just as a flow chart serves as a useful aid in writing software programs, algorithmic state machine (ASM) charts help in the hardware design of digital systems. An ASM chart is a special flow chart that has been developed specifically to define digital hardware algorithms. ASM charts provide a conceptually simple and easy to visualize method of implementing algorithms used in hardware design. ASM charts are advantageously used in the design of the control unit of the computer and in general control networks in any digital systems. ASM charts look similar to flow charts but differ in that certain specific rules must be observed in constructing them. An ASM chart is equivalent to a state diagram or a state graph. Normally state diagrams are used for the design of finite state machines. For larger circuits ASM charts are useful. ASM chart is a useful tool which leads directly to hardware realization.

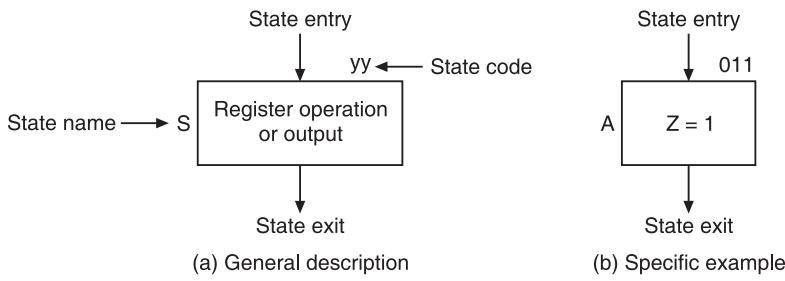
The ASM chart is an alternative for describing the behaviour of finite state machines and is similar to a conventional flow chart used in various engineering designs except for the timing considerations. A conventional flow chart describes the sequence of procedural steps and decision paths for an algorithm without any concern for their time relationship. The ASM chart on the other hand describes the sequence of events as well as the timing relationships between the states of a sequence controller and the events that occur while going from one state to the next state after each clock edge.

## 15.2 COMPONENTS OF ASM CHART

There are three components of ASM chart—(1) State box, (2) decision box and (3) conditional output box.

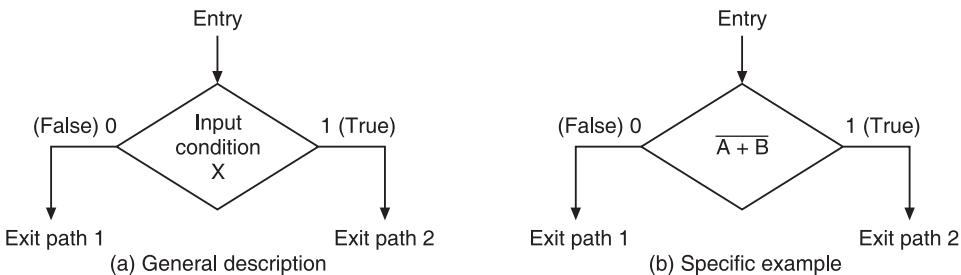
Out of these, the state and decision boxes are familiar from use in conventional flow charts. The third element, the conditional box, is unique to the ASM chart.

**1. State box:** A state of a clocked sequential circuit is represented by a rectangle called *state box*. It is equivalent to a node in the state diagram or a row in the state table. The name of the state is written to the left of the box. The binary code assigned to the state is indicated outside on the top right-side of the box. A list of unconditional outputs if any associated with the state are written within the box. They are clearly Moore type outputs. These outputs depend only on the values of the state variables that define the state. Finally a line drawn into the state box indicates the state entry and a line drawn from the state box known as state exit indicates the path to the next state. Figure 15.2 shows a state box.



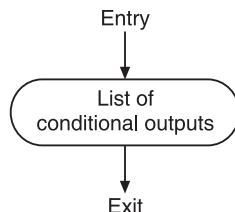
**Figure 15.2** State box.

**2. Decision box:** The decision box or condition box is represented by a diamond-shaped symbol with one input and two or more output paths. The output branches are true and false branches. The decision box describes the effect of an input on the control subsystem. A Boolean variable or input or expression written inside the diamond indicates a condition which is evaluated to determine which branch to take. The exit paths lead to the blocks corresponding to the next states of the circuit following the next clock pulse. For example, X written inside this box indicates that the decision is based on the value of X, whereas  $\overline{A + B}$  written inside the box indicates that the true path is chosen if  $A + B = 1$  and the false path is chosen otherwise. Figure 15.3 shows a decision box.



**Figure 15.3** Decision box.

**3. Conditional output box:** The conditional output box is represented by a rectangle with rounded corners or by an oval with one input line and one output line. The outputs that depend on both the state of the system and the inputs are indicated inside the box. These are Mealy type outputs. Figure 15.4 shows a conditional output box.



**Figure 15.4** Conditional output box.

**ASM block:** An ASM chart is constructed from one or more interconnected ASM blocks. Each ASM block contains a single state box and any decision and conditional output boxes associated with that state. An ASM block has a single entry path and single or multiple exit paths represented by the structure of the decision boxes. Each ASM block is associated with one specific state; therefore, it describes the finite state machine operation during the time the machine is in that state, i.e. it describes the state of the system under one clock pulse interval.

When a synchronous sequential system enters the state included in a given ASM block, the outputs in the output list in the state box are asserted (become true). The conditions in the decision boxes are evaluated to determine which path(s) is followed through the ASM block. When a conditional output box is encountered in such a path, the corresponding conditional outputs become true (asserted). A path through an ASM block from entry to exit is referred to as a link path.

### 15.3 SALIENT FEATURES OF ASM CHARTS

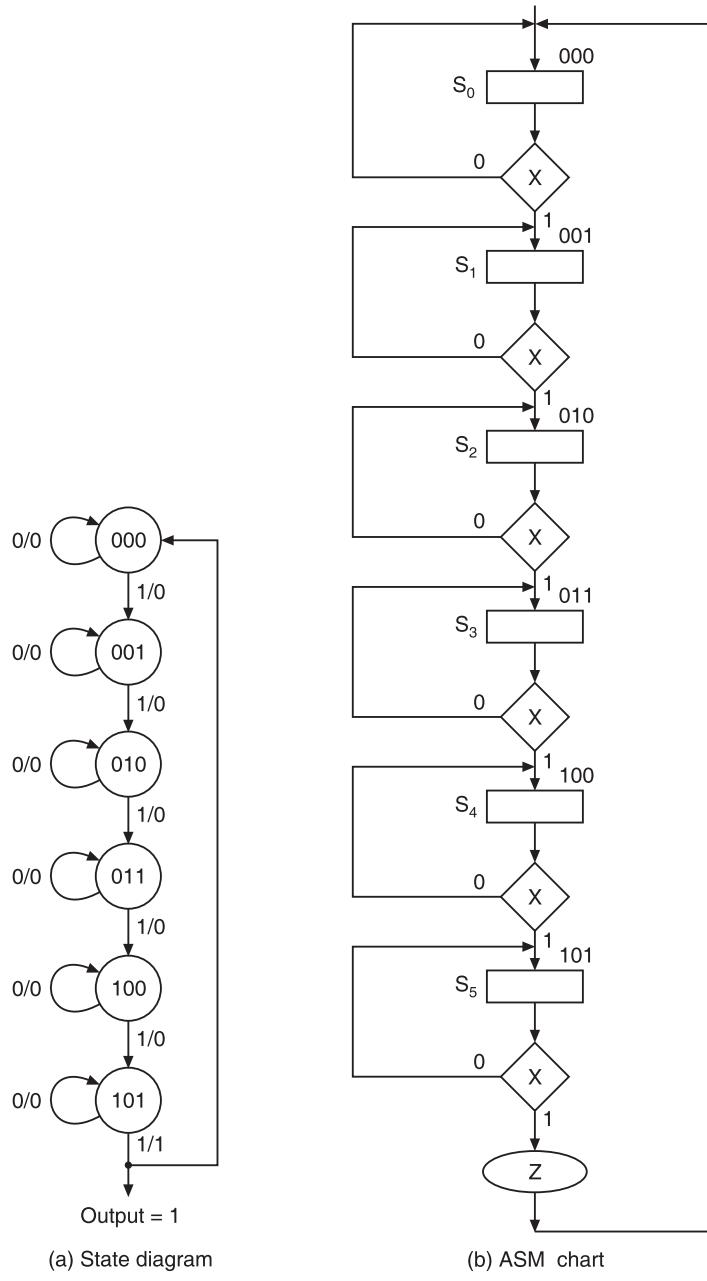
The following are the salient features of ASM charts:

1. An ASM chart describes the sequence of events as well as the timing relationship between the states of a sequential controller and the events that occur while going from one state to the next.
2. An ASM chart contains one or more interconnected ASM blocks.
3. Each ASM block contains exactly one state box together with the decision boxes and conditional output boxes associated with that state.
4. Every block in an ASM chart specifies the operations that are to be performed during one common clock pulse.
5. An ASM block has exactly one entrance path and one or more exit paths represented by the structure of the decision boxes.
6. A path through an ASM block from entrance to exit is referred to as a link path.
7. The operations specified within the state and conditional output boxes in the block are performed in the datapath subsystem.
8. Internal feedback within an ASM block is not permitted. Even so, following a decision box or conditional output boxes, the machine may reenter the same state.
9. Each block in the ASM chart describes the state of the system during one clock pulse interval. When a digital system enters the state associated with a given ASM block, the outputs indicated within the state box become true. The conditions associated with the decision boxes are evaluated to determine which path or paths to be followed to enter the next ASM block.

### 15.4 INTRODUCTORY EXAMPLES OF ASM CHARTS

**Example 1 Mod-6 counter:** The state diagram of a mod-6 counter is shown in Figure 15.5a. ASM chart equivalent of Figure 15.5a is shown in Figure 15.5b. The states are now represented by state boxes instead of nodes. State transitions are still represented by arrows but the inputs indicated adjacent to arrows in the state diagram are replaced by decision boxes with true and false branches. In the state diagram the outputs are indicated along with inputs using a separator slash or comma.

In the ASM chart Mealy type outputs are now indicated in conditional output boxes and Moore type outputs are indicated within the state box itself.



**Figure 15.5** State diagram and ASM chart for mod-6 counter.

Look at the ASM chart which has six state boxes named  $S_0$  through  $S_5$ . For every clock pulse,  $X$  is sensed. If 1, then, the machine goes to the next state: if 0, then, the machine remains in

the same state as indicated by a return branch. All this happens in the same clock cycle. When the machine is in state  $S_5$ , on the occurrence of the clock pulse, if  $X$  is 1, the machine produces an output pulse indicated as  $Z$  and goes to the initial state  $S_0$ . The states are assigned binary numbers using three flip-flops.

Let us design it using D flip-flops. With D flip-flops, the excitation  $D_i$  has to be the same as the next state variable  $Y_i$ . Observing only state assignment indicated on the ASM chart, we notice that the next value  $Y_0$  has to become 1 for the present states  $y_2 y_1 y_0 = 000, 010, 100$  only. Hence using decimal codes and remembering that states 6 (110), and 7 (111) never occur, we get

$$D_0 = Y_0 = X [\Sigma m(0, 2, 4) + d(6, 7)]$$

Similarly, we obtain expressions for all the excitations and outputs by merely inspecting the ASM chart.

$$D_1 = Y_1 = X [\Sigma m(1, 2) + d(6, 7)]$$

$$D_2 = Y_2 = X [\Sigma m(3, 4) + d(6, 7)]$$

$$Z = X [\Sigma m(5) + d(6, 7)]$$

Notice that the input  $X$  is ANDed with each of the expressions for the excitations. If the input  $X$  is exclusively provided, then the circuit counts the number of clock pulses in the duration when  $X$  is at level 1. In other words,  $X$  enables the counter. Some times,  $X$  is not provided, in which case the clock pulses are counted modulo 6 and an output pulse is produced for every 6 clock pulses.

**Example 2 Sequence detector:** Let us say we want a sequence detector to detect the sequence 1010. The state diagram of the sequence detector is shown in Figure 15.6a. The ASM chart of the sequence detector corresponding to that state diagram is shown in Figure 15.6b.

The machine has four states. So, four state boxes are required. While at each state, the machine has to decide to which next state it has to go when the input 0 or 1 is given. Four decision boxes with two branches each are also required. It has to output a 1 once the sequence 1010 is detected. So it requires one conditional output box. The state assignment is arbitrary. Let it be  $S_0 = 00$ ,  $S_1 = 01$ ,  $S_2 = 10$  and  $S_3 = 11$ .

Notice that there are four ASM blocks, each containing one state box associated with one decision box and the last ASM block has an output box as well. All the operations associated with a state box from one ASM block have to be performed at the same time in one clock pulse period. There are four states and  $y_1 y_0$  are the binary values assigned to the states. If we decide to use two D flip-flops, the excitation table is identical to the transition table and  $D_i = Y_i$ . In this ASM chart, note that the next state variable  $Y_0$  becomes 1 in states  $S_1$  (01) and  $S_3$  (11). The corresponding present states are  $S_0$  ( $y_1 y_0 = 00$ ) and  $S_2$  ( $y_1 y_0 = 10$ ) and transition occurs if  $X = 1$  and 1 respectively. Thus, we may write the expressions for excitations by inspection as follows.

$$D_0 = Y_0 = X \bar{y}_1 \bar{y}_0 + X y_1 \bar{y}_0 = X \bar{y}_0 (\bar{y}_1 + y_1) = X \bar{y}_0$$

From the ASM chart observe that  $Y_1$  becomes 1 for  $S_2$  and  $S_3$  on  $X = 0$  from  $S_1$  and  $X = 1$  from  $S_2$  respectively.

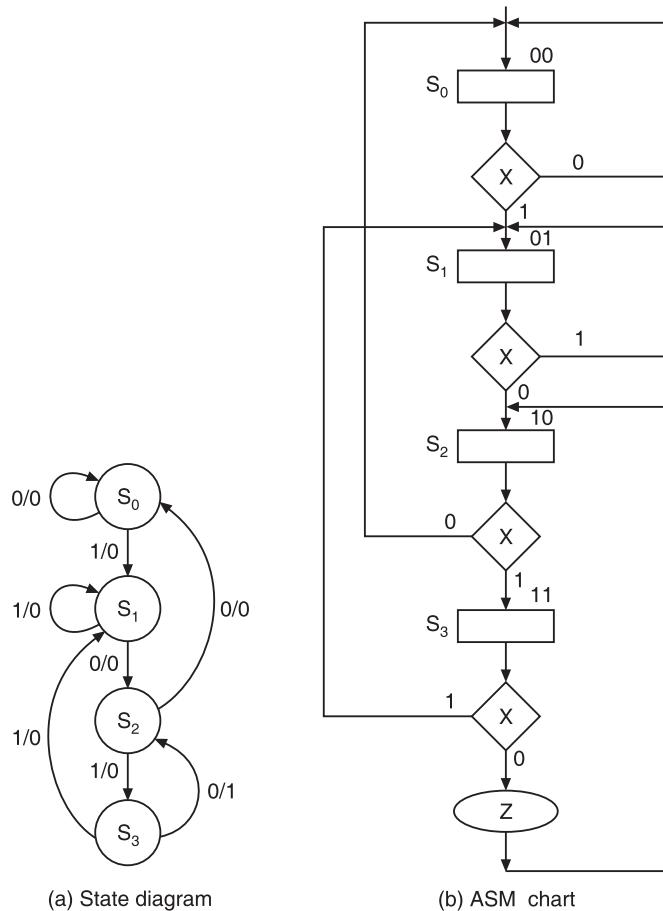
$$D_1 = Y_1 = \bar{X} \bar{y}_1 y_0 + X y_1 \bar{y}_0$$

The output  $Z = 1$  when input  $X = 1$  and the present state  $y_1 y_0 = 11$ . Therefore,

$$Z = X y_1 y_0$$

A logic circuit can be drawn based on the above expressions.

The ASM chart is drawn like this:



**Figure 15.6** State diagram and ASM chart of a sequence detector.

**Example 3 Serial adder:** We discussed earlier that a serial adder adds serial binary numbers. The state diagram of the serial adder is shown in Figure 15.7a. The structure of the serial adder is shown in Figure 15.7b. The ASM chart equivalent to the state diagram describing the same behaviour is shown in Figure 15.8. In the ASM chart observe that there are four exit paths from each state depending on the values of inputs A and B arranged in the form of a binary tree. Also observe that all the paths merge into one entry path for each state. Comparing it with the state diagram, we observe that in the state diagram from each state there is an exit path to the other state and three re-entry paths to itself. This is because there are four different values that AB can assume 00, 01, 10 and 11 and hence there must be four exit paths—three of them happen to be re-entry paths in this example. Observe in the state diagram that there are four incoming paths which merge into one entry path to each state. Correspondingly, there are as many branches in the ASM chart. If the number of inputs is large, the state graph will become quite complex and the ASM chart will become unwieldy. The decision boxes are usually binary as we aim at an algorithmic procedure comprising primitive steps. As we have two inputs we have to show four exit paths from each state.

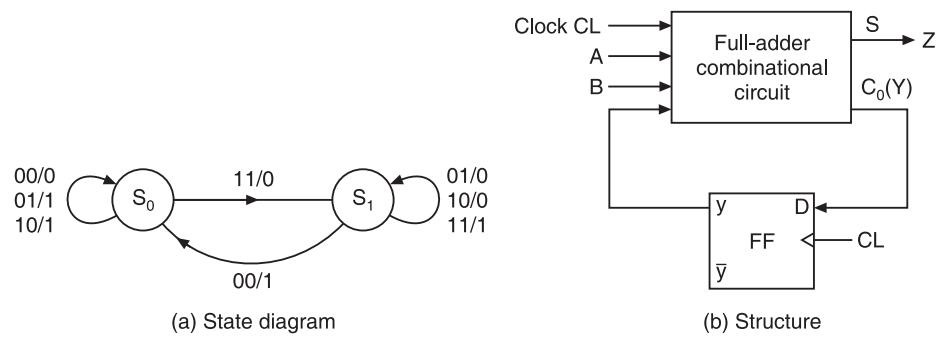


Figure 15.7 State diagram and structure of a serial adder.

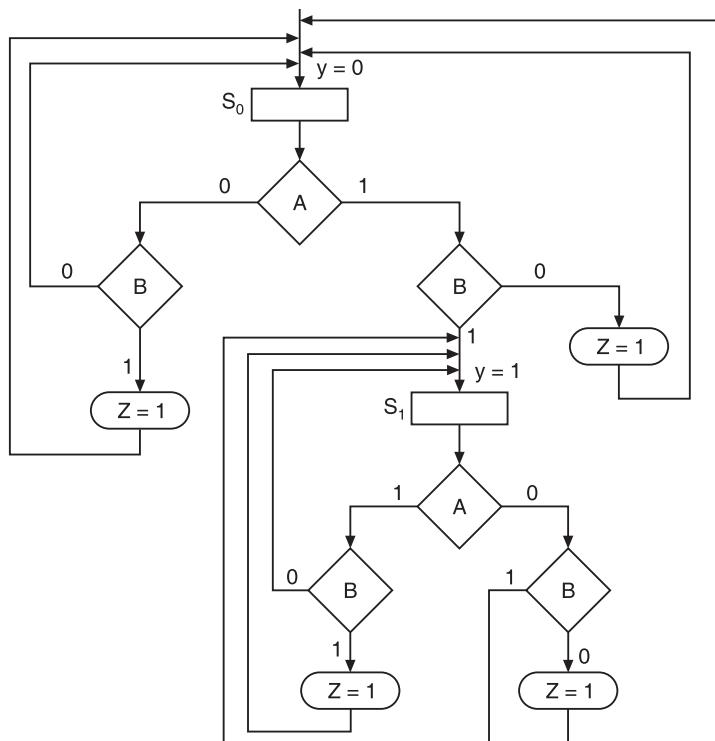


Figure 15.8 ASM chart of serial adder.

*Synthesis of the circuit:* By inspection of the ASM chart, we note that there are only two states  $S_0$  and  $S_1$  and hence we need to have only one flip-flop. We choose a D flip-flop, because D is easily synthesized looking at the entry paths into each state. For the only flip-flop, D becomes 1 for state  $S_1$ . There are four arrows confluent on entry into  $S_1$ . Hence, by inspection and tracing the paths indicated by arrows, we write the expression for D given below. Remember that y is the present state variable of the only flip-flop.

$$\begin{aligned}
 D &= \bar{y}AB + y\bar{A}\bar{B} + yAB + \bar{y}\bar{A}\bar{B} \\
 &= AB(y + \bar{y}) + Ay(B + \bar{B}) + By(A + \bar{A}) \\
 &= AB + Ay + By
 \end{aligned}$$

By a similar reasoning, the state  $S_0$  is characterized by  $y = 0$ . The corresponding excitation is  $\bar{D}$ . There are four paths confluent on the entry into  $S_0$ . By inspection and tracing the paths, we may write

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

See the consistency in the expressions for  $D$  and  $\bar{D}$ . There are four conditional output boxes.

Remember that the output  $Z$  is the sum bit to be produced serially. Tracing the corresponding paths it is easy to write the expression for  $Z$ .

$$\begin{aligned}
 Z &= \bar{y}AB + \bar{y}AB + y\bar{A}\bar{B} + yAB \\
 &= \bar{y}(AB + AB) + y(\bar{A}\bar{B} + AB) \\
 &= \bar{y}(A \oplus B) + y(A \odot B) \\
 &= A \oplus B \oplus y
 \end{aligned}$$

Notice that the expressions for  $Z$  and  $D$  are the same as those for sum  $S$  and output carry  $C_0$  of a full-adder.

*Control subsystem implementation:* The control subsystem consists of a sequential circuit. The process to implement the sequential circuit described by ASM chart consists of the following steps:

1. Translate the ASM chart to a state table.
2. Assign the states and convert the state table to a transition and output table.
3. Select the type of flip-flops and obtain the excitation table.
4. Obtain the minimal expressions for circuit outputs and memory element inputs in terms of the present state variables and external inputs.
5. Draw a logic diagram based on those minimal expressions.

**EXAMPLE 15.1** Develop an ASM chart and state table for a controllable waveform generator that will output any one of the four waveforms given in Figure 15.9 as determined by the values of its two inputs  $X_1$  and  $X_2$ . The period of the first two waveforms is four clock cycles, the period of the third is three, and the period of the fourth waveform is two clock cycles respectively. When an input change does occur, the new waveform may begin at any point in its period.

### **Solution**

The longest waveform given is of four cycles. So the ASM chart for the above waveforms can be drawn with four states, one for each clock cycle of the waveforms as shown in Figure 15.10a. For each state, the output will be conditional on the values of input lines in effect at that time. Let us observe the different conditions at different states.

1. **State A:** In the first state A, i.e. in the first clock cycle, all waveforms are at logic 1. Therefore, the output  $Z = 1$  (unconditional output) and is listed in the state box for the first state A.

- 2. State B:** In the second state B, i.e. in the second clock cycle, the output is 1 for input combinations  $X_1X_2 = 01$  and  $10$ , and the output is 0 for input combinations  $X_1X_2 = 00$  and  $11$ . These conditions can be tested by  $X_1$  in the first decision box followed by  $X_2$  in other two decision boxes. The outputs from state B are conditional, separate conditional output boxes are used. Looking at Figure 15.9, we can observe that the fourth waveform repeats after two cycles. Therefore, when  $X_1X_2 = 11$ , the third and fourth states are not used and the line goes back to the first state to start the new cycle.
- 3. State C:** In the third state C, i.e. in the third clock cycle, the output will be 1, if  $X_2 = 1$ . The condition of  $X_2$  is checked and accordingly output is made 1 by decision box and conditional box in state C. Looking at Figure 15.9, we can observe that the third waveform repeats after three clock cycles. Therefore, when  $X_1X_2 = 10$ , the fourth state is not used and the line goes back to the first state to start the new cycle.
- 4. State D:** In the fourth state D, i.e. in the fourth clock cycle, the output is 0 for  $X_1X_2 = 00$  and  $01$ , and for  $X_1X_2 = 10$  and  $11$  the next cycle has already started. So from state D, the circuit returns to state A so that the waveforms for inputs  $X_1X_2 = 00$  and  $01$  may be repeated.

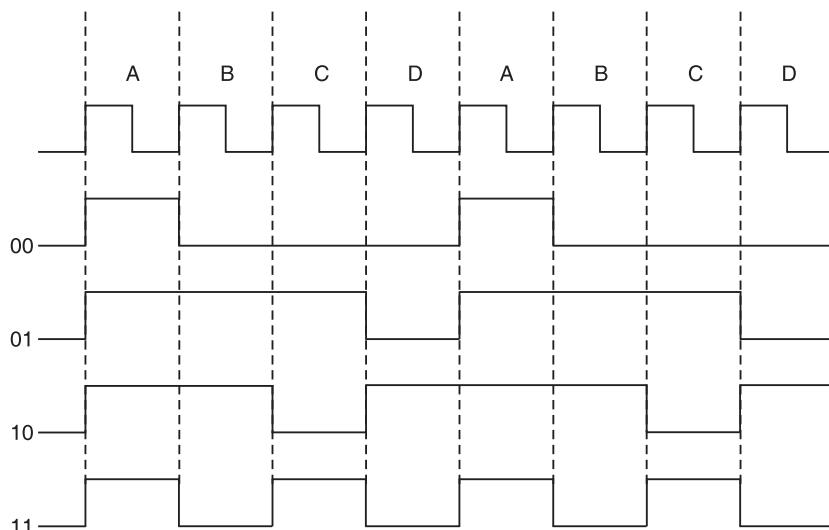


Figure 15.9 Example 15.1: Waveforms.

The ASM chart of the waveform generator can be expressed as a state table as shown in Figure 15.10b. In the state table, the outputs along with next states are listed in the columns corresponding to each combination of input values and the states are listed in the rows.

In the state table there are three don't cares. When the machine is in state C, for input  $X_1X_2 = 11$ , the next state and output are don't cares because that waveform is repeated after two clock cycles. Also when the machine is in state D, the next state and output entries for  $X_1X_2 = 10$  and  $11$  are don't cares because these waveforms are repeated after the third clock pulse.

**EXAMPLE 15.2** Determine the transition table for the waveform generator from the state table given in Figure 15.10b.

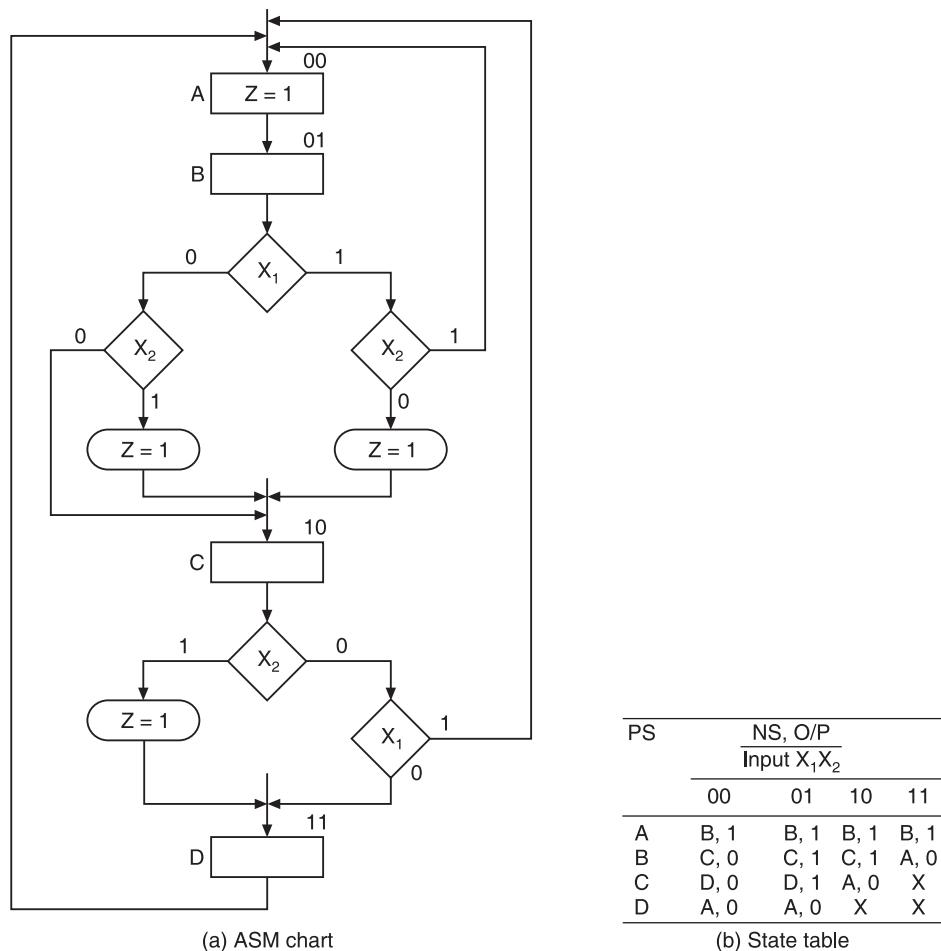


Figure 15.10 Example 15.2: ASM chart and state table for the waveform generator.

### Solution

From the ASM chart we observe that the sequential machine has four states. So two state variables are required. Assign the states as  $A \rightarrow 00$ ,  $B \rightarrow 01$ ,  $C \rightarrow 10$ , and  $D \rightarrow 11$ . Substituting these values in the state table, we get the transition and output table as shown in Table 15.1.

Table 15.1 Example 15.2: Transition table

PS	NS				O/P				
	Input $X_1X_2$				Input $X_1X_2$				
	00	01	10	11		00	01	10	11
$A \rightarrow 00$	0 1	0 1	0 1	0 1		1	1	1	1
$B \rightarrow 01$	1 0	1 0	1 0	0 0		0	1	1	0
$C \rightarrow 10$	1 1	1 1	0 0	x		0	1	0	x
$D \rightarrow 11$	0 0	0 0	x	x		0	0	x	x

**EXAMPLE 15.3** Determine the excitation table for Example 15.1.

**Solution**

Select D flip-flops and from the entries of the transition and output table (Table 15.1) obtain the excitation table as shown in Table 15.2.

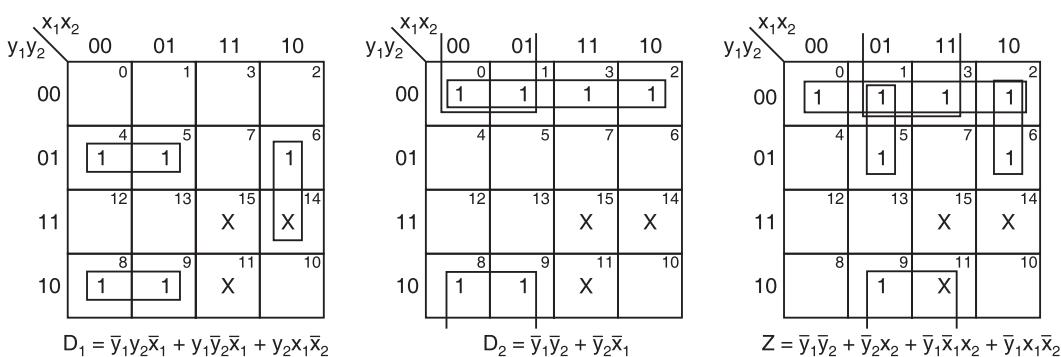
**Table 15.2** Example 15.3: Excitation table

PS		I/Ps		NS		Required excitations		O/P
$y_1$	$y_2$	$X_1$	$X_2$	$Y_1$	$Y_2$	$D_1$	$D_2$	Z
0	0	0	0	0	1	0	1	1
0	0	0	1	0	1	0	1	1
0	0	1	0	0	1	0	1	1
0	0	1	1	0	1	0	1	1
0	1	0	0	1	0	1	0	0
0	1	0	1	1	0	1	0	1
0	1	1	0	1	0	1	0	1
0	1	1	1	0	0	0	0	0
1	0	0	0	1	1	1	1	0
1	0	0	1	1	1	1	1	1
1	0	1	0	0	0	0	0	0
1	0	1	1	x	x	x	x	x
1	1	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0
1	1	1	0	x	x	x	x	x
1	1	1	1	x	x	x	x	x

*Determination of minimal expressions:* Based on the information in the excitation table, the minimal expressions for excitations and output can be determined using

1. K-map simplification method or
2. Multiplexer control method.

*K-map simplification method:* The K-maps for  $D_1$ ,  $D_2$  and Z, their minimization, and the minimal expressions obtained from them are shown in Figure 15.11.



**Figure 15.11** Example 15.3: K-maps.

*Logic diagram:* Based on the minimal expressions, a logic diagram using conventional hardware can be drawn as shown in Figure 15.12.

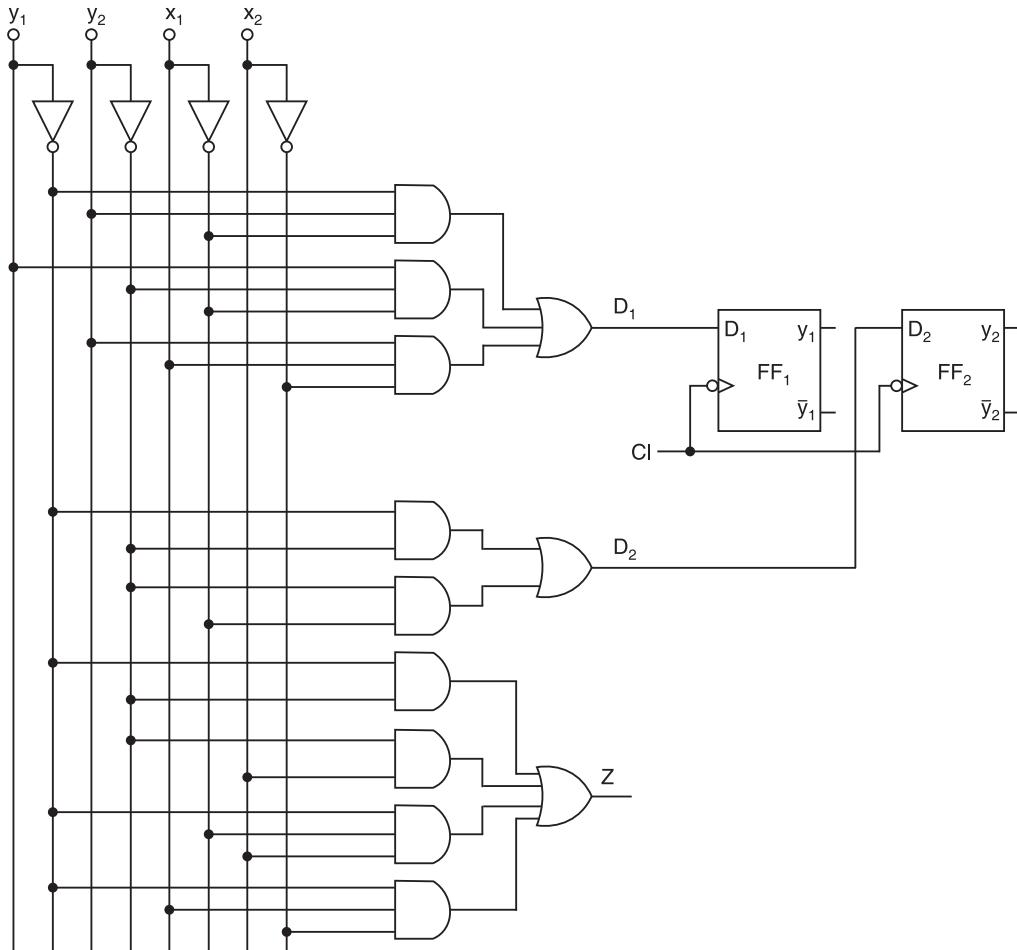
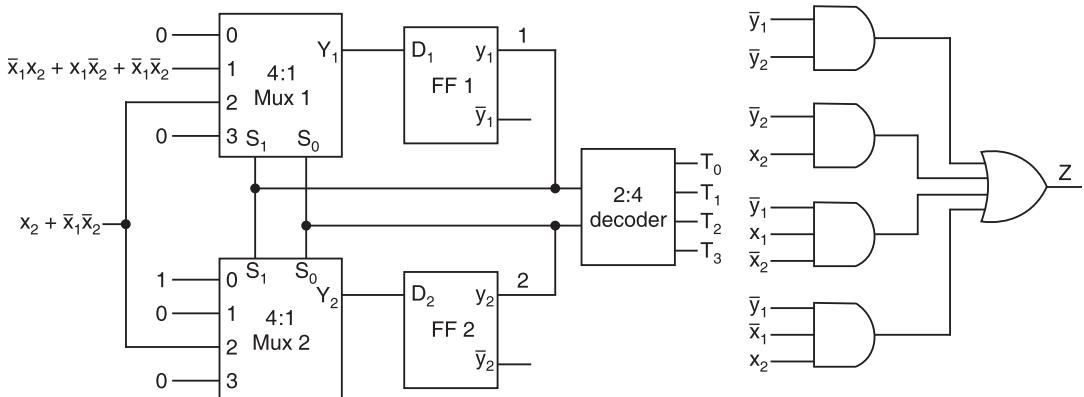


Figure 15.12 Example 15.3: Logic diagram.

*Multiplexer control method:* It is a simpler and straightforward method for the realization of combinational circuit for any controller. In this method, the gates and flip-flops are replaced by multiplexers and registers respectively. In this method, there are three levels of the components. The first level consists of multiplexers that determine the next state of the register. So the outputs of the multiplexers are to be connected to the inputs of the flip-flops in the register. The second level contains a register that holds the present binary state. The third level has a decoder that provides a separate output for each control state. Sometimes a combinational circuit is used in place of a decoder.

Consider, for example, the ASM chart of Figure 15.10a. It consists of four states and two control inputs  $X_1$  and  $X_2$ . Figure 15.13 shows the three level implementation. It consists of two 4:1 multiplexers, Mux 1 and Mux 2; a register with two flip-flops and a decoder. A combinational

circuit is used to determine the output. The outputs of the register are used to select the inputs of the multiplexers. In this way the present state of the register is used to select one of the inputs from each multiplexer. The outputs of the multiplexers are then applied to the D inputs of flip-flops 1 and 2. The purpose of each multiplexer is to produce an input to its corresponding flip-flop equal to the binary value of the next state.



**Figure 15.13** Example 15.3: Logic diagram using multiplexers.

The inputs of the multiplexers are determined from the decision boxes and state transitions given in the ASM chart (refer to Figure 15.10a). The present states, next states and conditions for transitions can be tabulated for ASM chart given in Figure 15.10a as shown in Table 15.3.

**Table 15.3** Example 15.3: Transition table

No.	Present state		Next state		Condition of transition	
		$y_1$		$y_0$		
1	A	0	0	B	0 1	1
2	B	0	1	A	0 0	$x_1 x_2$
				C	1 0	$x_1 \bar{x}_2$
				C	1 0	$\bar{x}_1 \bar{x}_2$
				C	1 0	$\bar{x}_1 x_2$
3	C	1	0	A	0 0	$x_1 \bar{x}_2$
				D	1 1	$x_2$
				D	1 1	$\bar{x}_1 \bar{x}_2$
4	D	1	1	A	0 0	1

*Inputs for multiplexers:* MUX 1 generates input for flip-flop 1 and MUX 2 generates input for flip-flop 2. The multiplexer input (see Table 15.4) can be determined by including the condition of transition corresponding to logic 1 bit position in the next state. Consider the transition from A to B. For flip-flop 1, the next state is 0, hence the corresponding input (input 0) of multiplexer 1 is 0. For flip-flop 2, the next state is 1, hence the corresponding input (input 0) of multiplexer 2 is the

given condition of transition, i.e. 1. Consider the transition from B to A or C. In this case, the next state for flip-flop 1 is 0 for A and 1 for C. Therefore, the sum of the conditions of transition corresponding to C, i.e.  $x_1\bar{x}_2 + \bar{x}_1x_2 + \bar{x}_1\bar{x}_2$  is taken as the corresponding input (input 1) of multiplexer 1. The next state for flip-flop 2 is 0 for both A and C, hence, the corresponding input (input 1) of multiplexer 2 is 0.

**Table 15.4** Example 15.3: Inputs for multiplexers

MUX 1	MUX 2
0 → 0	0 → 1
1 → $\bar{x}_1x_2 + x_1\bar{x}_2 + \bar{x}_1\bar{x}_2$	1 → 0
2 → $x_2 + \bar{x}_1\bar{x}_2$	2 → $x_2 + \bar{x}_1\bar{x}_2$
3 → 0	3 → 0

Consider the transition from C to A or D. In this case, the next state to both flip-flop 1 and flip-flop 2 is 0 for A and 1 for D. Therefore, the sum of the conditions of transition corresponding to D, i.e.  $x_2 + \bar{x}_1\bar{x}_2$  is taken as the corresponding input (input 2) of the multiplexers 1 and 2. Similarly, consider the transition from D to A. In this case, the next state for flip-flop 1 is 0. For flip-flop 2 also it is 0 only. So connect 0 to input 3 of both the multiplexers.

The equation for output Z can be directly derived from the ASM chart. For this we have to observe the ASM chart and find the conditions when output is 1. For example, in state A,  $Z = 1$ , therefore, A will appear in the equation for output Z. After collecting all the conditions where  $Z = 1$ , we get

$$Z = A + Bx_1\bar{x}_2 + B\bar{x}_1x_2 + Cx_2 = \bar{y}_1\bar{y}_2 + \bar{y}_1y_2x_1\bar{x}_2 + \bar{y}_1y_2\bar{x}_1x_2 + y_1\bar{y}_2x_2$$

Drawing a K-map and simplifying, the expression for output Z is

$$Z = \bar{y}_1\bar{y}_2 + \bar{y}_2x_2 + \bar{y}_1\bar{x}_1x_2 + \bar{y}_1x_1\bar{x}_2$$

**Logic diagram:** A logic diagram using multiplexers can be drawn as shown in Figure 15.13.

**PLA control:** We have already discussed in Chapter 5 how to implement combinational logic circuit using PLA. We can also implement sequential circuit using PLA with the help of registers. The PLA part of the circuit implements the combinational circuit and the register part implements the memory.

**EXAMPLE 15.4** An ASM chart of a waveform generator is given in Figure 15.14.  $X_1$  and  $X_2$  are the inputs and Z is the output. Draw the waveforms and design the circuit.

### Solution

From the ASM chart we notice that there are two inputs  $X_1$  and  $X_2$  and one output Z. The two inputs can be combined in  $2^2 = 4$  ways. So four output waveforms can be generated. There are four states. So the longest waveform is of four pulses.

**State A (00):** During the first clock pulse, the machine is in state A. While in A, it produces an unconditional output  $Z = 1$  and goes to state B. So during the first pulse all the four waveforms are at logic 1 level.

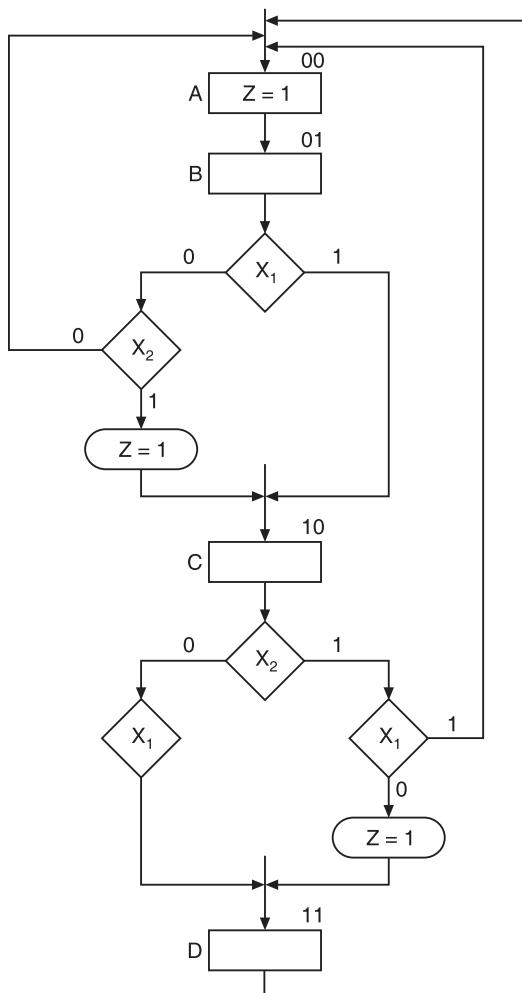


Figure 15.14 Example 15.4: ASM chart.

**State B (01):** During the second clock pulse, the machine is in state B. While in state B, if input  $X_1 = 1$ , the machine outputs a 0 and goes to state C. So the waveforms corresponding to  $X_1X_2 = 10$  and 11 will be at logic 0 level during this period. If  $X_1X_2 = 00$ ,  $Z = 0$  and the machine goes to state A to repeat the waveform. So during this period the waveform for  $X_1X_2 = 00$  is at logic 0 level. Hence the waveform corresponding to  $X_1X_2 = 00$  is of two cycles only. If  $X_1X_2 = 01$ ,  $Z = 1$  and the machine goes to state C. So during the second clock pulse the waveform for  $X_1X_2 = 01$  is at logic 1 level.

**State C (10):** During the third clock pulse, the machine is in state C. While in state C, if  $X_1X_2 = 11$ , it outputs a 0 and goes to state A for restarting. So during this period the waveform for  $X_1X_2 = 11$  is at logic 0 level. Hence the waveform of  $X_1X_2 = 11$  is of three cycles only. If  $X_1X_2 = 01$ ,  $Z = 1$ . So during the third clock pulse the waveform for  $X_1X_2 = 01$  is at logic 1 level. If  $X_1X_2 = 00$  or 10,  $Z = 0$  and the machine goes to state D. So the waveforms for  $X_1X_2 = 00$  and 10 are at logic 0 level during this period.

**State D (11):** During the fourth clock pulse, the machine is in state D. While in state D, if  $X_1X_2 = 01$  or 10 the output is at logic 0 level. In state D the inputs  $X_1X_2 = 00$  and 11 are invalid because the corresponding waveforms are already getting repeated. From here the machine goes to state A.

Based on the above description the waveforms are drawn as shown in Figure 15.15.

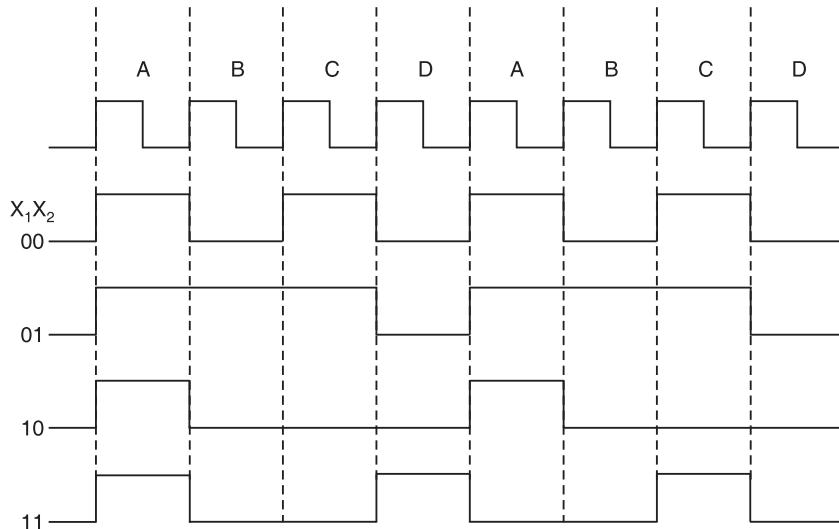


Figure 15.15 Example 15.4: Waveforms.

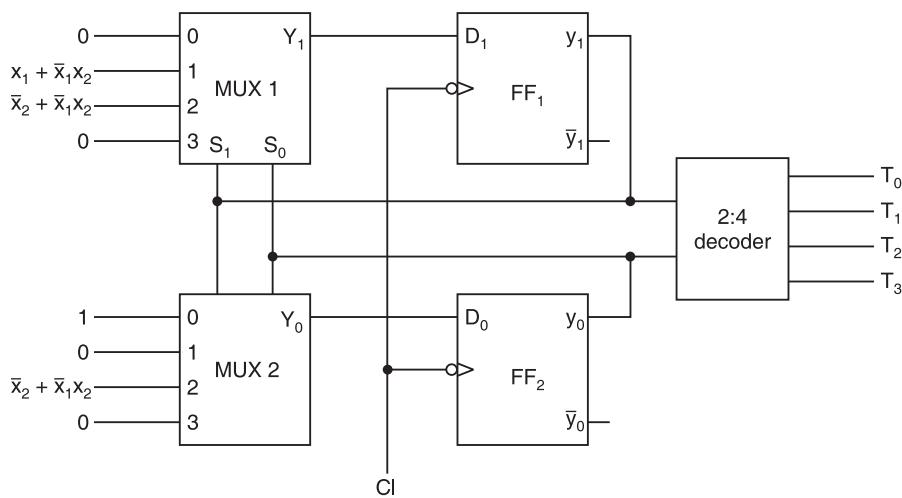
Let us design the circuit using the multiplexer control method. The transition table is shown in Table 15.5. The inputs for multiplexers are shown in Table 15.6. The logic diagram using multiplexers is shown in Figure 15.16.  $T_0, T_1, T_2, T_3$  are the control signals when the machine is in states A, B, C and D respectively.

Table 15.5 Example 15.4: Transition table

No.	PS		NS		Condition of transition	
		$y_1$	$y_0$	$Y_1$	$Y_0$	
1	A	0	0	B	0 1	1
2	B	0	1	C	1 0	$x_1$
				C	1 0	$\bar{x}_1x_2$
				A	0 0	$\bar{x}_1\bar{x}_2$
3	C	1	0	A	0 0	$x_1x_2$
				D	1 1	$\bar{x}_1x_2$
				D	1 1	$\bar{x}_2$
4	D	1	1	A	0 0	1

**Table 15.6** Example 15.4: Inputs for multiplexers

MUX 1	MUX 2
$0 \rightarrow 0$	$0 \rightarrow 1$
$1 \rightarrow x_1 + \bar{x}_1x_2$	$1 \rightarrow 0$
$2 \rightarrow \bar{x}_2 + \bar{x}_1x_2$	$2 \rightarrow \bar{x}_2 + \bar{x}_1x_2$
$3 \rightarrow 0$	$3 \rightarrow 0$

**Figure 15.16** Example 15.4: Logic diagram.

## 15.5 ASM FOR BINARY MULTIPLIER

The binary multiplier is designed using the add shift algorithm. The manual working is given in Figure 15.17 using the unsigned binary numbers 1101 as the multiplicand and 1010 as the multiplier. The important points in the multiplication process are as follows:

$$\begin{array}{r}
 1\ 1\ 0\ 1 \\
 1\ 0\ 1\ 0 \\
 \hline
 0\ 0\ 0\ 0 \\
 1\ 1\ 0\ 1 \\
 0\ 0\ 0\ 0 \\
 1\ 1\ 0\ 1 \\
 \hline
 1\ 0\ 0\ 0\ 0\ 1\ 0
 \end{array} \quad \leftarrow \begin{array}{l} 13_{10} \dots \text{Multiplicand} \\ 10_{10} \dots \text{Multiplier} \\ \text{Partial product 1} \\ \text{Partial product 2} \\ \text{Partial product 3} \\ \text{Partial product 4} \\ \text{Product} \end{array}$$

**Figure 15.17** Manual multiplication algorithm.

1. Form the partial products by multiplying the multiplicand with each of the multiplier bits starting from the LSB and proceeding towards the MSB. When the multiplier bit is a 0, the partial product is 0 and when the multiplier bit is a 1 the partial product is equal to the multiplicand itself.

2. Write the partial products successively one below the other, shifting left each time by one bit position.
3. Sum all the partial products to produce the final product.

The product of two  $n$  digit numbers can be a number of  $2n$  digits atmost.

Digital implementation requires the following changes.

1. In manual working, we perform left shift on the subsequent partial product which is yet to be formed, but this kind of anticipatory job is not done by a physically realizable machine. A real machine can operate only on the existing operands but not on future results. Hence, we shift the partial product already formed to the right by one bit and add the next partial product in its normal position. This would produce the correct results as the reality positions of the operands for additions are as they should be.
2. Instead of forming all the partial products and then adding, which would require a large number of registers to store them, each partial product is added to register A (accumulator) and shifted right. This job is repeated  $n$  times where  $n$  is the number of bits in the multiplier.

With this background we can proceed to design the data processor subsystem first and the control subsystem next.

### 15.5.1 Datapath Subsystem for Binary Multiplier

Figure 15.18 shows the datapath subsystem for the binary multiplier. It comprises the following:

1. Register B to hold the multiplicand.
2. Register Q to hold the multiplier.
3. Register A, called the accumulator, to hold the cumulative sum of products.
4. A parallel adder circuit.

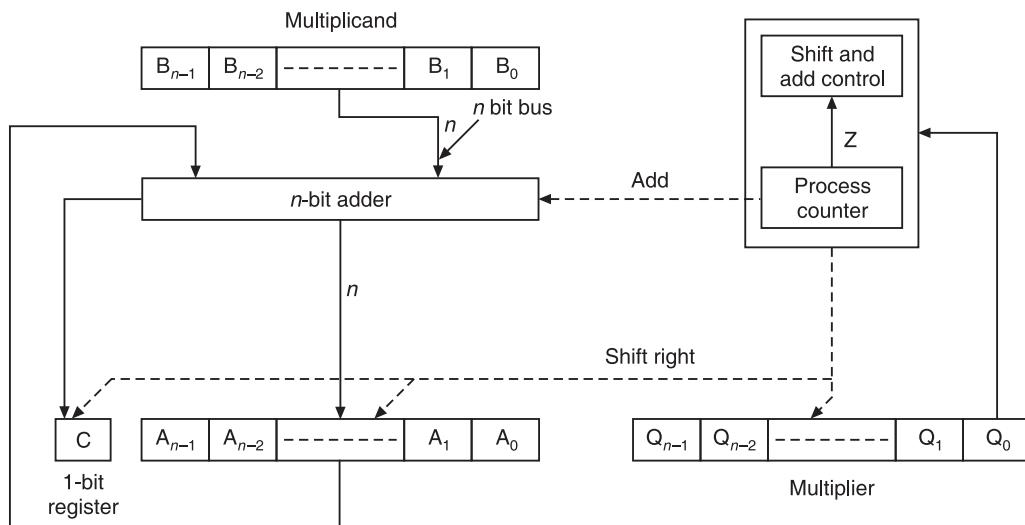


Figure 15.18 Datapath subsystem for binary multiplier.

5. One flip-flop C to hold the carry, if any, produced in addition.
6. A counter P which is initialized to the word length  $n$ .
7. Provision to decrement P to check for zero.
8. Provision to concatenate the registers C, A, Q to form a combined register of length  $1 + n + n = 2n + 1$  bits with facility to shift right.

Figure 15.19 illustrates how a digital computer performs multiplication using the add-shift algorithm. As an example, the operations involved and their results in the multiplication process to multiply 1101 with 1010 are shown. Initially the multiplicand is loaded into register B and the multiplier is loaded into register Q. C and A are cleared. The word length  $n$  is loaded into register P which acts as a counter. The least significant bit  $Q_0$  of the multiplier Q is sensed by the machine. If  $Q_0 = 1$ , then form a partial product by adding B to A, that is  $A \leftarrow A + B$  and then shift the combined register C A Q to the right by 1 bit position so that C goes into  $A_{n-1}$ , A<sub>0</sub> into  $Q_{n-1}$  and  $Q_0$  goes out. In place of  $Q_0$ , present  $Q_1$  becomes  $Q_0$  for the next iteration. If  $Q_0 = 0$ , then the partial product is 0 and hence there is no need to add, so only shift CAQ to the right. Repeat the process  $n$  times in a loop starting from  $(n - 1)$  and proceed to 0. After traversing through the loop, test the counter P. If it shows up 0, stop and exit from the loop. A flow chart for multiplication operation is shown in Figure 15.20.

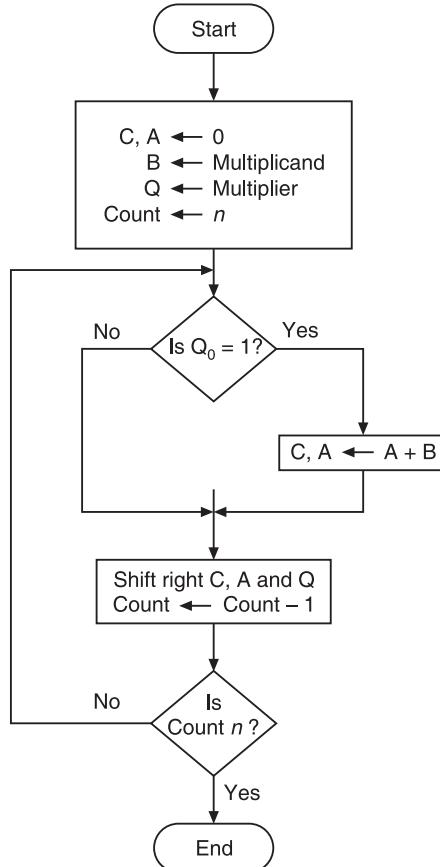
B	C	A	Q	Components	Count P
1 1 0 1	0	0 0 0 0	1 0 1 0	$B \leftarrow$ Multiplicand $Q \leftarrow$ Multiplier $A \leftarrow 0, C \leftarrow 0, P \leftarrow n$	100 (4)
1 1 0 1	0	0 0 0 0	1 0 1 0	$P \leftarrow P - 1$	011 (3)
	0	0 0 0 0	0 1 0 1	$Q_0 = 0$ CAQ shifted right	
1 1 0 1	0	1 1 0 1	0 1 0 1	$P \leftarrow P - 1$	010 (2)
	0	0 1 1 0	1 0 1 0	$Q_0 = 1, A \leftarrow A + B$ CAQ shifted right	
1 1 0 1	0	0 1 1 0	1 0 1 0	$P \leftarrow P - 1$	001 (1)
	0	0 0 1 1	0 1 0 1	$Q_0 = 0,$ CAQ shifted right	
1 1 0 1	1	0 0 0 0	0 1 0 1	$P \leftarrow P - 1$	000 (0)
	0	1 0 0 0	0 0 1 0	$Q_0 = 1, A \leftarrow A + B$ CAQ shifted right	

Figure 15.19 Flow chart for multiplication in a computer.

### 15.5.2 ASM Chart for Binary Multiplier

Figure 15.21 shows the ASM chart for binary multiplier. Initially the multiplicand is in B register and the multiplier in Q register. The multiplication process is initiated when S = 1. Register A and flip-flop C are cleared and the process counter P is set to a binary number  $n$  which is equal to the

number of bits in the multiplier. In the next step, the loop is executed to form the partial products. The multiplier bit in  $Q_0$  is checked, and if it is equal to 1, the multiplicand in B is added to the partial product in A. The carry from the addition is transferred to C. The partial product in A is left unchanged if  $Q_0 = 0$ . The process counter P is decremented by 1 regardless of the value of  $Q_0$ . Registers C, A and Q are combined into one composite register CAQ, which is then shifted right by 1 bit to obtain a new partial product.



**Figure 15.20** Flow chart for multiplication operation.

The value in the process counter is checked after the formation of each partial product. If the content of P is not zero, control input Z is equal to 0 and the process is repeated to form a new partial product. The looping process is stopped when the process controller, P reaches 0 and the control input Z is equal to 1. It is important to note that the partial product formed in A is shifted into Q one bit at a time and eventually replaces the multiplier. The final product is available in A and Q with A holding the most significant bits and Q the least significant bits.

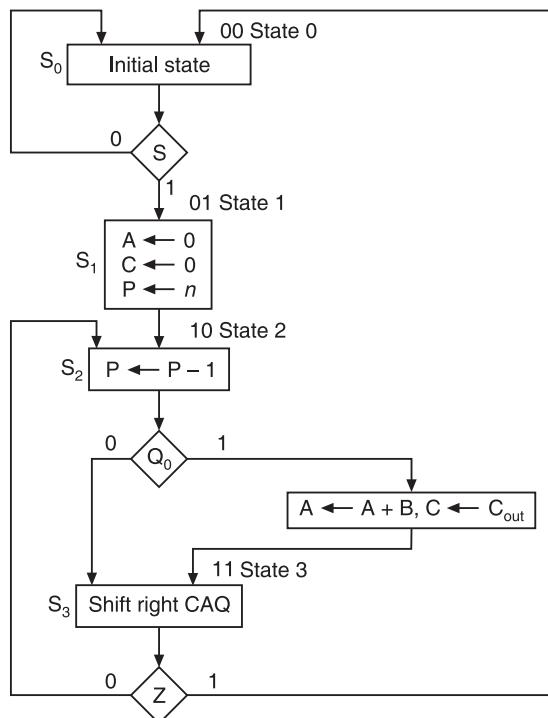


Figure 15.21 ASM chart for a binary multiplier.

### 15.5.3 Control Subsystem Using Logic Gates

From the ASM chart shown in Figure 15.21, we notice that the binary multiplier has four states and three inputs. The state table for the control subsystem of a binary multiplier is shown in Table 15.7. It is extremely simple to fill this table. Look at the present state PS ( $y_1 y_0$ ) and the next state NS ( $Y_1 Y_0$ ) columns first. Notice in the ASM chart that there are two transitions from the state 00. They are 00 to 00 (re-entry) and 00 to 01. Hence, provide two rows and enter the corresponding columns of PS and NS. Again from the ASM chart notice that the transition 00 to 00 occurs if  $S = 0$  and the other transition 00 to 01 occurs if  $S = 1$ . Make these entries in the S column. Other inputs for these transitions become don't care entries, shown by dashes.

For PS of 01, there is only one transition to state 10 in the ASM chart. Hence, only one row is provided and the columns PS and NS are filled accordingly. This transition has no input conditions and hence the columns of S,  $Q_0$ , Z become don't cares indicated by dashes.

For PS of 10, the transition to state 11 occurs in two paths depending on  $Q_0$ . If  $Q_0 = 0$ , then the machine goes straight to the state 11. If  $Q_0 = 1$ , then the machine has to add B to A and go to state 11. Although this could have been represented by one row, we show two rows for clarity as the value of  $Q_0$  is different. For PS of 11, there are two possible transitions—11 to 10 if  $Z = 0$  or 11 to 00 if  $Z = 1$ .

Now look at the output columns.  $T_0$  is equal to 1 for PS 00,  $T_1$  for PS 01,  $T_2$  for PS 10,  $T_3$  for PS 11. These are obtained by decoding the state variables  $y_1, y_0$ . Finally, the output  $Q_0 \cdot T_2$  is for the

purpose of enabling addition in state 10 as depicted by the oval block in the ASM chart. All these outputs eventually provide the control timing signals.

**Table 15.7** State table for the control subsystem

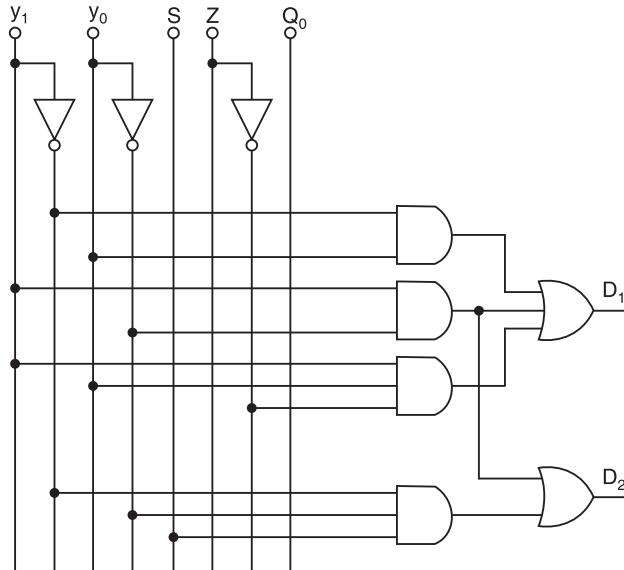
PS		Inputs			NS		Outputs (Timing signals)				
$y_1$	$y_0$	S	$Q_0$	Z	$Y_1$	$Y_0$	$T_0$	$T_1$	$T_2$	$T_3$	$Q_0 T_2$
0	0	0	—	—	0	0	1				
0	0	1	—	—	0	1	1				
0	1	—	—	—	1	0		1			
1	0	—	0	—	1	1			1		
1	0	—	1	—	1	1			1	1	
1	1	—	—	0	1	0			1		
1	1	—	—	1	0	0				1	

The control unit needs two flip-flops. If we choose D flip-flops, it is easy to get the following expressions for the excitations as functions of state variables  $y_1, y_0$  and the inputs S,  $Q_0, Z$ .

$$\begin{aligned} D_1 &= Y_1 = \bar{y}_1 y_0 + y_1 \bar{y}_0 \bar{Q}_0 + y_1 \bar{y}_0 Q_0 + y_1 y_0 \bar{Z} \\ &= \bar{y}_1 y_0 + y_1 \bar{y}_0 + y_1 y_0 \bar{Z} \end{aligned}$$

$$\begin{aligned} D_0 &= Y_0 = \bar{y}_1 \bar{y}_0 S + y_1 \bar{y}_0 \bar{S} + y_1 \bar{y}_0 S \\ &= \bar{y}_1 \bar{y}_0 S + y_1 \bar{y}_0 \end{aligned}$$

The structure of the control unit is indicated in Figure 15.22. The excitations for control subsystem are shown in Figure 15.23.



**Figure 15.22** Control subsystem using logic gates.

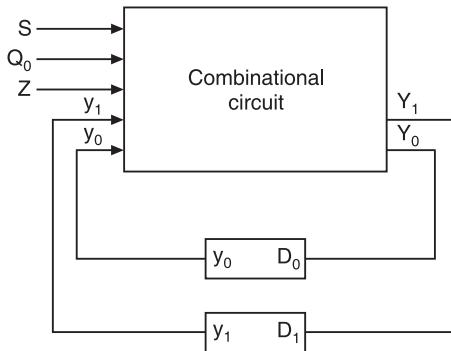


Figure 15.23 Excitations for control subsystem.

**Note:** The input  $Q_0$  nearly enables the adder in state 10 and hence it does not participate in the excitation functions. Nevertheless, it has to produce the control signals  $Q_0 T_2$  to enable the adder.

**PLA control:** Figure 15.24 shows the block diagram of a PLA control used for binary multiplication. The combinational circuit part is implemented using a PLA. The inputs to the PLA are the values of the present state of the flip-flops and the three control inputs. The outputs of the PLA provide the values for the next state in the flip-flops and the control output variables. For this circuit there is one output for each present state and an additional output for the conditional operation  $D = Q_0 T_2$ . When  $D$  input is 1, the conditional operation—add  $B$  to  $A$  is performed.

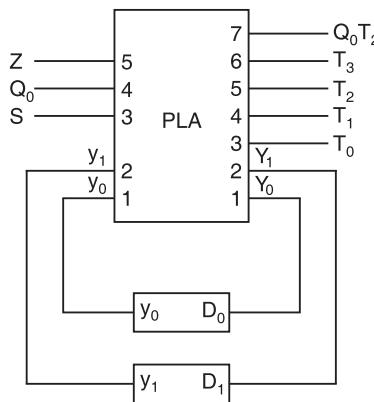


Figure 15.24 Block diagram of a PLA.

**PLA programming table:** In Chapter 8 we have discussed the internal organization of the PLA and how to obtain the PLA programming table. We can obtain the programming table for PLA using information given in the state table without any simplification. Table 15.8 shows the PLA programming table which lists the products, inputs and outputs. Dashes in the table are don't cares.

Note that

1. The don't cares in the input column of the state table indicate no connection for PLA.
2. The 0s in the output column of the state table indicate no connection to PLA.
3. No connection for PLA path is indicated by a dash (-) in the table. All other entries remain the same.

**Table 15.8** PLA programming table

Product term	Inputs					Outputs						
	S	Q <sub>0</sub>	Z	y <sub>1</sub>	y <sub>0</sub>	T <sub>0</sub>	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	Q <sub>0</sub> T <sub>2</sub>	Y <sub>1</sub>	Y <sub>0</sub>
	1	2	3	4	5	1	2	3	4	5	6	7
1	$\bar{y}_1 \bar{y}_0$	—	—	—	0	0	1					
2	$\bar{y}_1 y_0$	—	—	—	0	1		1				1
3	$y_1 \bar{y}_0$	—	—	—	1	0			1			1
4	$y_1 y_0$	—	—	—	1	1				1		
5	$y_0 \bar{Z}$	—	—	0	—	1						1
6	$\bar{y}_0 S$	1	—	—	—	0						1
7	$y_1 \bar{y}_0 Q_0$	—	1	—	1	0					1	

**Multiplexer control:** If we wish to design a multiplexer control, we need to have two MUXes, one each for the state variables  $y_1, y_0$ . The present state variables will be fed to address (select) inputs. Each MUX must have four inputs corresponding to the number of states. The inputs of the multiplexer are determined from the decision boxes and state transitions given in the ASM chart of the binary multiplier (Figure 15.21). The present states, next states and conditions for the transition can be tabulated for the ASM chart as shown in the transition Table 15.9.

**Table 15.9** Transition table

	Present state		Next state		Condition of transition
	y <sub>1</sub>	y <sub>0</sub>	Y <sub>1</sub>	Y <sub>0</sub>	
$S_0$	0	0	$S_0$	0	0
			$S_1$	0	1
$S_1$	0	1	$S_2$	1	0
$S_2$	1	0	$S_3$	1	1
$S_3$	1	1	$S_2$	1	0
			$S_0$	0	1

**Table 15.10** Inputs for multiplexers

MUX 1	MUX 2
$0 \rightarrow 0$	$0 \rightarrow S$
$1 \rightarrow 1$	$1 \rightarrow 0$
$2 \rightarrow 1$	$2 \rightarrow 1$
$3 \rightarrow \bar{Z}$	$3 \rightarrow 0$

The multiplexer inputs (Table 15.10) can be determined by including the condition of transition corresponding to logic 1 bit position in the next state.

Consider the transition from  $S_0$  to  $S_1$ . For flip-flop 1 the next state is 0, hence the corresponding input of multiplexer 1 is 0. For flip-flop 2 the next state is 1, hence the corresponding input of multiplexer is the condition of transition corresponding to  $S_1$ , i.e. S.

Consider the transition from  $S_1$  to  $S_2$ . For flip-flop 1 the next state is 1, hence the corresponding input of the multiplexer is the condition of transition corresponding to  $S_2$ , i.e. 1. For flip-flop 2 the next state is 0, hence the corresponding input of multiplexer 2 is 0.

Consider the transition from  $S_2$  to  $S_3$ . For both flip-flops 1 and 2, the next state is 1. Hence the corresponding input of both the multiplexers is the condition corresponding to  $S_3$ , i.e. 1.

Consider the transition from  $S_3$  to  $S_2$ . For flip-flop 1, the next state is 1. Hence the corresponding input of the multiplexer is the condition of transition corresponding to  $S_2$ , i.e.  $\bar{Z}$ . For flip-flop 2 the next state is 0, hence the corresponding input of multiplexer 2 is 0.

The multiplexer control of the binary multiplier is shown in Figure 15.25.

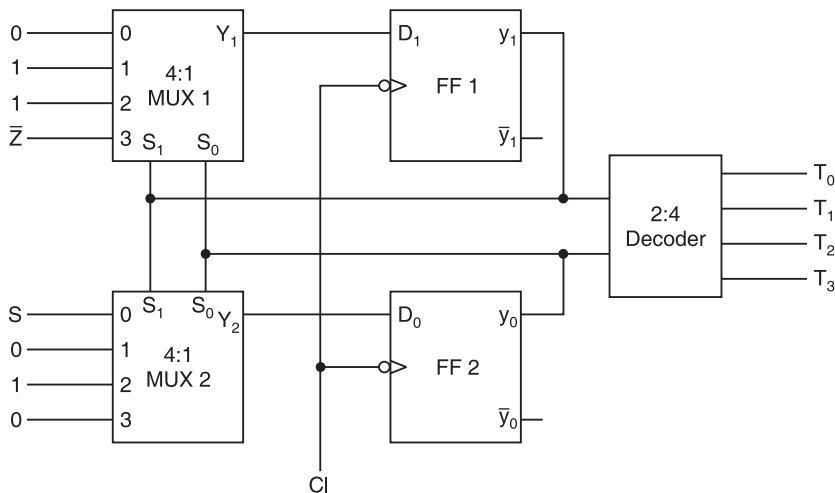


Figure 15.25 Multiplexer control of binary multiplier.

## 15.6 ASM FOR WEIGHING MACHINE

In the algorithm for tabular minimization of Boolean expressions, we have to arrange the minterms in the ascending order of their weights. This is only one of the many situations when we have to examine the 1s of a given binary word. The weight of a binary number is defined as the number of 1s present in its binary representation.

Here we want to design a sequential machine which will calculate the weight of a given binary number or in other words compute the number of 1s in a given binary number. Let the digital system (weighing machine) contain a register R to store the binary number whose weight is to be determined, another register W which acts as a counter to determine the weight of the given binary number and 1 flip-flop. Initially the number has to be loaded into register R and W has to be initialized. Then shift each bit of register R into the flip-flop F, one at a time. Sense the value of F;

whenever it is 1, the register W is incremented by one. At the end W contains the weight of the word.

The ASM chart for the weighing machine is shown in Figure 15.26. It has three inputs S (start), Z (zero), and F (flip-flop) and four states  $S_0$ ,  $S_1$ ,  $S_2$  and  $S_3$ . S is the start input ( $S = 1$  starts the weight computation process), Z is for sensing all zeros in R ( $Z = 1$  indicates all zeros in R), and the value of F decides whether W is to be incremented or not ( $F = 1$  indicates that W has to be incremented).

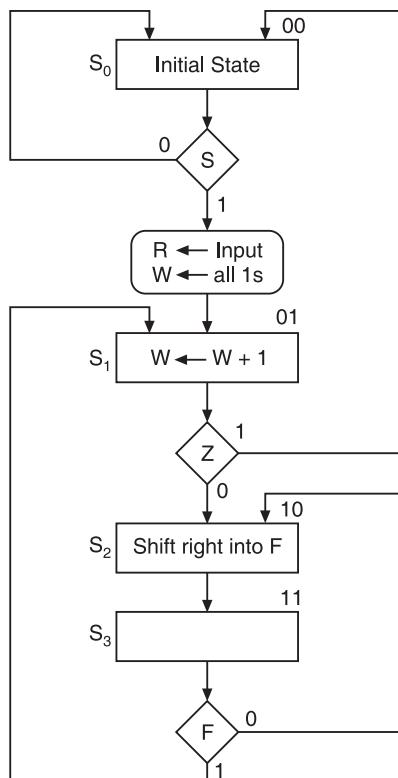


Figure 15.26 ASM chart for weighing machine.

*State  $S_0$ :* Initially the weighing machine is in state  $S_0$ . The weighing process starts when start (S) signal becomes 1. While in state  $S_0$ , if S is 1, the clock pulse causes three jobs to be done simultaneously:

1. Binary number is loaded into register R.
2. W register is set to all 1s.
3. The machine is transferred to state  $S_1$ .

*State  $S_1$ :* While in state  $S_1$ , the clock pulse causes two jobs to be done simultaneously:

1. Counter W is incremented by 1(in the first round, all 1s become all 0s).
2. If Z is 0, the machine goes to the state  $S_2$ ; if Z is 1, the machine goes to state  $S_0$ .

If  $Z$  is 0, it means that the weight of the word loaded into register  $R$  is 1 or more, and the machine has to go to  $S_2$  to continue the process. If  $Z$  is 1, it means that the word  $R$  contains in it all zeros and hence the weight of word is already indicated by  $W$ . In this case, the machine should go back to the initial state  $S_0$  having completed the task of computing the weight of the given word. The machine reaches this state again from  $S_3$ , if the value of  $F$  is sensed as 1.

*State  $S_2$ :* In this state, register  $R$  is shifted right by 1 bit so that LSB goes into  $F$  and MSB is loaded with 0.

*State  $S_3$ :* In this state, the value of  $F$  is checked. If it is 0, the machine is transferred to the state  $S_2$ , otherwise the machine is transferred to state  $S_1$ . Thus, when  $F = 1$ ,  $W$  is incremented.

All the operations occur in coincidence with the clock pulse while in the corresponding state. Also notice that the register  $R$  should eventually contain all 0s when the last 1 is shifted into it.

### 15.6.1 System Design

The system (also referred to as a machine) consists of two subsystems—data processor subsystem and control subsystem. The data processor subsystem performs the tasks prescribed for each state of the ASM block. The control subsystem ensures the proper sequence of states and transitions depending on the inputs and the present state of the machine.

### 15.6.2 Datapath Subsystem

Figure 15.27 shows the datapath subsystem for the weighing machine. It shows the registers, flip-flop, control subsystem circuit for zero checking, and connections of control signals which controls the operation of these. The control subsystem produces the signals  $T_0$ ,  $T_1$ ,  $T_2$  and  $T_3$  when the machine is in states  $S_0$ ,  $S_1$ ,  $S_2$  and  $S_3$  respectively. ( $T_0 = 1$  when the machine is in state  $S_0$ ;  $T_1 = 1$  when the machine is in state  $S_1$  and so on). These signals are used to activate desired operations in their respective states.

Initially, the machine will be in state  $S_0$ , that is  $T_0$  is 1. For starting the process, make the input  $S$  equal to 1. The clock pulse occurring while in  $S_0$  loads the input word parallelly into register  $R$  and sets the register  $W$  to all 1s at the same time. The same pulse takes the machine to state  $S_1$  and  $T_1$  becomes 1.

While in state  $S_1$ , the clock pulse increments  $W$  and makes it 0 to begin with. The same pulse senses  $Z$ . If  $Z = 1$ , then it would mean that the word in  $R$  contains all 0s and so the same clock pulse takes the machine to the initial state  $S_0$  as the weight of the given word is already indicated by  $W$  and the process ends. If  $Z$  is sensed as 0 during  $T_1$ , then it means that the binary word contains one or more 1s in it. So the process has to continue and the control ensures that the same clock pulse takes the machine to state  $S_2$  and  $T_2$  becomes 1.

While in state  $S_2$ , ( $T_2 = 1$ ), the clock pulse shifts  $R$  right by one bit so that LSB of  $R$  appears in  $F$  and 0 is shifted into the MSB place. The same clock pulse takes the machine to state  $S_3$  and  $T_3$  becomes 1.

During  $T_3$ ,  $F$  is sensed by the control circuit. If  $F = 1$ , the clock pulse takes the machine to  $S_1$  and  $T_1$  becomes 1. If  $F$  is 0, the same clock pulse takes the machine to  $S_2$  and  $T_2$  becomes 1. The process is repeated until all the bits of  $R$  are sensed or  $Z$  becomes 1, whichever is earlier.

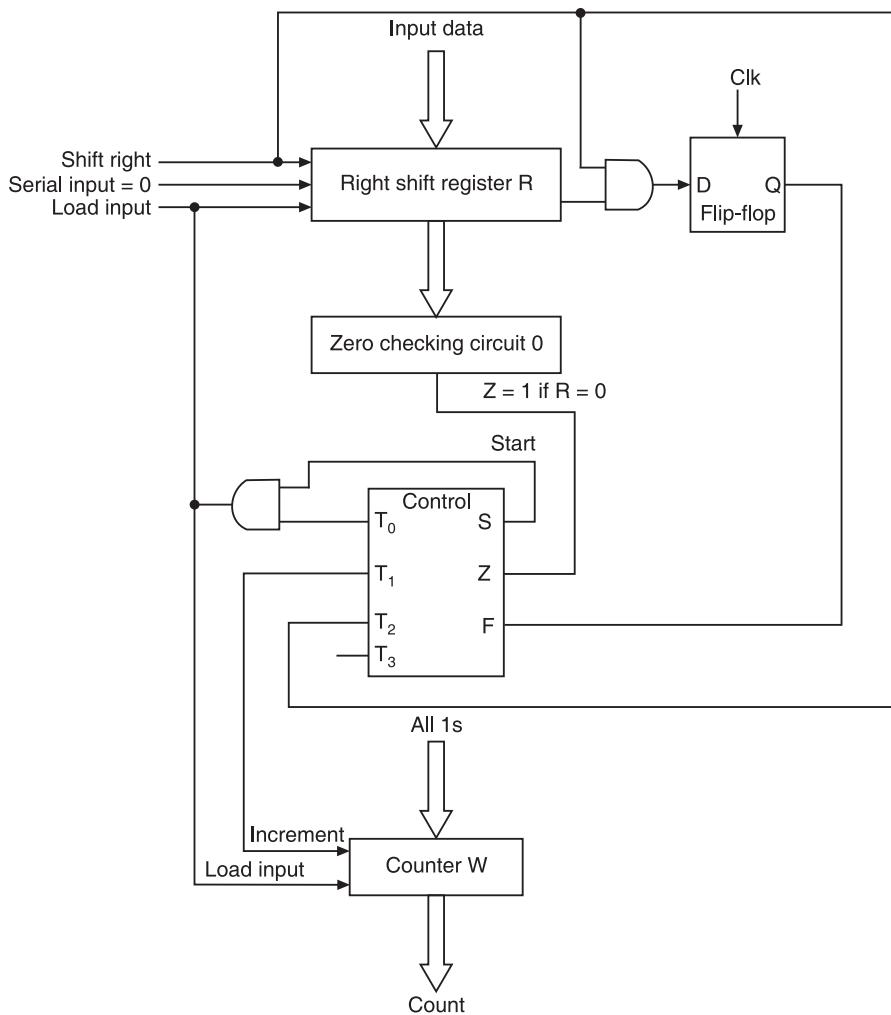


Figure 15.27 Datapath subsystem for weighing machine.

### 15.6.3 Control Subsystem Implementation

The weighing machine system requires four states. So two state variables  $y_1, y_0$  are required to provide these four states. By decoding the state variables, one can obtain the control timing signals  $T_0, T_1, T_2$  and  $T_3$ .

Let  $Y_1$  and  $Y_0$  be the next state variables which depend on the inputs  $S, Z, F$  and the present state variables  $y_1, y_0$ . Let us use D flip-flops. Then the excitations  $D_1$  and  $D_0$  are  $D_1 = Y_1$  and  $D_0 = Y_0$ . To design the control subsystem, we have to obtain the minimal expressions for  $D_1$  and  $D_0$  in terms of  $y_1, y_0, S, Z, F$ . Normally, it requires a five-variable map, instead we may as well work with the state table shown in Table 15.11.

**Table 15.11** State table

Present state		Inputs			Next state		Flip-flop inputs	
$y_1$	$y_0$	S	Z	F	$y_1$	$y_0$	$D_1 = Y_1$	$D_0 = Y_0$
0	0	0	—	—	0	0	0	S
		1	—	—	0	1		
0	1	—	1	—	0	0	$\bar{Z}$	0
	1	0	—	—	1	1		
1	1	—	—	1	0	1	$\bar{F}$	F

The state table shown is a modified state table. In this modified state table, we use one row for each transition. It consists of many don't cares. The values for  $D_1$  and  $D_0$  inputs of the two flip-flops are assigned as follows.

Row 0: Row 0 shows that in the next state, the output of flip-flop 1 is always 0, hence, we can put 0 in the column  $D_1$  of row 0. In the next state, the output of flip-flop 2 follows the input S ( $Y_0 = 0$ , when  $S = 0$ ;  $Y_0 = 1$ , when  $S = 1$ ), hence, we can put S in the column  $D_0$  of row 0.

Row 1: Row 1 shows that in the next state, the output of flip-flop 1 follows the complement of input Z, hence, we can put  $\bar{Z}$  in the column  $D_1$  of row 1. In the next state, the output of flip-flop 2 is always 0, hence, we can put 0 in the column  $D_0$  of row 1.

Row 2: Row 2 shows that in the next state, the outputs of both the flip-flops are logic 1, hence, we can put 1 in the column  $D_1$  and  $D_0$  of row 2.

Row 3: Row 3 shows that, in the next state, the output of flip-flop 1 follows the complement of input F, hence we can put  $\bar{F}$  in the column  $D_1$  of row 3. In the next state, the output of flip-flop 2 follows the input F, hence, we can put F in the column  $D_0$  of row 3.

Since the control signals  $T_0$ ,  $T_1$ ,  $T_2$  and  $T_3$  correspond to states  $S_0$ ,  $S_1$ ,  $S_2$  and  $S_3$ , i.e. to row 0, row 1, row 2 and row 3, the expressions for the inputs of flip-flop 1 and flip-flop 0 are given by

$$\begin{aligned} D_1 &= T_0 \cdot 0 + T_1 \cdot \bar{Z} + T_2 \cdot 1 + T_3 \cdot \bar{F} \\ &= T_1 \bar{Z} + T_2 + T_3 \bar{F} \\ D_0 &= T_0 \cdot S + T_1 \cdot 0 + T_2 \cdot 1 + T_3 \cdot F \\ &= T_0 S + T_2 + T_3 F \end{aligned}$$

**Control subsystem implementation using conventional hardware:** Based on the above expressions for  $D_1$  and  $D_0$ , a logic diagram can be drawn as shown in Figure 15.28.

**Control subsystem implementation using multiplexer control:** Use of multiplexers in the control circuit makes the design much more elegant and helps the designer to conceive how the states are changing with every clock pulse. The strategy is to use one multiplexer for each excitation. The present state variables, that is, outputs of the flip-flops are connected to the select (address) inputs of MUXs as shown in Figure 15.29. The weighing machine has four states realized by two D flip-flops,  $D_1$  and  $D_0$ . The outputs of MUXs feed the flip-flops and the MUX will have as many inputs as there are states in the machine. These are actually next state variables. One of these is chosen to follow to the output depending on the present state fed to the select inputs.

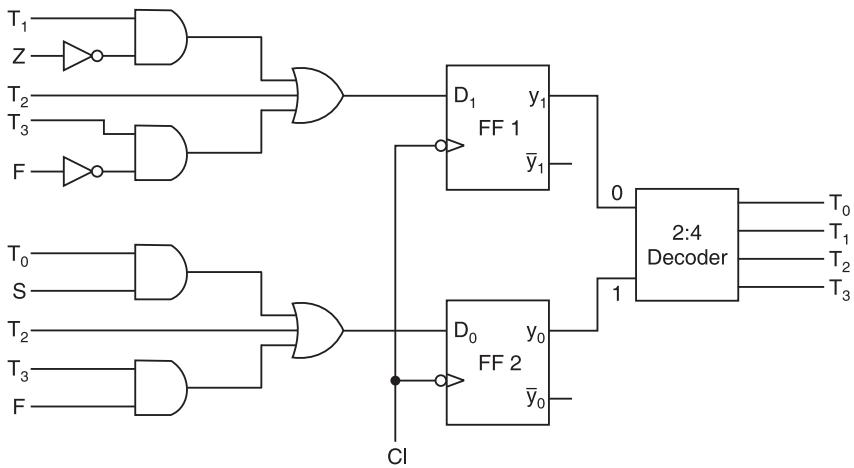


Figure 15.28 Logic circuit.

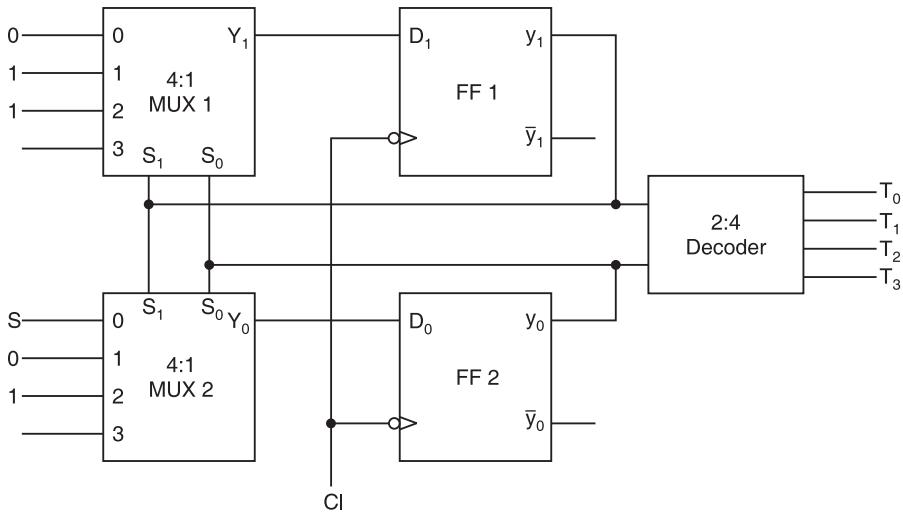


Figure 15.29 Logic circuit using multiplexer control.

**Control subsystem implementation using PLA control:** Besides the conventional hardware control or the multiplexer control, there is a third option using a PLA for designing the control subsystem. Figure 15.30 shows the PLA control block. The state variables  $y_1$ ,  $y_0$  and the inputs S, Z, F become the inputs numbered as 1 through 5. The excitations  $D_1$ ,  $D_0$  for the flip-flops (the same as next state variables  $Y_1$ ,  $Y_0$ ) and the control signals  $T_0$ ,  $T_1$ ,  $T_2$ ,  $T_3$  become the outputs of the PLA. The PLA programming table is shown in Table 15.12. It consists of product terms, inputs and outputs. The product terms are listed from the logic expressions of  $D_1$  and  $D_0$ .

Each row corresponds to a product term. The columns represent the state variables, inputs to the machine, inputs to the flip-flops and outputs of the machine. The machine will be in state  $S_0$  when  $T_0 = 1$ , in  $S_1$  when  $T_1 = 1$ , in  $S_2$  when  $T_2 = 1$ , and in  $S_3$  when  $T_3 = 1$ . In each row, the input

variables contained in the product are marked 0 or 1, depending on whether the variables appear in a complemented form or uncomplemented form. Each output column represents a sum of products and is hence marked 1 in the corresponding row.

Table 15.12 PLA programming table

Product terms	Inputs					Outputs					
	$y_0$ 1	$y_1$ 2	S 3	Z 4	F 5	$D_0 = Y_0$ 1	$D_1 = Y_1$ 2	$T_0$ 3	$T_1$ 4	$T_2$ 5	$T_3$ 6
$T_0 = \bar{y}_1 \bar{y}_0$	1	0	0	—	—	—	—	—	—	—	1
$T_1 = \bar{y}_1 y_0$	2	1	0	—	—	—	—	—	—	—	1
$T_2 = y_1 \bar{y}_0$	3	0	1	—	—	—	—	1	1	—	1
$T_3 = y_1 y_0$	4	0	1	—	—	—	—	—	—	—	1
$T_4 = \bar{y}_1 y_0 \bar{z}$	5	1	0	—	0	—	—	1	—	1	1
$T_5 = y_1 y_0 \bar{F}$	6	1	1	—	—	0	—	1	—	—	1
$T_6 = \bar{y}_1 \bar{y}_0 S$	7	0	0	1	—	—	—	1	—	—	1
$T_7 = y_1 y_0 F$	8	1	1	—	—	1	1	—	—	—	1

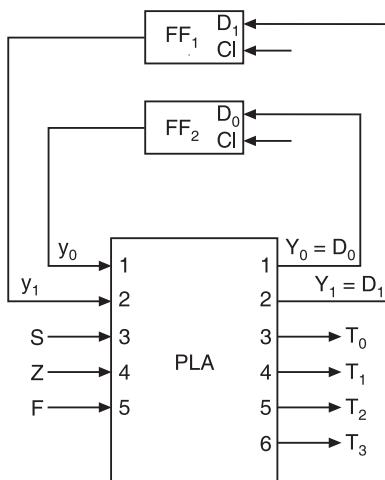


Figure 15.30 PLA control block.

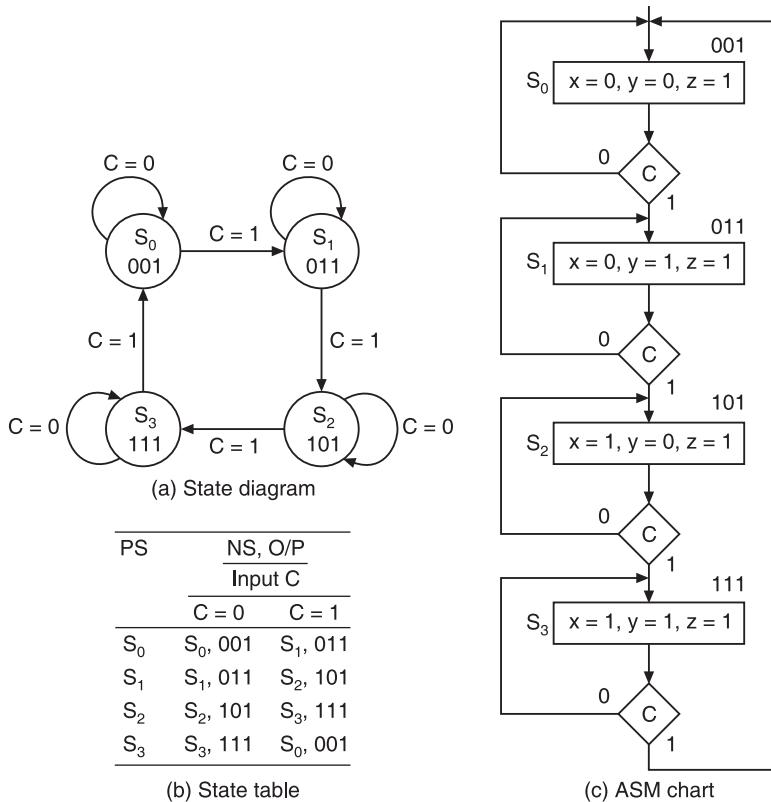
**EXAMPLE 15.5** Draw an ASM chart, state diagram and state table for the synchronous circuit having the following description:

The circuit has control input C, clock, and outputs x, y, z.

- (a) If  $C = 1$ , on every clock rising edge, the code on the output x, y, z changes from 001  $\rightarrow$  011  $\rightarrow$  101  $\rightarrow$  111  $\rightarrow$  001 and repeats.
- (b) If  $C = 0$ , the circuit holds the present state.

**Solution**

The outputs  $x, y, z$  represent the state of the circuit. So  $x, y, z$  are the present state variables. Let  $X, Y, Z$  be the next state variables. Let the states of the sequential circuit be  $S_0 = 001$ ,  $S_1 = 011$ ,  $S_2 = 101$  and  $S_3 = 111$ . The state diagram and the state table of the synchronous circuit are shown in Figures 15.31a and b respectively. The corresponding ASM chart is shown in Figure 15.31c.

**Figure 15.31** Example 15.5.

**EXAMPLE 15.6** Draw the state diagram, state table, and ASM chart for a 2-bit binary counter having one enable line E such that  $E = 1$  counting enabled, and  $E = 0$  counting disabled.

**Solution**

The given synchronous circuit is a 2-bit counter. So it has four states  $S_0 = 00$ ,  $S_1 = 01$ ,  $S_2 = 10$  and  $S_3 = 11$ . When enabled, i.e. when  $E = 1$ , it goes from one state to the next state in sequence. When disabled, i.e. when  $E = 0$ , it goes to the starting state from any present state. The state diagram and the state table of the 2-bit binary counter are shown in Figures 15.32a and b respectively. The corresponding ASM chart is shown in Figure 15.32c.

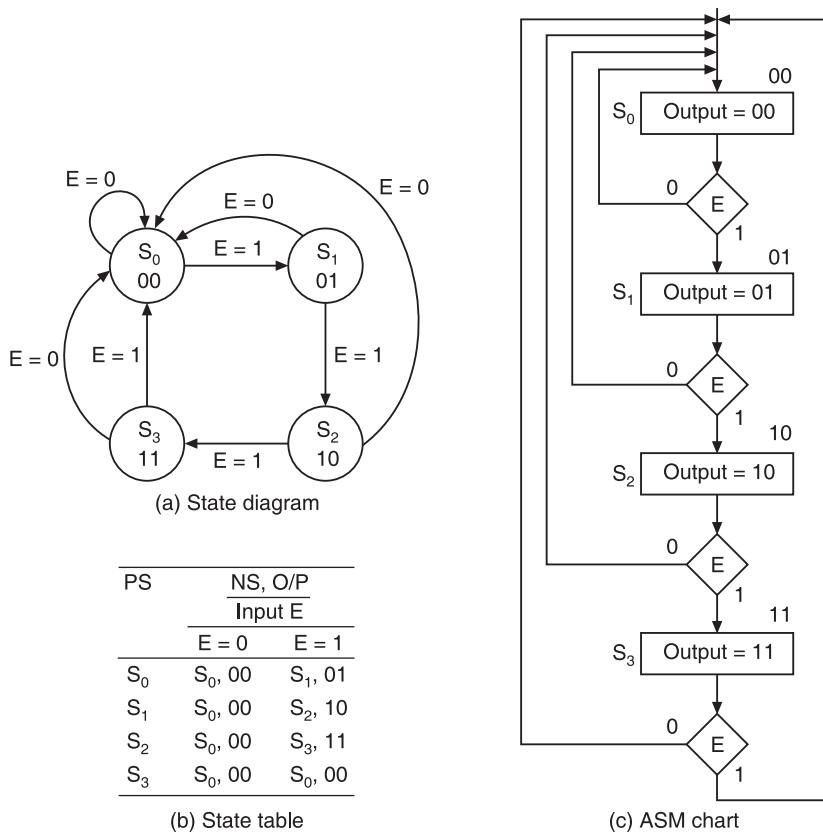


Figure 15.32 Example 15.6.

**EXAMPLE 15.7** Draw an ASM chart and state table for a 2-bit up-down counter having mode control input.

$M = 1$ : Up counting

$M = 0$ : Down counting

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

### Solution

The given synchronous circuit is a 2-bit up-down counter. It counts up, i.e. 00, 01, 10, 11, 00, ... when  $M = 1$ , counts down, i.e. 00, 11, 10, 01, 00, ... when  $M = 0$ . It outputs a 1 only when the state is 00 or 11. The counter has four states. The states are A = 00, B = 01, C = 10 and D = 11. The state diagram, and the state table of the 2-bit up-down counter are shown in Figures 15.33a and b respectively. The corresponding ASM chart is shown in Figure 15.33c.

**EXAMPLE 15.8** Draw the state diagram, state table and ASM chart for a sequence detector to detect the sequences 1111 and 0000. It has to output a 1 when the sequence is detected. Overlapping is not permitted.

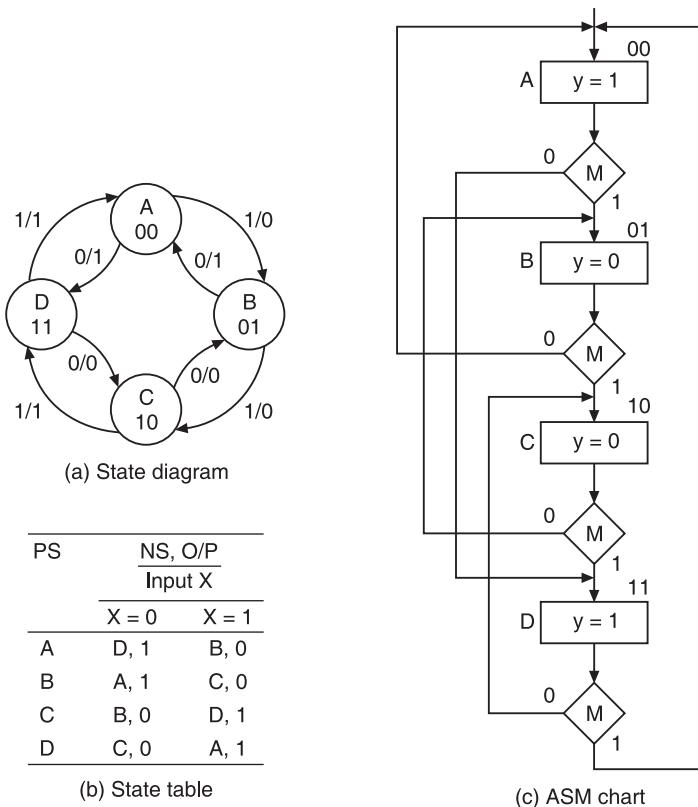


Figure 15.33 Example 15.7.

**Solution**

A sequence detector is a single input-single output sequential circuit. It has one external input, let us say X in addition to the clock and has one output Z. The output Z will be 1 only

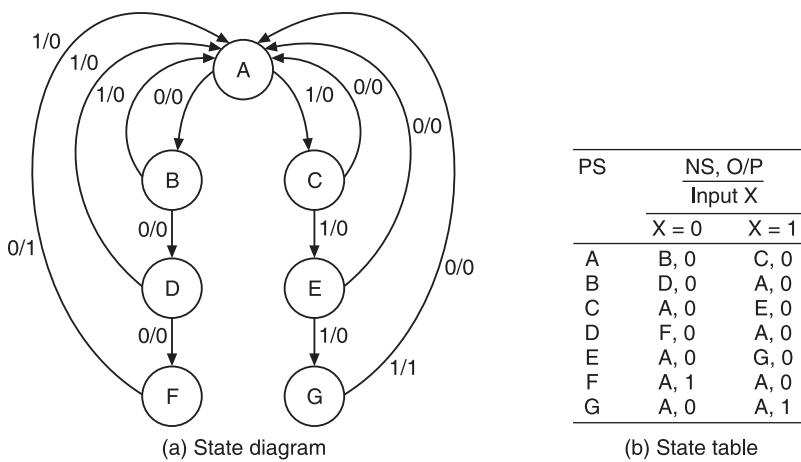


Figure 15.34 Example 15.8.

when four consecutive 1s or 0s are inputted. All other times  $Z$  will be 0. When  $Z = 1$ , i.e. the sequence is detected, the machine will go to the starting state, i.e. no overlapping is permitted. The state diagram and the state table of the sequence detector are shown in Figures 15.34a and b respectively. The corresponding ASM chart is shown in Figure 15.35.

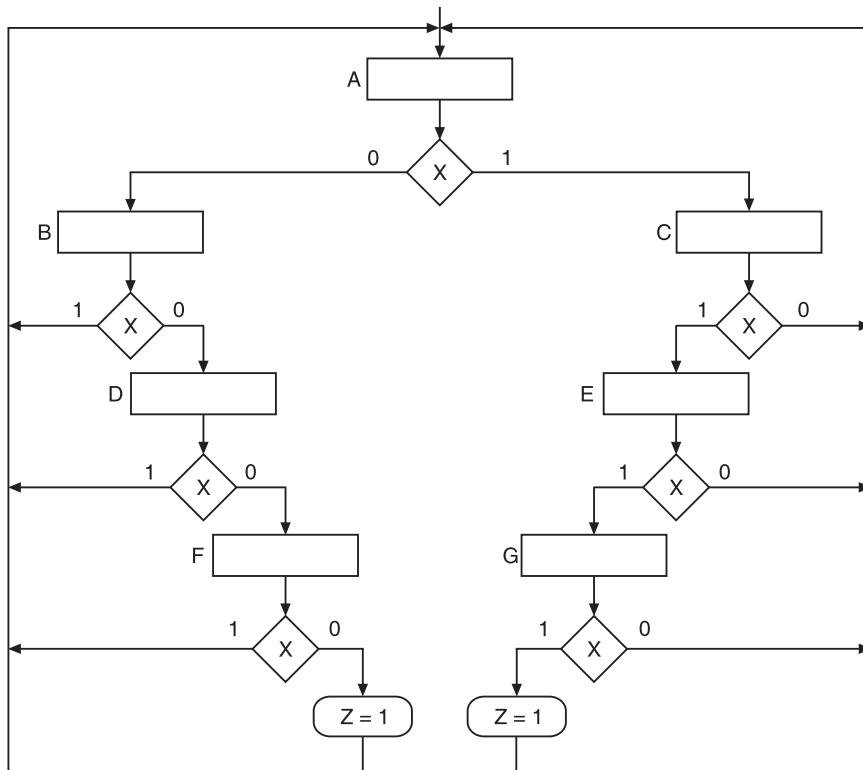


Figure 15.35 Example 15.8: ASM chart.

**EXAMPLE 15.9** Draw an ASM chart and state diagram for the circuit shown in Figure 15.36.

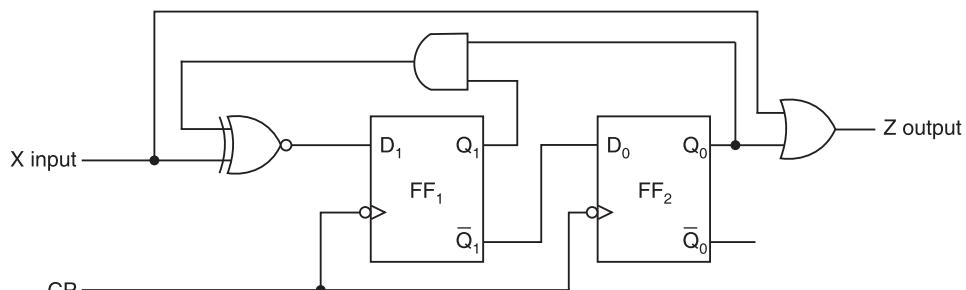


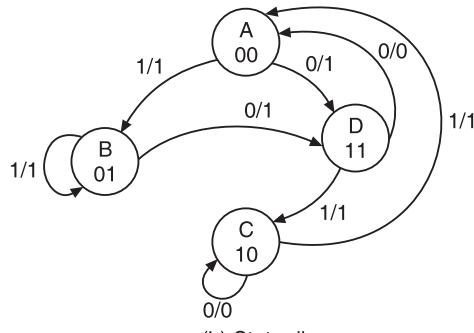
Figure 15.36 Example 15.9: Circuit diagram.

**Solution**

It is easier to draw the state diagram and the ASM chart from the state table. There are two flip-flops, so two state variables and four states will be there. Let the states be A(00), B(01), C(10) and D(11). The state table for the given circuit is shown in Figure 15.37a. The state diagram based on the state table is shown in Figure 15.37b.

PS	I/P	NS	O/P		
$y_1$	$y_0$	X	$y_1$	$y_0$	Z
0	0	0	1	1	1
0	0	1	0	1	1
0	1	0	1	1	1
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	0	0	1
1	1	0	0	0	0
1	1	1	1	0	1

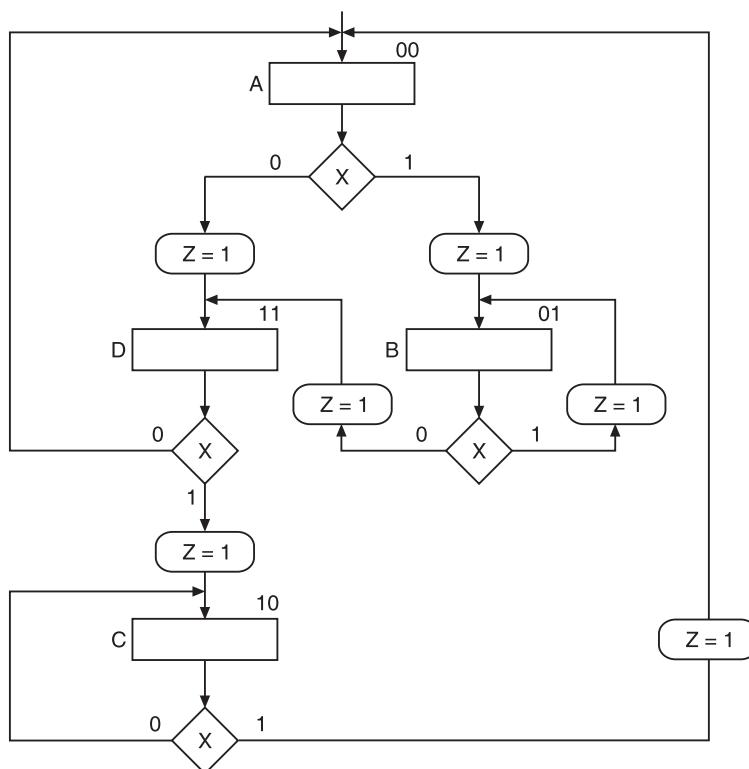
(a) State table



(b) State diagram

**Figure 15.37** Example 15.9.

From the state diagram, the ASM chart can be drawn as shown in Figure 15.38.

**Figure 15.38** Example 15.9: ASM chart.

**EXAMPLE 15.10** Draw the state diagram, state table and ASM chart for a D flip-flop.

**Solution**

A D flip-flop has two states A(0) and B(1). In one state (A), it stores a 0 and in the other state (B) it stores a 1. Its output is same as the present state which is equal to the input after time delay. While in state 0, it may receive the D input as 0 or 1. If the input D = 0, the flip-flop will remain in the same state and outputs a 0. If the input D = 1, the flip-flop will go to state 1 and outputs a 1. While in state 1, it may again receive the input as 0 or 1. If it is 0, it goes to state 0 and outputs a 0. If it is 1, it remains in the same state and outputs a 1.

The state diagram, and the state table of the D flip-flop are shown in Figures 15.39a and b respectively. The ASM chart of the D flip-flop is shown in Figure 15.39c. The circuit will have two state boxes for states 0 and 1 and two decision boxes controlled by D input.

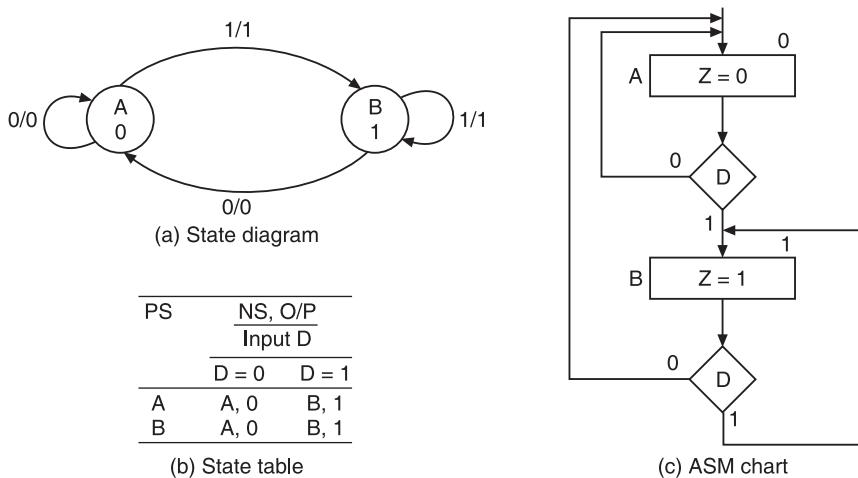


Figure 15.39 Example 15.10.

**EXAMPLE 15.11** Draw the state diagram, state table and ASM chart for a J-K flip-flop.

**Solution**

We know that the J-K flip-flop has two states A(0) and B(1). While at A, if J-K = 00 or J-K = 01, it remains in state A itself and outputs a 0. If J-K = 10 or J-K = 11, it goes to state B and outputs a 1. While at B, if J-K = 00 or J-K = 10, it remains in state B itself and outputs a 1. If J-K = 01 or J-K = 11, it goes to state A and outputs a 0. The state diagram and the state table of a J-K flip-flop are shown in Figures 15.40a and b respectively. The ASM chart of the J-K flip-flop is shown in Figure 15.40c.

**EXAMPLE 15.12** Design a ROM-based controller for the ASM chart shown in Figure 15.41. Tabulate the contents of ROM.

**Solution**

In the given ASM chart, there are four state boxes. So the machine has got four states A, B, C and D. Therefore, two flip-flops are required. There are four input signals X, Y, Z and W.

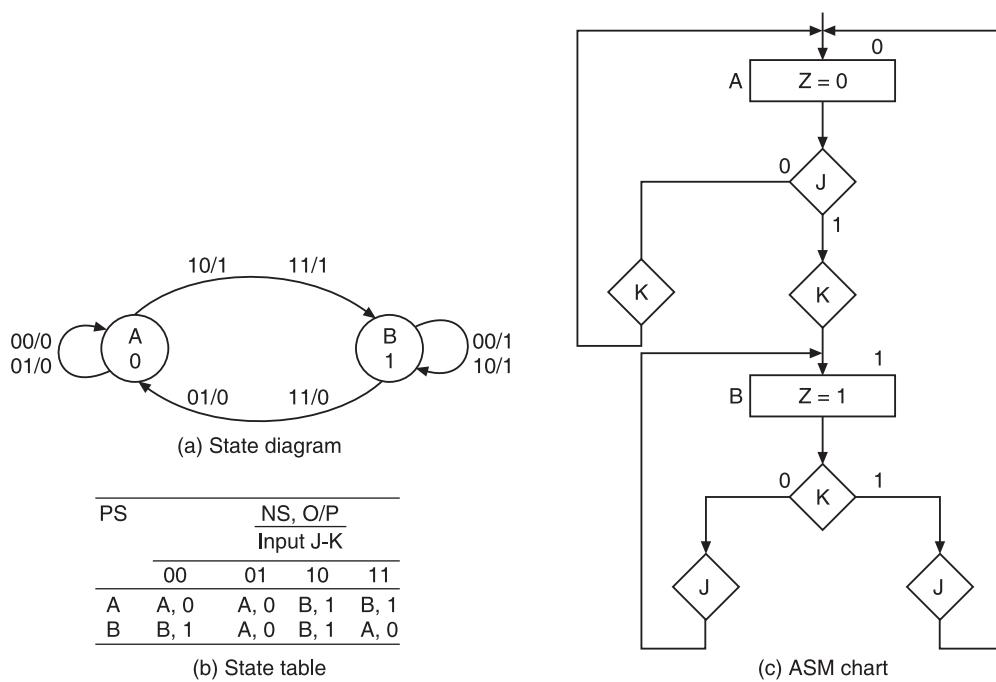


Figure 15.40 Example 15.11.

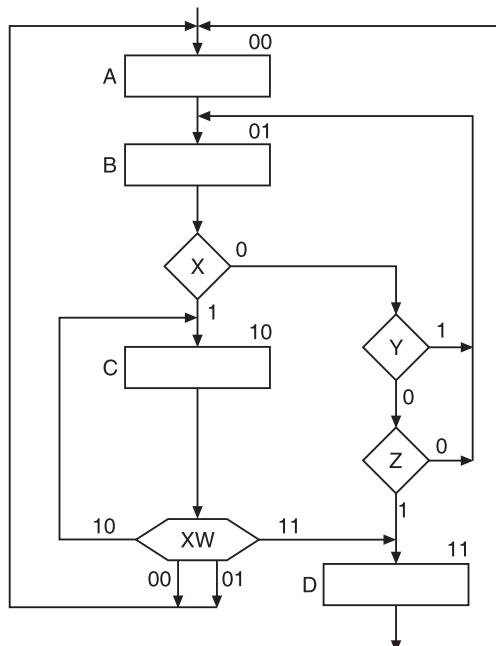


Figure 15.41 Example 15.12: ASM chart.

**ROM selection:** The controller should have a ROM of size  $2^n \times m$ , where

$$n = \text{number of inputs} + \text{number of flip-flops}$$

$$= 4 + 2 = 6 \text{ and}$$

$$m = \text{number of outputs} + \text{number of flip-flops}$$

$$= 0 + 2 = 2$$

**Logic diagram:** The logic diagram of the ROM-based controller using D flip-flops is shown in the Figure 15.42. The outputs (present states) of the flip-flops are fed to the ROM and the inputs to the flip-flops (next states) are fed from the ROM.

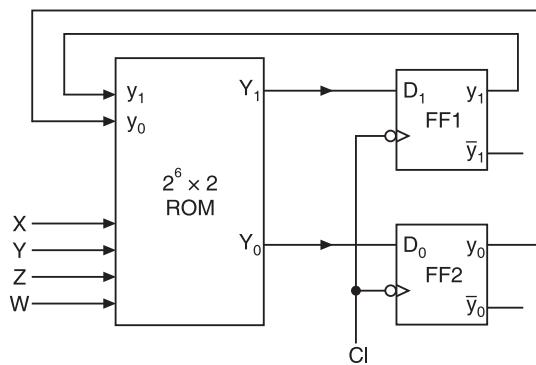


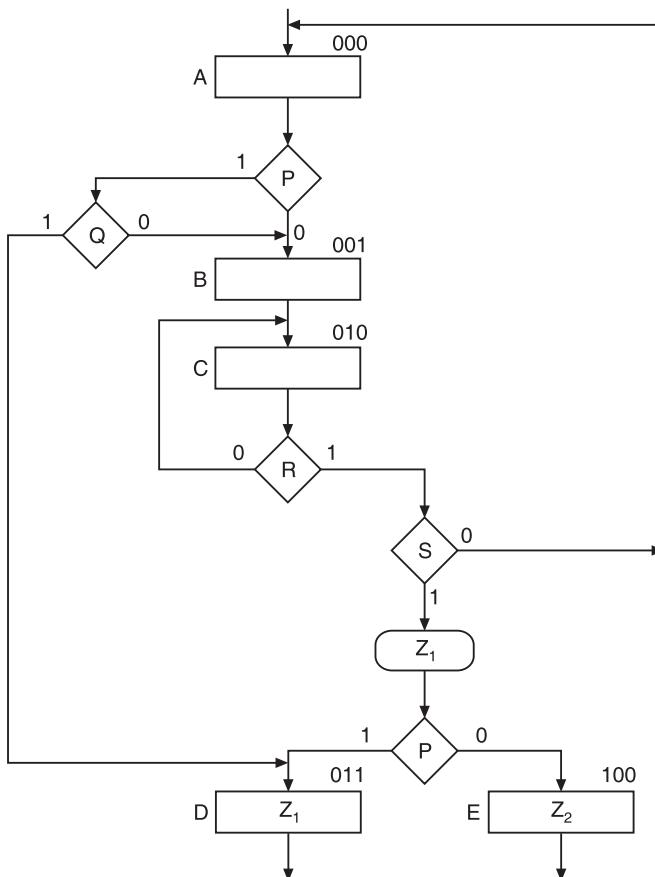
Figure 15.42 Example 15.12: ROM-based controller.

**ROM tabulation:** The ROM tabulation is shown in Table 15.13. The dashes in the table indicate that the corresponding inputs are don't cares. For example all dashes in inputs of row 1 indicate that when the machine is in state A, the next pulse takes it to state B independent of the values of the inputs X, Y, Z and W.

Table 15.13 Example 15.12: ROM tabulation

PS		Inputs				NS	
y <sub>1</sub>	y <sub>0</sub>	X	Y	Z	W	Y <sub>1</sub>	Y <sub>0</sub>
0	0	—	—	—	—	0	1
0	1	0	0	0	—	0	1
0	1	0	0	1	—	1	1
0	1	0	1	—	—	0	1
0	1	1	—	—	—	1	0
1	0	0	—	—	0	0	0
1	0	0	—	—	1	0	0
1	0	1	—	—	0	1	0
1	0	1	—	—	1	1	1
1	1	—	—	—	—	0	0

**EXAMPLE 15.13** Design a ROM-based controller for the ASM chart shown in Figure 15.43. Tabulate the contents of ROM.



**Figure 15.43** Example 15.13: ASM chart.

### Solution

In the ASM chart, there are five state boxes. So the machine has five states A(000), B(001), C(010), D(011) and E(100). Therefore, three flip-flops are required. But three flip-flops can have eight states. So the remaining three states are invalid. There are four inputs P, Q, R and S and two outputs  $Z_1$  and  $Z_2$ .

**ROM selection:** The controller should have a ROM of size  $2^n \times m$ , where

$$\begin{aligned} n &= \text{number of inputs} + \text{number of flip-flops} \\ &= 4 + 3 = 7 \text{ and} \end{aligned}$$

$$\begin{aligned} m &= \text{number of outputs} + \text{number of flip-flops} \\ &= 2 + 3 = 5 \end{aligned}$$

**Logic diagram:** The logic diagram of the ROM-based controller using D flip-flops is shown in Figure 15.44. The outputs of the flip-flops (present states) are fed to the ROM and the inputs to flip-flops (next states) are fed from the ROM.

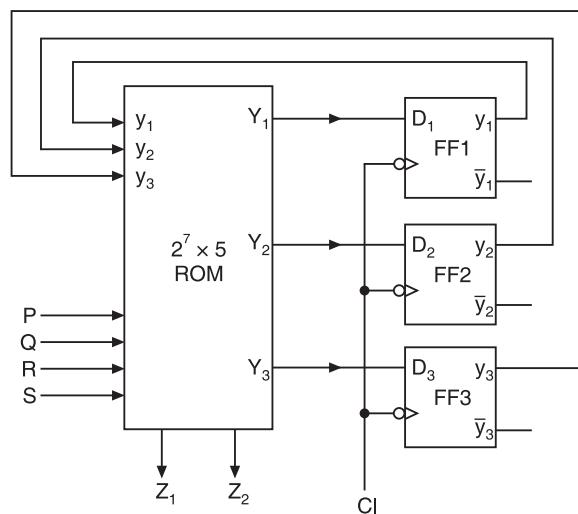


Figure 15.44 Example 15.13: ROM-based controller.

**ROM tabulation:** The ROM tabulation is shown in Table 15.14. The dashes in the table indicate that the corresponding inputs are don't cares. It is assumed that all illegal states go to state A(000) without activating any outputs.

Table 15.14 Example 15.13: ROM tabulation

PS			Inputs				NS			Outputs	
$y_1$	$y_2$	$y_3$	P	Q	R	S	$Y_1$	$Y_2$	$Y_3$	$Z_1$	$Z_2$
0	0	0	0	-	-	-	0	0	1	0	0
0	0	0	1	0	-	-	0	0	1	0	0
0	0	0	1	1	-	-	0	1	1	0	0
0	0	1	-	-	-	-	0	1	0	0	0
0	1	0	-	-	0	-	0	1	0	0	0
0	1	0	-	-	1	0	0	0	0	0	0
0	1	0	0	-	1	1	1	0	0	1	0
0	1	0	1	-	1	1	0	1	1	1	0
0	1	1	-	-	-	-	0	0	0	1	0
1	0	0	-	-	-	-	0	0	0	0	1
1	0	1	-	-	-	-	0	0	0	0	0
1	1	0	-	-	-	-	0	0	0	0	0
1	1	1	-	-	-	-	0	0	0	0	0

**EXAMPLE 15.14** For the ASM chart given in Figure 15.45a (a) draw the state diagram, (b) design the control unit using D flip-flops and a decoder, and (c) design the control unit using multiplexers and a register.

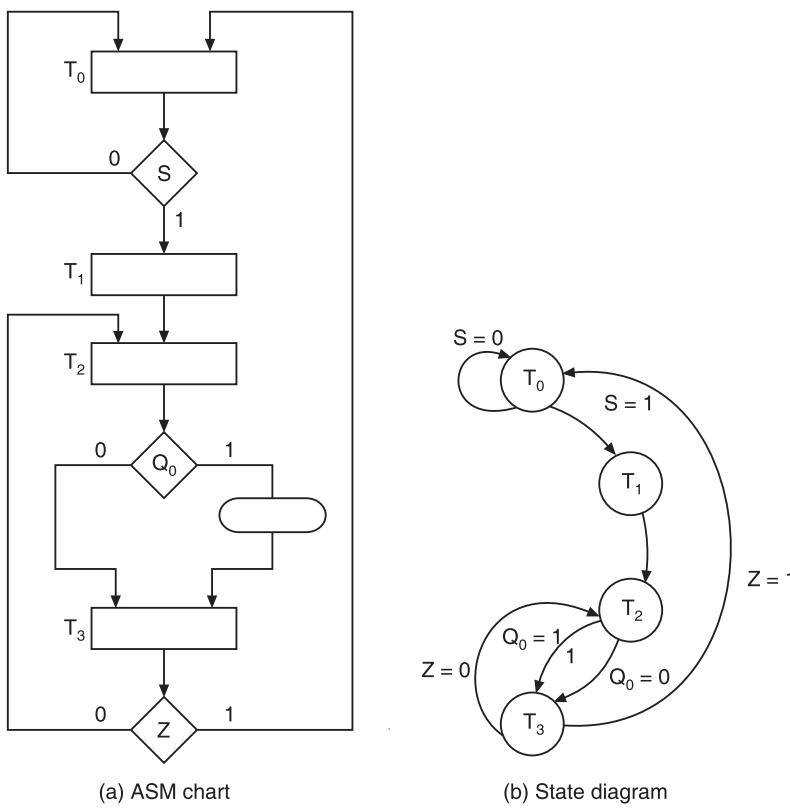


Figure 15.45 Example 15.14.

**Solution**

- (a) The state diagram for the given ASM chart is shown in Figure 15.45b. The ASM chart has four state boxes. So the state diagram will have four nodes.
- (b) To design the control unit using D flip-flops and a decoder we require two flip-flops because there are four states in the ASM chart. Let the state assignment be  $T_0 = 00$ ,  $T_1 = 01$ ,  $T_2 = 10$  and  $T_3 = 11$ . The state table for the control subsystem is as shown in Table 15.15.

Table 15.15 Example 15.14: State table

PS		Inputs			NS		Flip-flop inputs	
$y_1$	$y_0$	S	$Q_0$	Z	$Y_1$	$Y_0$	$D_1$	$D_0$
0	0	0	—	—	0	0	0	S
		1	—	—	0	1		
0	1	—	—	—	1	0	1	0
		—	—	—	1	1		
1	0	—	—	0	1	0	1	1
		—	—	1	0	0		
1		—			—		Z̄ 0	

The inputs to the flip-flops are obtained as follows:

Row 0: Row 0 shows that in the next state, the output of flip-flop 1 is always 0. Hence we can put 0 in the column  $D_1$  of row 0. In the next state, the output of flip-flop 2 follows the input S. Hence we can put S in the column  $D_0$  of row 0.

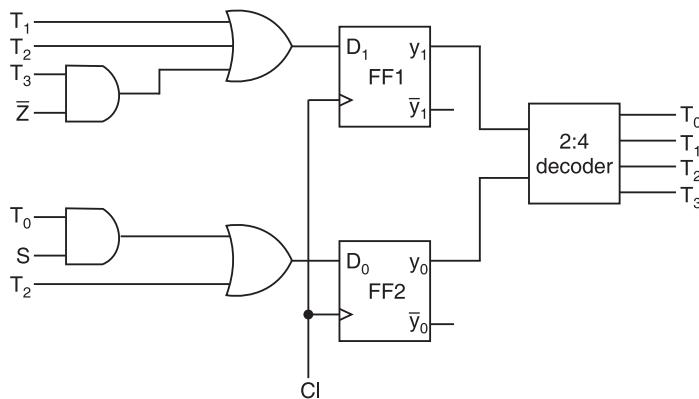
Row 1: Row 1 shows that in the next state, the output of flip-flop 1 is 1 and the output of flip-flop 2 is 0. Hence we can put 1 in the column  $D_1$  and 0 in the column for  $D_0$ .

Row 2: Row 2 shows that in the next state, the outputs of both the flip-flops are logic 1. Hence we can put 1 in the columns  $D_1$  and  $D_0$  of row 2.

Row 3: Row 3 shows that in the next state, the output of flip-flop 1 is the complement of Z. Hence we can put  $\bar{Z}$  in the column  $D_1$  of the row 3. In the next state, the output of flip-flop 2 is always 0. Hence we can put 0 in the column  $D_0$  of row 3.

$$\begin{aligned} D_1 &= T_0 \cdot 0 + T_1 \cdot 1 + T_2 \cdot 1 + T_3 \cdot \bar{Z} \\ &= T_1 + T_2 + T_3 \cdot \bar{Z} \\ D_0 &= T_0 \cdot S + T_1 \cdot 0 + T_2 \cdot 1 + T_3 \cdot 0 \\ &= T_0 S + T_2 \end{aligned}$$

The logic diagram using flip-flops and a decoder is shown in Figure 15.46.



**Figure 15.46** Example 15.14: Control circuit using flip-flops and decoder.

- (c) For designing the control circuit using multiplexers, two multiplexers are required because two state variables are there. The inputs to the multiplexers are the same as the inputs of the flip-flops corresponding to states 00, 01, 10, and 11 given in the state table. The outputs of flip-flops are connected to the select lines. The realization of the control unit using multiplexers and a register is as shown in Figure 15.25.

**EXAMPLE 15.15** (a) For the state diagram of a control circuit given in Figure 15.47, obtain the ASM chart.

(b) Design the circuit using multiplexers.

#### Solution

(a) The ASM chart for the control circuit with the given state diagram is shown in Figure 15.48.

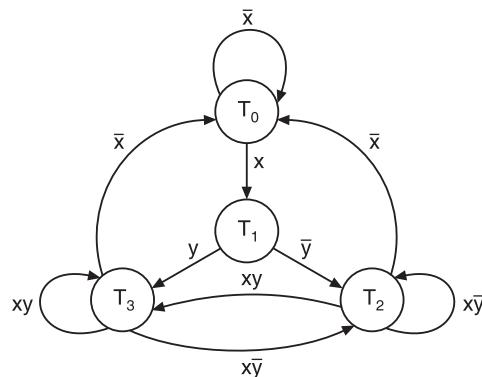


Figure 15.47 Example 15.15: State diagram.

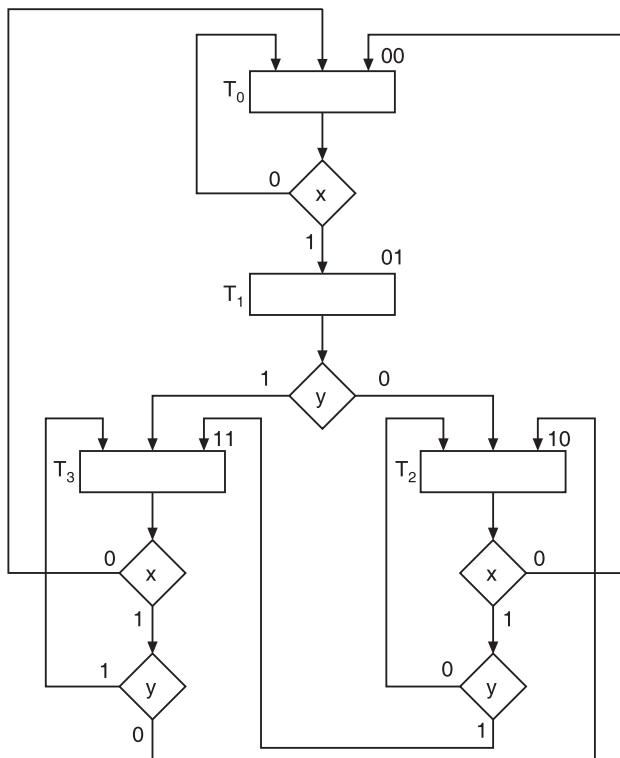


Figure 15.48 Example 15.15: ASM chart.

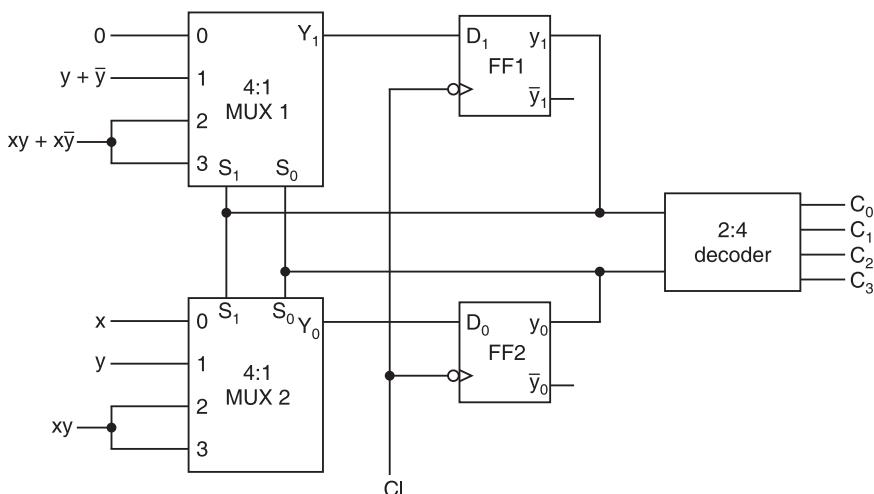
- (b) Let the state assignment be  $T_0 = 00$ ,  $T_1 = 01$ ,  $T_2 = 10$  and  $T_3 = 11$ . To design the circuit using multiplexers, the transition table is drawn as shown in Table 15.16. The inputs to the multiplexers are determined from the transition table (Table 15.17). Since the circuit has four states, two flip-flops and two 4:1 MUXes and one 2:4 decoder are required. Let the control signals be  $C_0 = 1$ ,  $C_1 = 1$ ,  $C_2 = 1$  and  $C_3 = 1$  when the circuit is in states  $T_0$ ,  $T_1$ ,  $T_2$  and  $T_3$  respectively. The circuit using multiplexers is shown in Figure 15.49.

**Table 15.16** Example 15.15: Transition table

No	PS		NS		Condition of transition	
	y <sub>1</sub>	y <sub>0</sub>	Y <sub>1</sub>	Y <sub>0</sub>		
1	T <sub>0</sub>	0	0	T <sub>0</sub>	0	$\bar{x}$
				T <sub>1</sub>	1	x
2	T <sub>1</sub>	0	1	T <sub>2</sub>	1	$\bar{y}$
				T <sub>3</sub>	1	y
3	T <sub>2</sub>	1	0	T <sub>0</sub>	0	$\bar{x}$
				T <sub>2</sub>	0	$x\bar{y}$
				T <sub>3</sub>	1	xy
4	T <sub>3</sub>	1	1	T <sub>0</sub>	0	$\bar{x}$
				T <sub>3</sub>	1	xy
				T <sub>2</sub>	0	$x\bar{y}$

**Table 15.17** Example 15.15: Inputs for multiplexers

MUX 1	MUX 2
0 → 0	0 → x
1 → y + $\bar{y}$	1 → y
2 → x $\bar{y}$ + xy	2 → xy
3 → xy + x $\bar{y}$	3 → xy

**Figure 15.49** Example 15.15: Logic circuit using multiplexers.

**EXAMPLE 15.16** (a) Draw the ASM chart for the following state transitions. Start from the initial state  $T_1$ , then, if  $xy = 00$  go to  $T_2$ , if  $xy = 01$  go to  $T_3$ , if  $xy = 10$  go to  $T_1$ , otherwise go to  $T_3$  and design its control circuit using

- (i) D flip-flop and decoder
- (ii) Input multiplexer and register
- (b) Show the exit paths in an ASM block for all binary combinations of control variables  $x$ ,  $y$  and  $z$  starting from an initial state.

**Solution**

- (a) The ASM chart for the given state transitions is shown in Figure 15.50a. The corresponding state diagram and state table are shown in Figures 15.50b and c respectively.

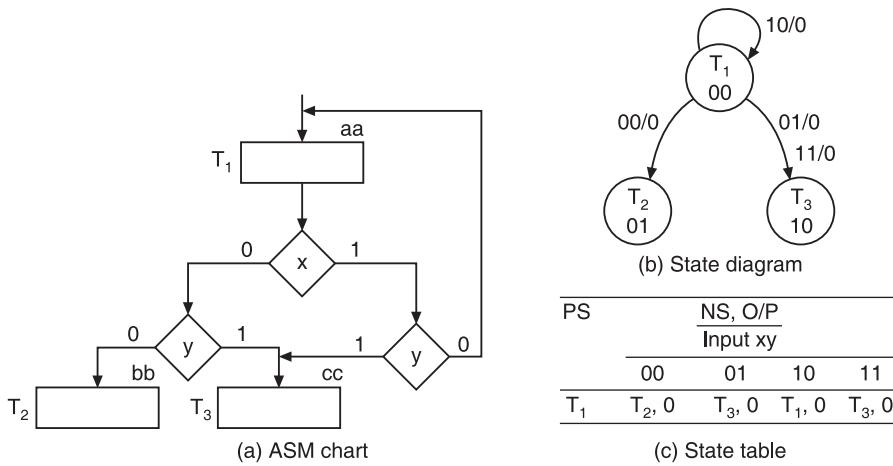
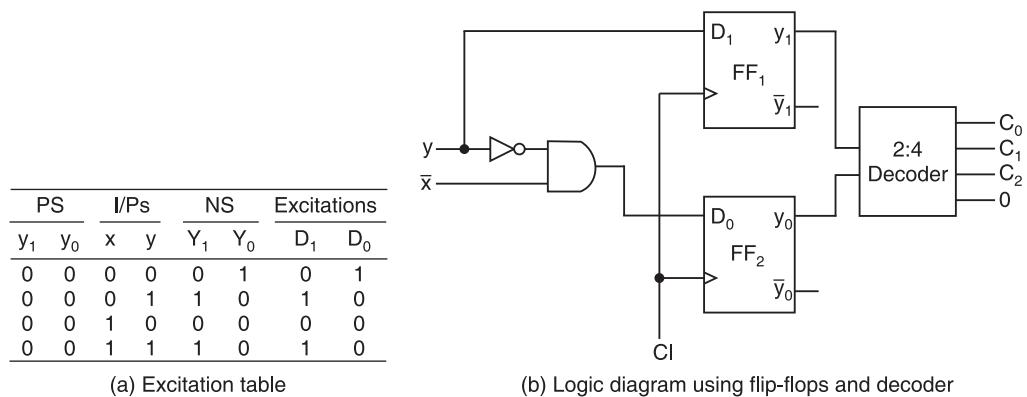


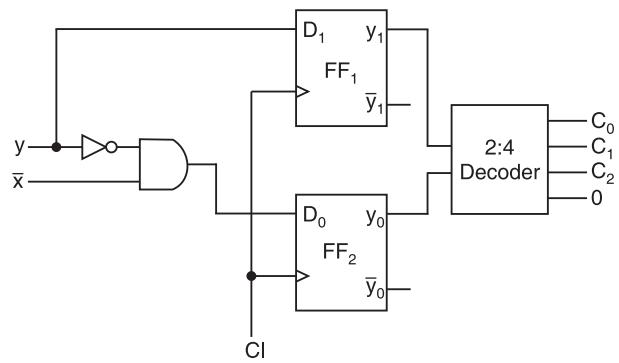
Figure 15.50 Example 15.16.

The sequential machine has three states. So two state variables are required. Two state variables can represent four states. Since only three are used, the remaining state is invalid. Also since transitions only from  $T_1$  are given and the transitions from the other states are not specified, they may also be treated as invalid. Since D flip-flops are used two D flip-flops are required. Let the control signals be  $C_1$  while in state  $T_1$ ,  $C_2$  while in state  $T_2$ ,  $C_3$  while in state  $T_3$ , and 0 while in state  $T_4$  (since  $T_4$  is invalid). So the outputs of the decoder will be  $C_1$ ,  $C_2$ ,  $C_3$ , 0. The inputs to the flip-flops are obtained from the excitation table shown in Figure 15.51a as  $D_1 = y$  and  $D_0 = \bar{x}\bar{y}$ . When the multiplexer control is used, two 4:1 MUXes are required. Since the transitions occur only from the state  $T_1$ , the input to the input terminal 1 of MUX1 is  $y$  and the input to the input terminal 1 of MUX2 is  $\bar{x}\bar{y}$ . All other input terminals are connected to logic 0. The designed control circuit using D flip-flop and decoder is shown in Figure 15.51b. The circuit using multiplexers and a register is shown in Figure 15.52.

- (b) The exit paths in an ASM block for all binary combinations of control variables  $x$ ,  $y$  and  $z$  starting from an initial state are shown in Figure 15.53. The ASM block contains one state box and eight decision boxes. No conditional output box is required.  $x$ ,  $y$  and  $z$  can be combined in eight possible ways. So eight decision boxes are required.



(a) Excitation table



(b) Logic diagram using flip-flops and decoder

Figure 15.51 Example 15.16a.

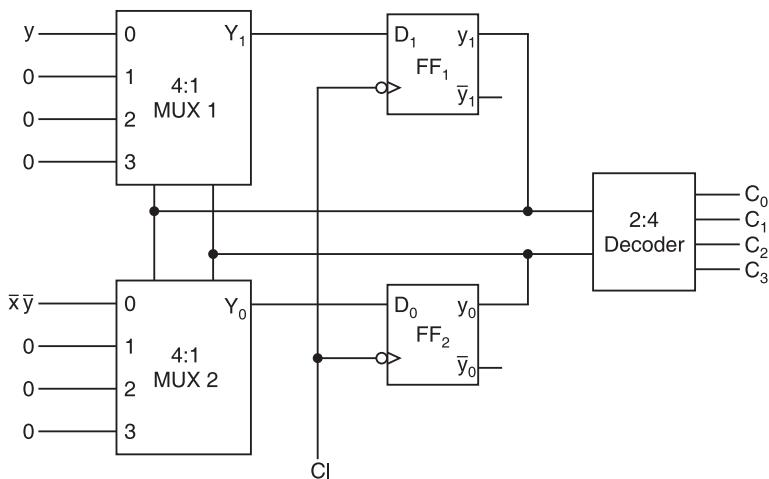


Figure 15.52 Example 15.16a: Circuit using multiplexers and register.

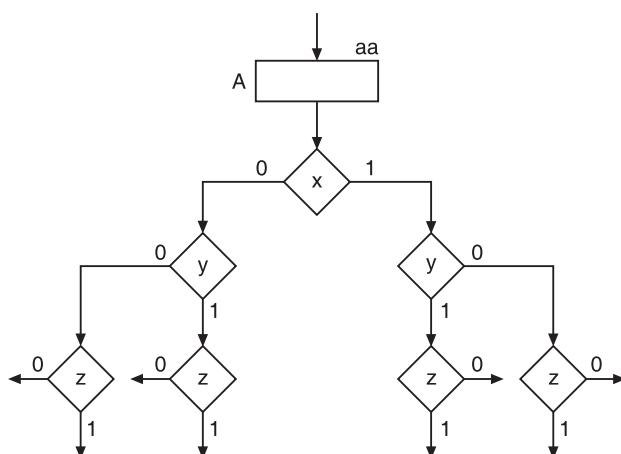


Figure 15.53 Example 15.16b: ASM chart.

**EXAMPLE 15.17** Obtain the ASM charts for the following state transitions:

- If  $x = 0$ , control goes from state  $T_1$  to state  $T_2$ . If  $x = 1$ , generate the conditional operation and go from  $T_1$  to  $T_2$ .
- If  $x = 1$ , control goes from  $T_1$  to  $T_2$  and then to  $T_3$ . If  $x = 0$ , control goes from  $T_1$  to  $T_3$ .

**Solution**

The ASM charts for the given state transitions are shown in Figure 15.54.

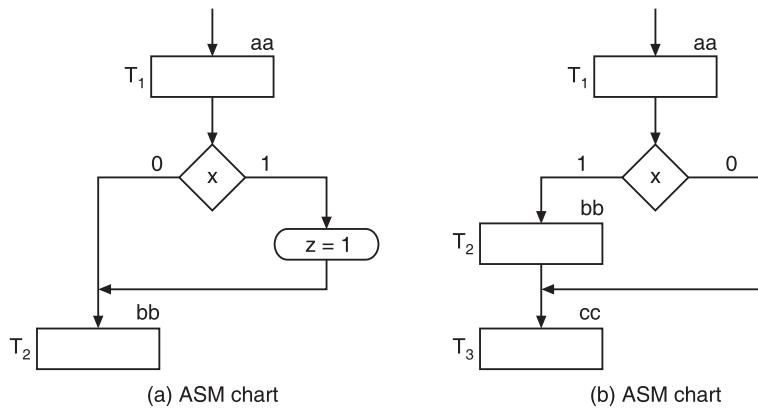


Figure 15.54 Example 15.17: ASM charts.

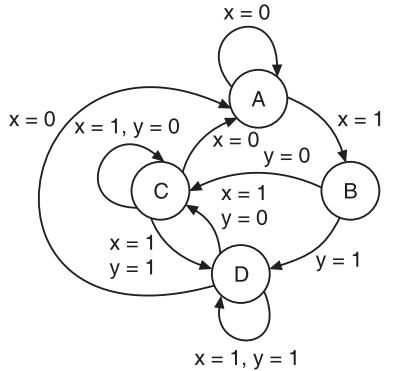
**EXAMPLE 15.18** (a) Draw the state diagram and the state table of the control unit for conditions given below.

- Draw the equivalent ASM chart leaving the state box empty.
- Design the control unit with the multiplexers for the above problem.
- Design the control unit using D flip-flops and a decoder.
  - From 00 state, if  $x = 1$ , it goes to 01 state and if  $x = 0$ , it remains in the same state 00.
  - From 01 state, if  $y = 1$ , it goes to 11 state and if  $y = 0$ , it goes to 10 state.
  - From 10 state, if  $x = 1$  and  $y = 0$ , it remains in the same state 10 and if  $x = 1$  and  $y = 1$ , it goes to 11 state, and if  $x = 0$ , it goes to 00 state.
  - From 11 state, if  $x = 1$ ,  $y = 0$ , it goes to 10 state and if  $x = 1$  and  $y = 1$ , it remains in the same state, and if  $x = 0$ , it goes to 00 state.

**Solution**

- The state diagram and the state table for the given conditions are shown in Figures 15.55a and b respectively.
- The equivalent ASM chart of the control unit leaving the state box empty is shown in Figure 15.56. Let the state assignment be A = 00, B = 01, C = 10, D = 11.
- The design of control unit using multiplexers is as follows. From the ASM chart write the transition table. Looking at the conditions of transition in the transition table, write the inputs of the multiplexers as shown in the inputs for multiplexers table. The control circuit using multiplexers is drawn as shown in Figure 15.57.

(d) To design the control unit using D flip-flops and decoder write the state table as shown in Table 15.18.



(a) State diagram

PS	NS			
	Input x y	00	01	10
A $\rightarrow$ 00	A	A	B	B
B $\rightarrow$ 01	C	D	C	D
C $\rightarrow$ 10	A	A	C	D
D $\rightarrow$ 11	A	A	C	D

(b) State table

Figure 15.55 Example 15.18.

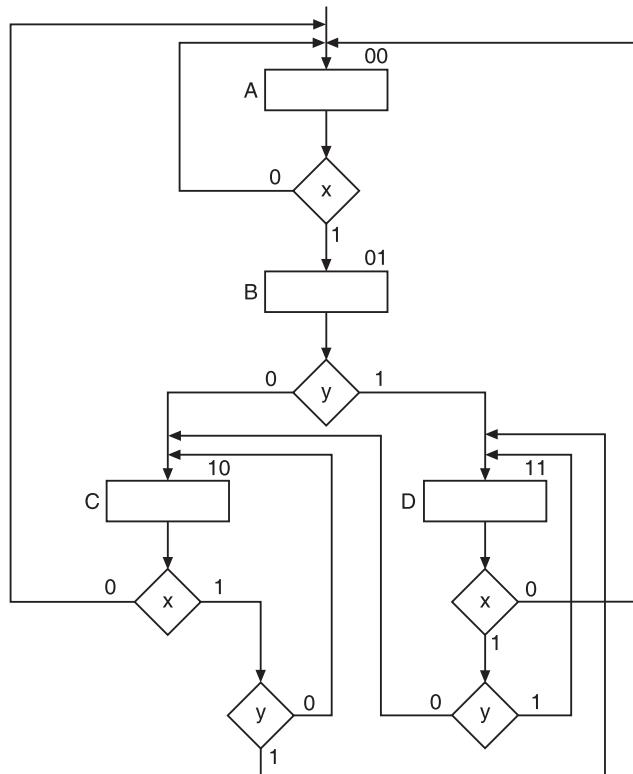


Figure 15.56 Example 15.18: ASM chart.

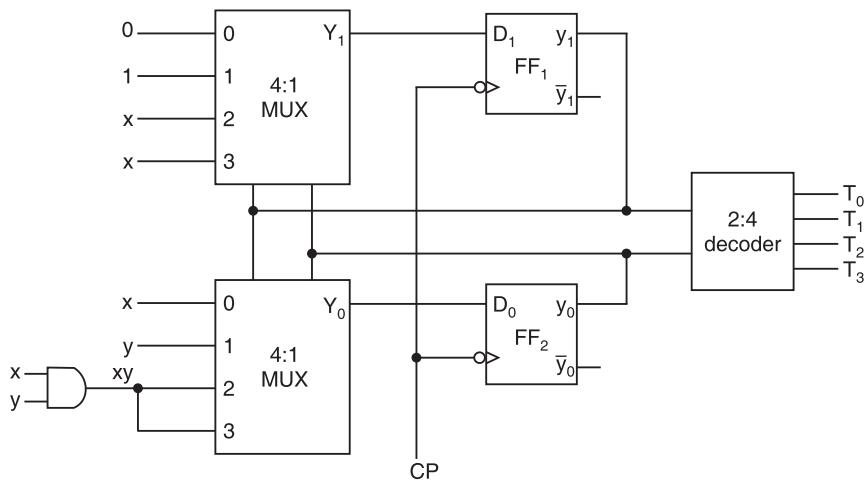


Figure 15.57 Example 15.18: Multiplexer control.

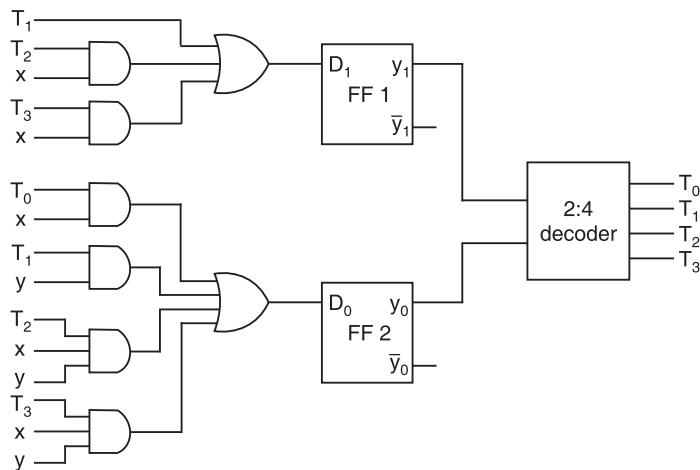
Table 15.18 Example 15.18: Transition table

PS		I/Ps		NS		FF inputs	
$y_1$	$y_0$	x	y	$y_1$	$y_0$	$D_1$	$D_0$
0	0	0	-	0	0	0	x
		1	-	0	1		
	1	-	0	1	0	1	y
		-	1	1	1		
1	0	1	0	1	0	x	xy
		0	-	0	0		
		1	1	1	1		
	1	0	-	1	0		
1	1	1	0	1	0	x	xy
		1	1	1	1		

Let the control signals produced be  $T_0 = 1$  while in state A,  $T_1 = 1$  while in state B,  $T_2 = 1$  while in state C, and  $T_3 = 1$  while in state D. Therefore, the inputs to the flip-flops are as follows:

$$\begin{aligned}
 D_1 &= T_0 \cdot 0 + T_1 \cdot 1 + T_2 \cdot x + T_3 \cdot x \\
 &= T_1 + T_2 x + T_3 x \\
 D_2 &= T_0 \cdot x + T_1 \cdot y + T_2 \cdot xy + T_3 \cdot xy
 \end{aligned}$$

The control subsystem implementation using D flip-flops and a decoder is shown in Figure 15.58.



**Figure 15.58** Example 15.18: Control subsystem using flip-flops and decoder.

### SHORT QUESTIONS AND ANSWERS

1. What is data?  
A. Data is discrete elements of information that are manipulated to perform arithmetic, logic, shift and other similar data processing tasks.
2. What is the purpose of control information?  
A. Control information provides command signals that supervise the various operations in the data section in order to accomplish the desired data processing tasks.
3. What are the two distinct parts of the logic design of a digital system?  
A. The logic design of a digital system can be divided into two distinct parts. One part is concerned with the design of the digital circuits that perform the data processing operations. The other part is concerned with the design of the control circuits that determine the sequence in which the various operations are performed.
4. What is an algorithm?  
A. An algorithm is a finite number of procedural steps that specify how to obtain a solution to a problem.
5. What is a hardware algorithm?  
A. A hardware algorithm is a procedure for implementing the problem with a given piece of equipment.
6. What is meant by the term ASM?  
A. ASM is an acronym for algorithmic state machine which is the same as a synchronous sequential circuit.
7. What is an ASM chart?  
A. An ASM chart is a flow-chart used to describe a clocked sequential circuit. It is a hardware algorithm which specifies the control sequence and datapath tasks of a digital system. A hardware

algorithm is a step-by-step procedure to implement the desired task with selected circuit components. It has three elements: state box, decision box, and conditional output box. ASM charts look similar to flow-charts but differ in that certain specific rules must be observed in constructing them. An ASM chart is equivalent to a state diagram or state graph.

**8. What are the elements of an ASM chart?**

A. The elements of an ASM chart are: state box, decision box, and conditional output box.

**9. Define a state box.**

A. A state box represents a state of a synchronous sequential circuit and is represented by a rectangular box with an arrow entering the box from top and an arrow leaving the box from bottom. The name of the state is written on the left side of the box and the binary code of the state is written on the top of the box. It is equivalent to a node in the state diagram or a row in the state table.

**10. What is a decision box?**

A. A decision box is represented by a diamond-shaped box with the variable or logic expression written inside it. The decision is taken on the basis of the variable or the logic expression being true or false. It has one arrow entering the box and two arrows leaving the box corresponding to true and false outcome.

**11. What is meant by conditional output box?**

A. Output of a synchronous sequential circuit or ASM is represented by conditional output box. It is an oval-shaped box with the output variable written inside. It has one arrow entering the box and one arrow leaving the box.

**12. What is an ASM block?**

A. An ASM block is a structure consisting of a state box and all the decision and conditional output boxes connected to its exit path. An ASM block has one entrance, any number of exit paths represented by the structure of the decision boxes. In an ASM chart there is one ASM block for each state. An ASM chart consists of one or more interconnected blocks. Each ASM block describes the state of the machine during one clock pulse.

**13. What is a link path?**

A. A path through an ASM block from entry to exit is referred to as a link path.

**14. Differentiate between an ASM chart and a conventional flow chart?**

A. The ASM chart is an alternative for describing the behaviour of finite state machines and is similar to a conventional flow chart. A conventional flow chart describes the sequence of procedural steps and decision paths for an algorithm without any concern for their time relationship. The ASM chart, on the other hand, describes the sequence of events as well as the timing relationship between the states of a sequence controller and the events that occur while going from one state to the next state after each clock edge. In an ASM chart every ASM block is considered as one unit, whereas in a flow-chart every operation block is considered as a separate unit. A conventional flow chart uses only two elements (a) state box and (b) decision box, whereas an ASM chart uses three elements (a) state box, (b) decision box and (c) conditional box.

**15. How many MUXs are required for the control subsystem?**

A. The number of MUXs required for the control subsystem is equal to the number of flip-flops used.

**16. What is the advantage of using MUXs for control?**

A. Multiplexer control results in a systematically organized structure which is easily interpreted and the sequence of states is easily found and visualized.

- 17.** What is the advantage of using PLA for control?  
**A.** The advantage of using PLA for control is that PLA control affords the facility of integration in a chip form.
- 18.** Distinguish between a data processing subsystem and a control subsystem.  
**A.** A data processing subsystem comprises both the combinational as well as the sequential circuits but a control subsystem essentially consists of sequential circuits which provide different operations and interstate transitions based on some predesigned conditions.
- 19.** Compare the ASM chart and the state diagram.  
**A.** The ASM chart is very similar to a state diagram. Each state box is equivalent to a state in a sequential circuit. The decision box is equivalent to binary information about the input applied written along the directed lines that connect two states in a state diagram. The conditional output box is equivalent to binary information about the output obtained written along with the input applied with a slash on the directed lines. (The directed lines indicate the conditions that determine the next state.) As a consequence, it is sometimes convenient to convert the chart into a state diagram and then use the sequential circuit procedure to design the control logic.

### **REVIEW QUESTIONS**

- 1.** What is an ASM chart?
- 2.** Name the elements of an ASM chart and define each one of them.
- 3.** Differentiate between an ASM chart and a conventional flow chart.
- 4.** What are the salient features of an ASM chart?
- 5.** What is an ASM block?
- 6.** Write a short note on multiplexer control.
- 7.** Write a short note on PLA control.
- 8.** Explain in detail the ASM technique of designing a sequential circuit.
- 9.** Draw and explain the ASM chart for a weighing machine.
- 10.** Draw and explain the ASM chart for a binary multiplier.
- 11.** Explain the datapath subsystem for a weighing machine.
- 12.** Explain the control subsystem implementation of a weighing machine.

### **FILL IN THE BLANKS**

- 1.** A row in the state table is the same as a \_\_\_\_\_ in an ASM chart.
- 2.** The Moore type of outputs are represented inside the \_\_\_\_\_ in an ASM chart.
- 3.** Unconditional outputs are \_\_\_\_\_ type.
- 4.** Algorithmic state machines can be implemented using flip-flops and \_\_\_\_\_ circuits.
- 5.** The binary information stored in a digital system can be described as \_\_\_\_\_ information or \_\_\_\_\_ information.

6. The data processing path is commonly referred to as the \_\_\_\_\_ path.
7. \_\_\_\_\_ type of outputs are referred to as unconditional outputs.
8. A state machine is another name of \_\_\_\_\_.
9. \_\_\_\_\_ type of outputs are referred to as conditional boxes.
10. A path through an ASM block from entry to exit is referred to as a \_\_\_\_\_.

### OBJECTIVE TYPE QUESTIONS

1. An ASM chart consists of
  - (a) only state boxes
  - (b) only decision boxes
  - (c) only decision and conditional output boxes
  - (d) state, decision, and conditional output boxes
2. In an ASM chart, Moore type of outputs are represented by
 

(a) writing these outputs inside state box	(b) conditional output box
(c) unconditional output box	(d) none of the above
3. An ASM chart can be
  - (a) converted into a state diagram
  - (b) converted into a state table
  - (c) implemented using gates and flip-flops
  - (d) converted as (a) and (b) and can be implemented as (c)
4. In an ASM chart, Mealy type of outputs
  - (a) cannot be represented
  - (b) can be represented by conditional output boxes
  - (c) can be represented by writing output state variables inside state box
  - (d) can be represented inside the decision boxes
5. An ASM chart can be implemented using flip-flops and
 

(a) gates	(b) multiplexers
(c) programmable logic devices	(d) any of the above
6. Mealy type of outputs are
 

(a) independent of the inputs	(b) dependent only on inputs
(c) dependent only on present state	(d) dependent on the present state and inputs
7. Moore type of outputs are
 

(a) independent of the inputs	(b) dependent only on the inputs
(c) dependent on present state and inputs	(d) dependent on type of hardware used for implementation
8. A synchronous sequential circuit can be described by
 

(a) a state diagram only	(b) a state table only
(c) an ASM chart only	(d) any one of the above

9. An ASM chart of the Mealy model

  - (a) contains conditional output box
  - (b) does not contain conditional output box
  - (c) is represented by writing output state variable inside the state box
  - (d) contains only state and decision boxes

10. A decision box in an ASM chart

  - (a) does not have exit paths
  - (b) has only one exit path
  - (c) has two exit paths
  - (d) has one entry and one exit path

11. A state box in an ASM chart

  - (a) is included only in one ASM block
  - (b) is not included in any ASM block
  - (c) may be included in any number of ASM blocks
  - (d) may be shared by two ASM blocks

12. ASM chart represents

  - (a) gates
  - (b) multiplexers
  - (c) synchronous sequential circuits
  - (d) PLAs

13. An algorithmic state machine consists of

  - (a) only gates
  - (b) only multiplexers
  - (c) combinational circuits and flip-flops
  - (d) only flip-flops

14. An algorithmic state machine is the same as

  - (a) synchronous sequential circuit
  - (b) clocked sequential circuit
  - (c) finite state machine
  - (d) all of the above

## PROBLEMS

- 15.1** Draw the state diagram, state table and the ASM chart for a 3-bit up-down counter. The circuit should generate a output 1 whenever the count becomes minimum or maximum.

**15.2** Draw the ASM chart for a sequence detector to detect the sequences 1011 and 1101. It has to output a 1 when the sequence is detected. Overlapping is not permitted.

**15.3** Design a mod-5 counter using multiplexers.

**15.4** Draw the ASM chart for (a) SR flip-flop and (b) T flip-flop.

**15.5** Draw an ASM chart, a state diagram and a state table for the synchronous circuit having the following description. The circuit has a control input C, clock and outputs x, y, z.

  - (i) If  $C = 1$ , on every clock rising edge, the code on the output x, y, z changes from  $000 \rightarrow 010 \rightarrow 100 \rightarrow 110 \rightarrow 000$  and repeats.
  - (ii) If  $C = 0$ , the circuit holds the present state.

**15.6** Develop an ASM chart and a state table for a controllable waveform generator that will output any one of the four wave forms given in Figure 15.6P as determined by the values of its inputs  $x_1$  and  $x_2$ . When an input change does occur, the new waveform may begin at any point in its period. Design the circuit using multiplexers.

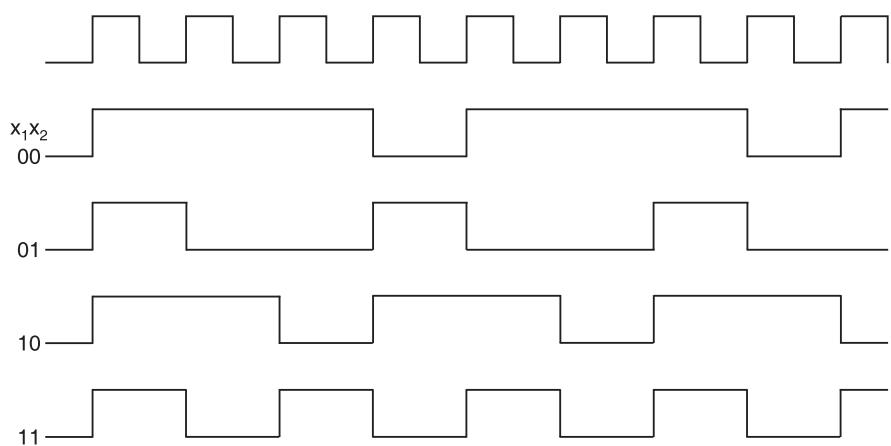


Figure 15.6P

## VHDL PROGRAMS

### 1. VHDL PROGRAM FOR BINARY MULTIPLIER

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity binary_multiplier is
    Port ( num1,num2 : in STD_LOGIC_VECTOR (2 downto 0);
           product : out STD_LOGIC_VECTOR (5 downto 0));
end binary_multiplier;

architecture Behavioral of binary_multiplier is
begin
process(num1,num2)
variable reg : std_logic_vector(5 downto 0);
variable add : std_logic_vector(3 downto 0);
begin
reg := "000" & num2;
for i in 1 to 3
loop
if(reg(0)='1') then
add:= ('0'& num1)+('0'& reg(5 downto 3));
reg:= add & reg(2 downto 1);
else
reg:= '0'&reg(5 downto 1);
end if;
end loop;
product <= reg;
end process;
end Behavioral;

```

### SIMULATION OUTPUT:

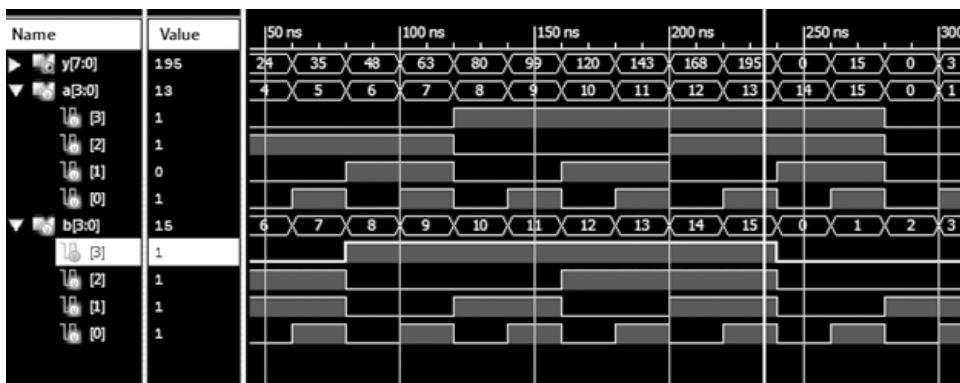
+/- /binary_multiplier/n...	7	7	6
+/- /binary_multiplier/n...	5	5	3
+/- /binary_multiplier/pr...	35	35	18

## VERILOG PROGRAMS

### 1. VERILOG PROGRAM FOR 4-BIT BINARY MULTIPLIER USING BEHAVIORAL MODELING

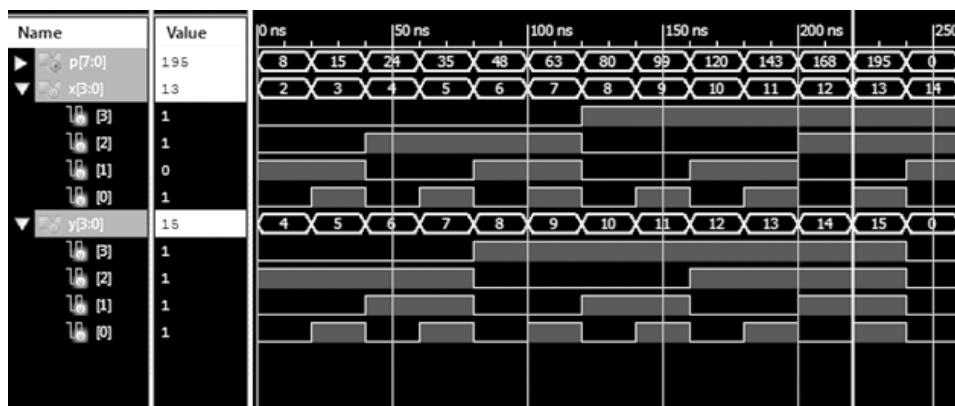
```
module mult4(y,a,b);
output [7:0]y;
input [3:0]a,b;
reg [7:0]y;
reg [7:0]acc;
integer i;
always @(a , b)
begin
acc=a;
y=0; // needs to zeroed
for(i=0;i<4;i=i+1)
begin
if(b[i])
y=y+acc; // must be a blocking assignment
acc=acc<<1;
end
end
endmodule
```

#### SIMULATION OUTPUT:



### 2. VERILOG PROGRAM FOR 4-BIT BINARY MULTIPLIER USING DATA FLOW MODELING

```
module mult4(p,x,y);
output [7:0]p;
input [3:0]x,y;
assign p = x*y;
endmodule
```

**SIMULATION OUTPUT:****3. VERILOG PROGRAM FOR DESIGN OF MAC UNIT IN STRUCTURAL MODELING**

```
module MAC_UNIT(clk,rst, a,b, z);
    input clk,rst;
    input [2:0] a,b;
    output [5:0] z;
    wire [5:0] w;
    multiplier U1(.a(a), .b(b), .p(w));
    pipe U2(.RIN(w), .clk(clk), .rst(rst), .ROUT(z));
endmodule
```

**VERILOG CODE FOR MULTIPLIER:**

```
module multiplier(a,b, p);
    input [2:0] a,b;
    output [5:0] p;
    wire [7:0]u;
    wire [1:0]su;
    wire [8:0]i;
    and (p[0],a[0],b[0]);
    and (u[0],a[1],b[0]);
    and (u[1],a[2],b[0]);
    and (u[2],a[0],b[1]);
    and (u[3],a[1],b[1]);
    and (u[4],a[2],b[1]);
    and (u[5],a[0],b[2]);
    and (u[6],a[1],b[2]);
    and (u[7],a[2],b[2]);
    hadd h1(.l(u[0]), .m(u[2]), .sum(p[1]), .cry(i[0]));
endmodule
```

```

hadd h2(.l(i[0]),.m(u[1]),.sum(su[0]),.cry(i[1]));
hadd h3(.l(u[3]),.m(u[5]),.sum(su[1]),.cry(i[2]));
hadd h4(.l(su[0]),.m(su[1]),.sum(p[2]),.cry(i[4]));
hadd h5(.l(i[1]),.m(i[2]),.sum(i[5]),.cry(i[6]));
or (i[7],i[5],i[4]);
fadd f3(.d(i[7]),.e(u[4]),.cin(u[6]),.s(p[3]),.cout(i[8]));
fadd f4(.d(i[8]),.e(i[6]),.cin(u[7]),.s(p[4]),.cout(p[5]));
endmodule

```

**VERILOG CODE FOR PARALLEL-IN, PARALLEL-OUT:**

```

module pipo(RIN, clk,rst, ROUT);
    input [5:0] RIN;
    input clk,rst;
    output [5:0] ROUT;
    reg [5:0] ROUT;
                                always @(posedge clk or negedge rst)
begin
    if(!rst)
        begin
            ROUT <= 6'b000000;
        end
    else
        begin
            ROUT <= RIN;
        end
end
endmodule

```

**VERILOG CODE FOR FULL-ADDER:**

```

module fadd(s,cout,d,e,cin);
    input d,e,cin;
    output s,cout;
    assign s = (d ^ e ^ cin);
    assign cout = ((d&e) | (e&cin) | (d&cin));
endmodule

```

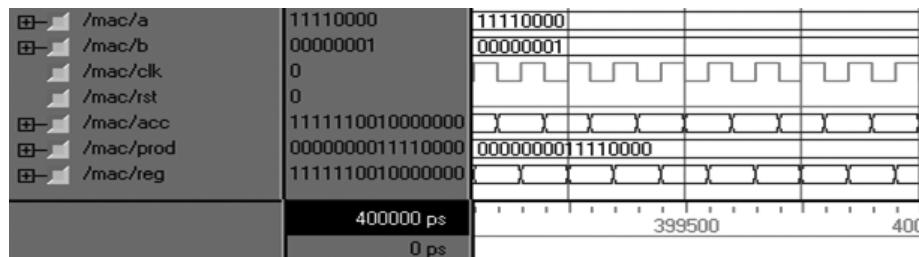
**VERILOG CODE FOR HALF- ADDER:**

```

module hadd(sum,cry,l,m);
    input l,m;
    output sum,cry;

```

```
wire sum,cry;
assign sum = (l^m);
assign cry = (l&m);
endmodule
```

**SIMULATION OUTPUT:**

# 16

## LOGIC FAMILIES

### 16.1 INTRODUCTION

Because of the advances in microelectronics, the digital IC technology in less than four decades has rapidly advanced from small scale integration (SSI), through medium scale integration (MSI), large scale integration (LSI), very large scale integration (VLSI), to ultra large scale integration (ULSI). The technology is now entering giant scale integration (GSI) in which millions of gate equivalent circuits are integrated on a single chip. The use of ICs has thus reduced the overall size of a digital system drastically. Consequently, the cost of digital systems has also reduced. The reliability has improved as well, because the number of external interconnections from one device to another has reduced. The power consumption of digital systems has also reduced greatly, because the miniature circuitry requires much less power.

ICs have certain limitations too. ICs cannot handle very large voltages or currents and also electrical devices like precision resistors, inductors, transformers, and large capacitors cannot be implemented on chips. So, ICs are principally used to perform low power circuit operations. The operations that require high power levels or devices that cannot be integrated are still handled by discrete components.

ICs are fabricated using various technologies such as TTL, ECL, and IIL which use bipolar transistors, whereas the MOS and CMOS technologies use unipolar MOSFETs.

## 16.2 DIGITAL IC SPECIFICATION TERMINOLOGY

The digital IC nomenclature and terminology is fairly standardized. The most useful specification terms are defined below.

### 16.2.1 Threshold Voltage

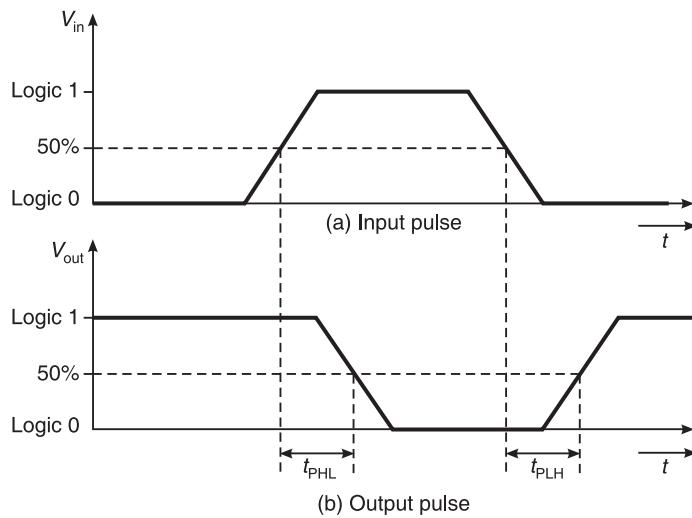
The threshold voltage is defined as that voltage at the input of a gate which causes a change in the state of the output from one logic level to the other.

### 16.2.2 Propagation Delay

A pulse through a gate takes a certain amount of time to propagate from input to output. This interval of time is known as the *propagation delay* of the gate. It is the average transition delay time  $t_{pd}$ , expressed by

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

where  $t_{PLH}$  is the signal delay time when the output goes from a logic 0 to a logic 1 state and  $t_{PHL}$  is the signal delay time when the output goes from a logic 1 to a logic 0 state (Figure 16.1).



**Figure 16.1** Propagation delay in an inverter.

### 16.2.3 Power Dissipation

Every logic gate draws some current from the supply for its operation. The current drawn in HIGH state is different from that drawn in LOW state. The *power dissipation*,  $P_D$ , of a logic gate is the power required by the gate to operate with 50% duty cycle at a specified frequency and is expressed in milliwatts. This means that 1 and 0 periods of the output are equal. The power dissipation of a gate is given by

$$P_D = V_{CC} \times I_{CC(\text{avg})}/n$$

where  $V_{CC}$  is the gate supply voltage,  $I_{CC}(\text{avg})$  is the average current drawn from the supply by the entire IC and  $n$  is the number of gates in the IC. Now

$$I_{CC}(\text{avg}) = \frac{I_{CCH} + I_{CCL}}{2}$$

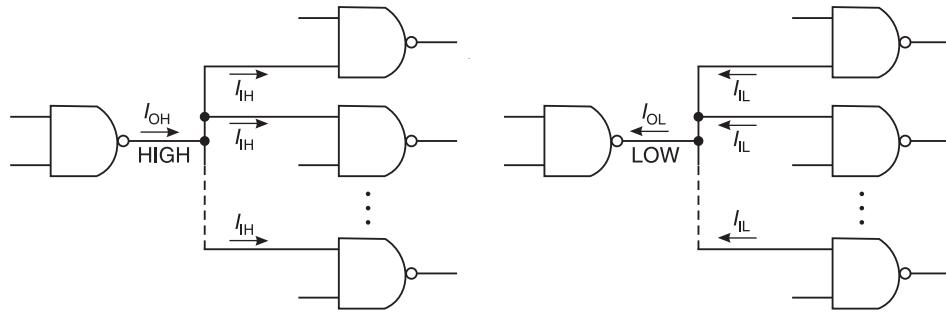
where  $I_{CCH}$  is the current drawn by the IC when all the gates in the IC are in HIGH state, and  $I_{CCL}$  is the current drawn by the IC when all the gates in the IC are in LOW state. The total power consumed by an IC is equal to the product of the power dissipated by each gate and the number of gates in that IC.

#### 16.2.4 Fan-in

The *fan-in* of a logic gate is defined as the number of inputs that the gate is designed to handle.

#### 16.2.5 Fan-out

The *fan-out* (also called the *loading factor*) of a logic gate is defined as the maximum number of standard loads that the output of the gate can drive without impairing its normal operation. A standard load is usually specified as the amount of current needed by an input of another gate of the same IC family. If a gate is made to drive more than this number of gate inputs, the performance of the gate is not guaranteed. The gate may malfunction.



**Figure 16.2** HIGH state, and LOW state fan-outs for TTL 7400 NAND gate.

Fan-out may be HIGH state fan-out, i.e. the fan-out of the gate when its output is logic 1, or it may be LOW state fan-output, i.e. the fan-out of the gate when its output is a logic 0. The smaller of these two numbers is taken as the actual fan-out. The fan-out of a gate affects the propagation delay time as well as saturation. The driving gate sinks current when it is in LOW state and sources current when it is in HIGH state.

$$\text{HIGH state fan-out} = \frac{I_{OH}(\text{max})}{I_{IH}}$$

where  $I_{OH}(\text{max})$  is the maximum current that the driver gate can source when it is in a 1 state and  $I_{IH}$  is the current drawn by each driven gate from the driver gate.

$$\text{LOW state fan-out} = \frac{I_{OL}(\text{max})}{I_{IL}}$$

where  $I_{OL}(\text{max})$  is the maximum current that the driver gate can sink when its output is a logic 0 and  $I_{IL}$  is the current drawn from each driven gate by the driver gate. Figure 16.2 depicts the current sourcing and current sinking actions for the TTL 7400 NAND gate.

### 16.2.6 Voltage and Current Parameters

The definitions of voltage and current levels corresponding to the logic 0 and logic 1 states are as follows:

**$V_{IH}(\text{min})$  (HIGH level input voltage):** It is the minimum voltage level required at the input of a gate for that input to be treated as a logic 1. Any voltage below this level will not be accepted as a logic 1 by the logic circuit.

**$V_{OH}(\text{min})$  (HIGH level output voltage):** It is the minimum voltage level required at the output of a gate for that output to be treated as logic 1. Any voltage below this level will not be accepted as logic 1 output.

**$V_{IL}(\text{max})$  (LOW level input voltage):** It is the maximum voltage level that can be treated as logic 0 at the input of the gate. Any voltage above this level will not be treated as a logic 0 input by the logic gate.

**$V_{OL}(\text{max})$  (LOW level output voltage):** It is the maximum voltage level that can be treated as logic 0 at the output of the gate. Any voltage above this level will not be treated as a logic 0 output.

**$I_{IH}(\text{HIGH level input current}):$**  The current that flows into an input when a specified HIGH level voltage is applied to that input.

**$I_{IL}(\text{LOW level input current}):$**  The current that flows into an input when a specified LOW level voltage is applied to that input.

**$I_{OH}(\text{HIGH level output current}):$**  The current that flows from an output in a logic 1 state under specified load conditions.

**$I_{OL}(\text{LOW level output current}):$**  The current that flows from an output in a logic 0 state under specified load conditions.

Figure 16.3 illustrates the currents and voltages in the HIGH and LOW states.

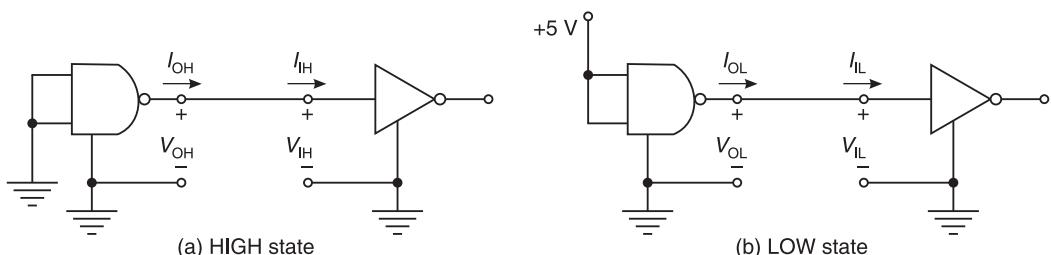


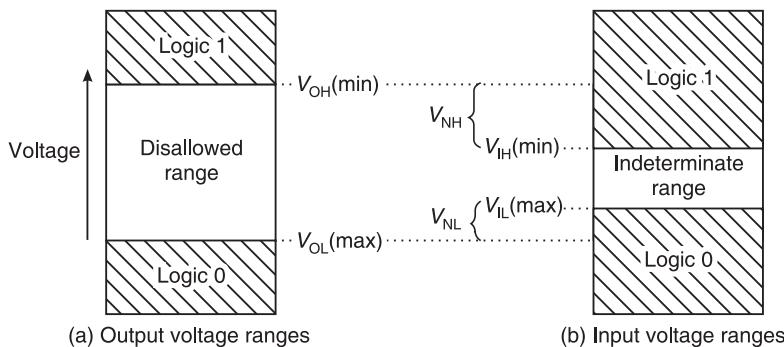
Figure 16.3 Currents and voltages in the HIGH and LOW states.

### 16.2.7 Noise Margin

When the digital circuits operate in noisy environment the gates may malfunction if the noise is beyond certain limits. The *noise immunity* of a logic circuit refers to the circuit's ability to tolerate noise voltages at its inputs. A quantitative measure of noise immunity is called *noise margin*.

Unwanted spurious signals are called *noise*. Noise may be ac noise or dc noise. A drift in the voltage levels of signals is called dc noise. The ac noise is a random pulse caused by other switching signals. Noise margin is expressed in volts and represents the maximum noise signal that can be added to the input signal of a digital circuit without causing an undesirable change in the circuit output. This is an important criterion for the selection of a logic family for certain applications where environment noise may be high.

Figure 16.4a shows the range of output voltages that can occur in a logic circuit. Voltages greater than  $V_{OH}(\text{min})$  are considered as a logic 1 and voltages lower than  $V_{OL}(\text{max})$  are considered as a logic 0. Voltages in the disallowed range should not appear at a logic circuit output under normal conditions. Figure 16.4b shows the input voltage requirements of a logic circuit. The logic circuit will respond to any input greater than  $V_{IH}(\text{min})$  as a logic 1 and to any input lower than  $V_{IL}(\text{max})$  as a logic 0. Voltages in the indeterminate range will produce an unpredictable response and should not be used.



**Figure 16.4** DC noise margins.

Noise margin may be HIGH state noise margin or LOW state noise margin:

**High state noise margin ( $NM_H$ )** is,  $V_{NH} = V_{OH}(\text{min}) - V_{IH}(\text{min})$

**Low state noise margin ( $NM_L$ )** is,  $V_{NL} = V_{IL}(\text{max}) - V_{OL}(\text{max})$

High state noise margin is the difference between the lowest possible high output and the minimum input voltage required for a HIGH. Low state noise margin is the difference between the largest possible low output and the maximum input voltage for a LOW.

### 16.2.8 Operating Temperatures

The IC gates and other circuits are temperature sensitive being semiconductor devices. However, they are designed to operate satisfactorily over a specified range of temperatures. The range specified for commercial applications is 0 to 70°C, for industrial it is 0 to 85°C, and for military applications it is – 55°C to 125°C.

### 16.2.9 Speed Power Product

A common means for measuring and comparing the overall performance of an IC family is the *speed power product*, which is obtained by multiplying the gate propagation delay by the gate power dissipation. A low value of speed power product is desirable. The smaller the product, the

better the overall performance. The speed power product has the units of energy and is expressed in picojoules. It is the *figure of merit* of an IC family. Suppose an IC family has an average propagation delay of 10 ns and an average power dissipation of 5 mW, the speed power product is

$$10 \text{ ns} \times 5 \text{ mW} = 50 \times 10^{-12} \text{ watt-seconds} = 50 \text{ picojoules (pJ)}$$

### 16.3 LOGIC FAMILIES

Many logic families have been developed. They are: Resistor Transistor Logic (RTL), Direct Coupled Transistor Logic (DCTL), Diode Transistor Logic (DTL), High Threshold Logic (HTL), Transistor Transistor Logic (TTL), Emitter Coupled Logic (ECL), Integrated Injection Logic (IIL), Metal-Oxide Semiconductor Logic (MOS), and Complementary Metal-Oxide Semiconductor Logic (CMOS). Out of these, RTL, DCTL, DTL, and HTL are obsolete. The logic families TTL, ECL, IIL, MOS, and CMOS are currently in use. The basic function of any type of gate is always the same regardless of the circuit technology used. The TTL and CMOS are suitable for SSI and MSI. The MOS and CMOS are particularly suitable for LSI. The IIL is mainly suitable for VLSI and ULSI. The ECL is mainly used in superfast computers. The logic families currently in use are compared in Table 16.1 in terms of the commonly used specification parameters.

**Table 16.1** Comparison of logic families

Logic family	Propagation delay time (ns)	Power dissipation per gate (mW)	Noise margin (V)	Fan-in	Fan-out	Cost
TTL	9	10	0.4	8	10	Low
ECL	1	50	0.25	5	10	High
MOS	50	0.1	1.5		10	Low
CMOS	< 50	0.01	5	10	50	Low
IIL	1	0.1	0.35	5	8	Very low

### 16.4 TRANSISTOR TRANSISTOR LOGIC (TTL)

The TTL or T<sup>2</sup>L family is so named because of its dependence on transistors alone to perform basic logic operations. It is the most popular logic family. It is also the most widely used bipolar digital IC family. The TTL uses transistors operating in saturated mode. It is the fastest of the saturated logic families. The basic TTL logic circuit is the NAND gate. Good speed, low manufacturing cost, wide range of circuits, and the availability in SSI and MSI are its merits. Tight  $V_{CC}$  tolerance, relatively high power consumption, moderate packing density, generation of noise spikes and susceptibility to power transients are its demerits.

The TTL logic family consists of several subfamilies or series such as:

**Standard TTL, High Speed TTL, Low power TTL, Schottky TTL, Low power Schottky TTL, Advanced Schottky TTL, Advanced low power Schottky TTL and F(fast)TTL.**

The differences between the various TTL subfamilies are in their electrical characteristics such as delay time, power dissipation, switching speed, fan-out, fan-in, noise margin, etc. For standard TTL, propagation delay time = 9 ns, power dissipation per gate = 10 mW, noise margin = 0.4 mV, fan-in = 8, and fan-out = 10.

For standard TTL, 0 V to 0.8 V is treated as a logic 0 and 2 V to 5 V is treated as logic 1. Signals in 0.8 V to 2 V range should not be applied as input as the corresponding response will be indeterminate. If a terminal is left open in TTL, it is equivalent to connecting it to HIGH, i.e. +5 V. But such a practice is not recommended, since the floating TTL is extremely susceptible to picking up noise signals that can adversely affect the operation of the device.

#### 16.4.1 Two-input TTL NAND Gate

In the circuit of the two-input TTL NAND gate shown in Figure 16.5 the input transistor  $Q_1$  is a multiple emitter transistor. Transistor  $Q_2$  is called the phase splitter. Transistor  $Q_3$  ‘sits above’  $Q_4$

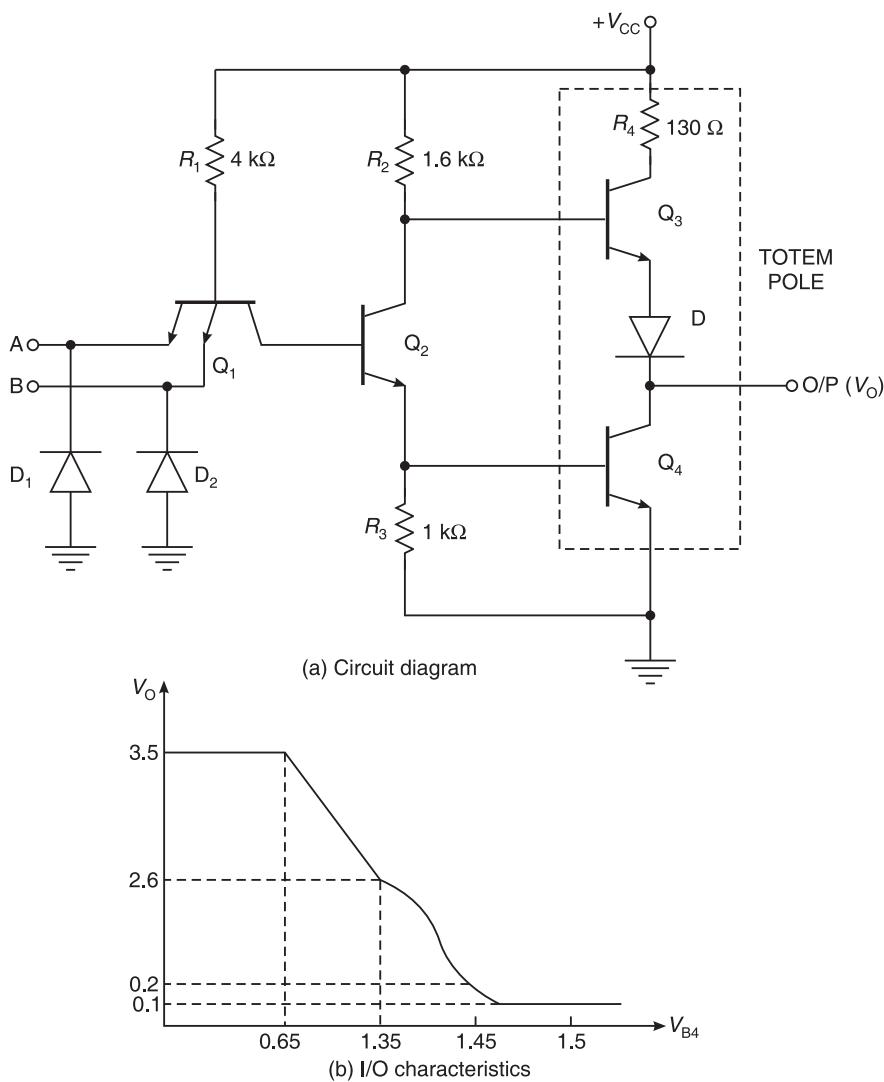


Figure 16.5 TTL NAND gate.

and, therefore,  $Q_3$  and  $Q_4$  make a *totem pole* arrangement. Diodes  $D_1$  and  $D_2$  protect  $Q_1$  from being damaged by the negative spikes of voltages at the inputs. When negative spikes appear at the input terminals, the diodes conduct and bypass the spikes to ground. Diode  $D$  ensures that  $Q_3$  and  $Q_4$  do not conduct simultaneously. Transistor  $Q_3$  acts as an emitter follower.

When both the inputs A and B are HIGH (+5 V), both the base-emitter junctions of  $Q_1$  are reverse biased. So, no current flows to the emitters of  $Q_1$ . However, the collector-base junction of  $Q_1$  is forward biased. So, a current flows through  $R_1$  to the base of  $Q_2$ , and  $Q_2$  turns on. Current from  $Q_2$ 's emitter flows into the base of  $Q_4$ . So,  $Q_4$  is turned on. The collector current of  $Q_2$  flows through  $R_2$  and, so, produces a drop across it thereby reducing the voltage at the collector of  $Q_2$ . Therefore,  $Q_3$  is OFF. Since  $Q_4$  is ON,  $V_O$  is at its low level ( $V_{CE}(\text{sat})$ ). So, the output is a logic 0. When either A or B or both are LOW, the corresponding base-emitter junction(s) is (are) forward biased and the collector-base junction of  $Q_1$  is reverse biased. So, the current flows to ground through the emitters of  $Q_1$ . Therefore, the base of  $Q_1$  is at 0.7 V, which cannot forward bias the base-emitter junction of  $Q_2$ . So,  $Q_2$  is OFF. With  $Q_2$  OFF,  $Q_4$  does not get the required base drive. So,  $Q_4$  is also OFF. Transistor  $Q_3$  gets enough base drive because  $Q_2$  is OFF, i.e. since no current flows into the collector of  $Q_2$ , all the current flows into the base of  $Q_3$ , and therefore,  $Q_3$  is ON. The output voltage,  $V_O = V_{CC} - V_{R2} - V_{BE3} - V_D \approx 3.4$  to 3.8 V, which is a logic HIGH level. So, the circuit acts as a two-input NAND gate. When  $Q_4$  is OFF, no current flows through it, but the stray and output capacitances between the output terminal, i.e. the collector of  $Q_4$ , and ground get charged to this voltage of 3.4 to 3.8 V. The I/O characteristics of a TTL NAND gate are shown in Figure 16.5b.

#### 16.4.2 Totem-pole Output

In the circuit diagram of the two-input TTL NAND gate, transistor  $Q_3$  sits above transistor  $Q_4$ . As shown in Figure 16.5,  $Q_3$  and  $Q_4$  are connected in 'totem-pole' fashion. At any time, only one of them will be conducting. Both cannot be ON or OFF simultaneously. Diode  $D$  ensures this. If  $Q_4$  is ON, its base is at 0.7 V w.r.t. ground.  $Q_4$  gets base drive from  $Q_2$ . So, when  $Q_4$  is ON,  $Q_2$  has to be ON. Therefore, its collector-to-emitter voltage is  $V_{CE}(\text{sat}) \approx 0.3$  V. Hence,  $V_{B3} = V_{C2} \approx 0.7$  V + 0.3 V  $\approx$  1 V. For  $Q_3$  to be ON, its base-emitter junction must be forward biased. When  $Q_4$  is ON,  $D$  has to be ON for  $Q_3$  to be ON simultaneously. So, the base voltage of  $Q_3$  must be  $V_{B3} = V_{CE4}(\text{sat}) + V_D + V_{BE3} \approx 0.7 + 0.3 + 0.7 \approx 1.7$  V, for it to be ON. Since  $V_{B3}$  is only 1 V when  $Q_4$  is ON,  $Q_3$  cannot be ON. Hence, it can be concluded that  $Q_3$  and  $Q_4$  do not be conducted simultaneously.

#### Advantages of totem-pole

- Even though the circuit can work with  $Q_3$  and  $D$  removed and  $R_4$  connected directly to the collector of  $Q_4$ , with  $Q_3$  in the circuit, there is no current through  $R_4$  in the output LOW state. So, the inclusion of  $Q_3$  and  $D$  keeps the circuit power dissipation low.
- In the output HIGH state,  $Q_3$  acts as an emitter follower with its associated low output impedance. This low output impedance provides a small time constant for charging up any capacitive load on the output. This action is commonly referred to as active pull-up and it provides very fast rise time waveforms at TTL output.

### ***Disadvantages of totem-pole***

1. During transition of the output from LOW to HIGH,  $Q_4$  turns off more slowly than  $Q_3$  turns on, and so, there is a period of a few nanoseconds during which both  $Q_3$  and  $Q_4$  are conducting and, therefore, relatively large currents will be drawn from the supply. So, TTL circuits suffer from internally generated current transients or current spikes because of the totem-pole connection.
2. Totem-pole outputs cannot be wire ANDed, that is, the outputs of a number of gates cannot be tied together to obtain AND operation of those outputs.

#### **16.4.3 Current Sinking**

A TTL circuit acts as a *current sink* in LOW state in that it receives current from the input of the gate it is driving.  $Q_4$  is the current-sinking transistor or the pull-down transistor, because it brings the output voltage down to its LOW state.

#### **16.4.4 Current Sourcing**

A TTL circuit acts as a *current source* in the HIGH state in that it supplies current to the gate it is driving.  $Q_3$  is the current-sourcing transistor or the pull-up transistor, because it pulls up the output voltage to its HIGH state.

#### **16.4.5 TTL Loading and Fan-out**

The TTL output has a limit,  $I_{OL}(\text{max})$ , on how much current it can sink in LOW state and a limit,  $I_{OH}(\text{max})$ , on how much current it can source in HIGH state. To determine the fan-out, we should know the drive capabilities of the output, i.e.  $I_{OL}(\text{max})$  and  $I_{OH}(\text{max})$  and the current requirements of each input, i.e.  $I_{IL}$  and  $I_{IH}$ :

$$\text{HIGH state fan-out} = \frac{I_{OH}(\text{max})}{I_{IH}}$$

$$\text{LOW state fan-out} = \frac{I_{OL}(\text{max})}{I_{IL}}$$

The smaller of these two numbers is the actual fan-out capability of the gate.

Suppose,  $Q_4$  can sink up to 16 mA before its output reaches  $V_{OL}(\text{max}) = 0.4$  V. Suppose  $I_{IL} = 1.6$  mA. This means that  $Q_4$  can sink the current from up to  $16/1.6 = 10$  loads. If it is connected to more than 10 loads,  $V_{OL}$  increases above 0.4 V, and so, the noise margin is reduced and  $V_{OL}$  may even go to the indeterminate state.

#### ***HIGH state fan-out***

When the TTL output is in a HIGH state,  $Q_3$  is acting as an emitter follower that is, sourcing a total current  $I_{OH}$ , which is the sum of the  $I_{IH}$  currents of the different TTL inputs. If too many loads are being driven, the current  $I_{OH}$  will become large enough to cause a large voltage drop across  $R_2$ , to bring  $V_{OH}$  ( $= V_{CC} - V_{R2} - V_{BE3} - V_D$ ) below  $V_{OH}(\text{min})$ . This is undesirable because it reduces the HIGH state noise margin and could even cause  $V_{OH}$  to go into the indeterminate range.

Suppose  $Q_3$  can source 0.4 mA of current before  $V_{OH}$  falls below  $V_{OH}(\min)$ , and each load receives  $I_{IH} = 40 \mu A$ . This means  $Q_3$  can source up to  $0.4 \text{ mA}/40 \mu A = 10$  loads.

### **Unit load**

Unit load means the current drawn or sourced back by similar gates. For 7400,

$$\begin{aligned}\text{One unit load} &= 40 \mu A \text{ in HIGH state} = I_{IH}(\max) \\ &= 1.6 \text{ mA in LOW state} = I_{IL}(\max)\end{aligned}$$

If the output of 7400 IC is rated at  $I_{OH}(\max) = 800 \mu A$  and  $I_{OL}(\max) = 48 \text{ mA}$ , then

$$\begin{aligned}\text{HIGH state fan-out} &= \frac{I_{OH}(\max)}{I_{IH}} = \frac{800 \mu A}{40 \mu A} = 20 \text{ unit loads} \\ \text{LOW state fan-out} &= \frac{I_{OL}(\max)}{I_{IL}} = \frac{48 \text{ mA}}{1.6 \text{ mA}} = 30 \text{ unit loads}\end{aligned}$$

Therefore, the actual fan-out is equal to the smaller of the above two, i.e. 20 unit loads.

**EXAMPLE 16.1** In Figure 16.5, the 7400 NAND gate has  $V_{CC} = +5 \text{ V}$  and a  $5\text{-k}\Omega$  load connected to its output. Find the output voltage (a) when both the inputs are  $+5 \text{ V}$  and (b) when both the inputs are  $0 \text{ V}$ .

### **Solution**

When both the inputs are HIGH, i.e.  $+5 \text{ V}$ ,  $Q_2$  and  $Q_4$  are in saturation. When  $Q_4$  is in saturation, its output is  $V_{CE}(\text{sat}) = \text{LOW} = 0.3 \text{ V}$ .

When both the inputs are LOW, i.e.  $0 \text{ V}$ , the output is HIGH ( $Q_3$  is ON but  $Q_2$  and  $Q_4$  are OFF). Therefore,

$$\begin{aligned}V_{OH} &= V_{CC} - I_L(R_4) - V_{CE}(\text{sat}) - V_D \\ &\approx [5 - (I_{OH} \times 130) - 0.1 - 0.7] \text{ V} \\ &\approx (4.2 - 130I_L) \text{ V}\end{aligned}$$

The load current,  $I_{OH} = \frac{V_{OH}}{5 \text{ k}\Omega}$ . Therefore,

$$V_{OH} = \left[ 4.2 - 130 \times \frac{V_{OH}}{5000} \right] \text{ V}$$

$$\text{or } V_{OH} \approx \frac{4.2}{1 + \frac{13}{500}} \text{ V} = 4.09 \text{ V}$$

**EXAMPLE 16.2** In Figure 16.5, what is the minimum value of the load resistance that can be used if the HIGH state output voltage is to be not less than  $3.5 \text{ V}$ ?

### **Solution**

$$\begin{aligned}V_{OH} &= 3.5 \text{ V} = V_{CC} - 130I_{OH} - V_{CE}(\text{sat}) - V_D \\ \text{or } 3.5 &= (5 - 130I_{OH} - 0.1 - 0.7) \text{ V} \\ &= (4.2 - 130I_{OH}) \text{ V}\end{aligned}$$

$$= \left[ 4.2 - 130 \times \frac{3.5}{R_L} \right] V$$

Therefore,

$$R_L = \frac{130 \times 3.5}{0.7} = 650 \Omega$$

**EXAMPLE 16.3** Determine the fan-out of the circuit of Figure 16.6. Also, find its noise margin.

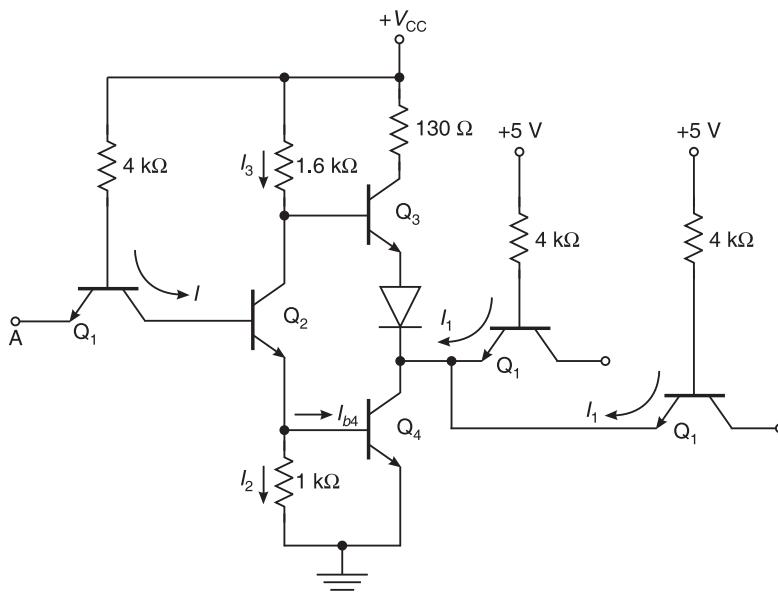


Figure 16.6 Example 16.3: Circuit.

### Solution

The given circuit is a standard TTL inverter. From the data sheets,  $I_{OL}(\text{max}) = 16 \text{ mA}$  and  $I_{IL}(\text{max}) = 1.6 \text{ mA}$ . Therefore,

$$\text{Fan-out} = \frac{I_{OL}(\text{max})}{I_{IL}(\text{max})} = \frac{16 \text{ mA}}{1.6 \text{ mA}} = 10$$

When the output is LOW,  $Q_4$  is in saturation. Therefore,

$$V_{OL} = V_{CE}(\text{sat}) \approx 0.3 \text{ V}$$

But  $V_{IL}(\text{max}) = 0.8 \text{ V}$ . Therefore,

$$\text{Low level noise margin} = V_{HL} = V_{IL}(\text{max}) - V_{OL} = 0.8 \text{ V} - 0.3 \text{ V} = 0.5 \text{ V}$$

*Another way of determining fan-out:* The fan-out is limited by the amount of current  $Q_4$  can sink, when it is in saturation. Let  $V_{OL}(\text{max}) = 0.4 \text{ V}$  (the limiting value). Let  $I_1$  be the current sunk from each load gate. Therefore,

$$I_1 = \frac{V_{CC} - V_{BE}(\text{sat}) - V_{OL}(\text{max})}{4 \text{ k}\Omega} \approx \frac{(5 - 0.75 - 0.4)\text{V}}{4 \text{ k}\Omega} \approx 1 \text{ mA}$$

The ability of the gate to sink current while keeping  $Q_4$  in saturation is severely limited at its lowest operating temperature,  $-55^\circ\text{C}$ . This is about 30 mA. So, fan-out can be taken as  $30 \text{ mA}/1 \text{ mA} = 30$ , but to keep  $V_{\text{OL}}$  well below the 0.4 V limit,  $I_{\text{OL}}(\text{max})$  is limited to 16 mA. So, the fan-out should be less than  $16 \text{ mA}/1 \text{ mA} = 16$ . For safety, the fan-out is taken as 10.

*The approximate fan-out may also be calculated as follows:*

$$\begin{aligned}\text{Current drawn from each driven gate, } I_1 &= \frac{V_{\text{CC}} - V_{\text{BE}} - V_{\text{OL}}(\text{max})}{4 \text{ k}\Omega} \\ &= \frac{(5 - 0.75 - 0.4)\text{V}}{4 \text{ k}\Omega} \approx 1 \text{ mA}\end{aligned}$$

Calculate the collector current of  $Q_4$ . In Figure 16.5,

$$\begin{aligned}I &= \frac{V_{\text{CC}} - V_{\text{BE}4} - V_{\text{BE}2} - V_{\text{BC}1}}{4 \text{ k}\Omega} \\ &\approx \frac{(5 - 0.75 - 0.75 - 0.7)\text{V}}{4 \text{ k}\Omega} \\ &\approx \frac{2.8 \text{ V}}{4 \text{ k}\Omega} \approx 0.7 \text{ mA} \\ I_2 &= \frac{V_{\text{BE}4}}{1 \text{ k}\Omega} \approx \frac{0.7 \text{ V}}{1 \text{ k}\Omega} \approx 0.7 \text{ mA} \\ I_3 &= \frac{V_{\text{CC}} - V_{\text{B}3}}{1.6 \text{ k}\Omega} = \frac{V_{\text{CC}} - V_{\text{BE}4} - V_{\text{CE}2}}{1.6 \text{ k}\Omega} \\ &\approx \frac{(5 - 0.7 - 0.3)\text{V}}{1.6 \text{ k}\Omega} \approx \frac{4 \text{ V}}{1.6 \text{ k}\Omega} \approx 2.5 \text{ mA}\end{aligned}$$

Therefore,

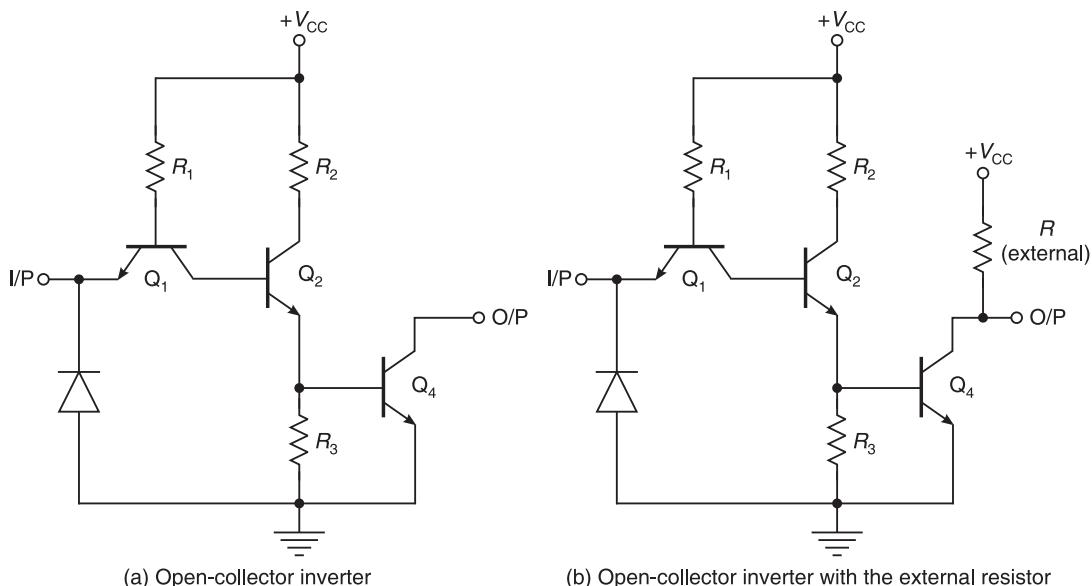
$$I_{\text{b}4} = I + I_2 - I_3 = (0.75 + 2.5 - 0.7) \text{ mA} = 2.55 \text{ mA}$$

The transistor  $Q_4$  is in saturation. Therefore,  $I_{\text{C}4}$  is also saturated. Let  $I_{\text{C}4}$  be about 4 to 5 times  $I_{\text{b}4}$  (worst case), i.e. about 10 to 15 mA. Hence, the fan-out is equal to 10 (the worst case).

## 16.5 OPEN-COLLECTOR GATES

The TTL gates may have totem-pole output or open-collector output. In the open-collector TTL, the output is at the collector of  $Q_4$  with nothing connected to it (i.e. pull-up transistor  $Q_3$  and diode D of the totem-pole output are omitted), therefore, the name *open collector*. The open-collector inverter circuit is shown in Figure 16.7a. In order to get the proper HIGH and LOW logic levels out of the circuit, an external pull-up resistor is connected to  $V_{\text{CC}}$  from the collector of  $Q_4$  as shown in Figure 16.7b. When  $Q_4$  is OFF, the output is pulled to  $V_{\text{CC}}$  through the external resistor  $R$ . When  $Q_4$  is ON, the output is connected to near ground through the saturated transistor. The value of  $R$  must be so chosen that when one gate output goes LOW while the others are HIGH, the sink current through the LOW output does not exceed the  $I_{\text{OL}}(\text{max})$  limit. Since the output is pulled to

logic HIGH level through a resistor, it is called the *passive pull-up*. The open-collector arrangement is much slower than the totem-pole arrangement, because the time constant with which the load capacitance charges in this case is considerably larger. (In the case of totem-pole, it is active pull-up, i.e. pull-up is through transistor  $Q_3$ . The  $R_{ON}$  of  $Q_3$  is very small; so, the charging time constant is low and the output rises fast.) The speed can be increased only a little bit by choosing a smaller resistance. For this reason, the open-collector circuits should not be used in applications where switching speed is a principal consideration. The traditional symbols for logic circuits with open-collector outputs are the same as those for totem-pole outputs.



**Figure 16.7** Circuit diagram and logic symbol of open-collector inverter.

### 16.5.1 Wired AND Operation

Sometimes the outputs of a number of NAND gates may have to be ANDed. This can be achieved by using two more NAND gates (4 and 5) as shown in Figure 16.8a. The same logic operation can be performed by simply tying the outputs of NAND gates 1, 2, and 3 as shown in Figure 16.8b. This is called *wired AND operation*, because the AND operation is obtained by simply connecting the output wires together. With this, when any of the gate outputs go to a LOW state, the common output point also goes LOW as a result of its shorting to ground through the ON transistor. The common output will be HIGH only when all the gate outputs are in a HIGH state. This arrangement has the advantage of needing fewer gates compared to the conventional arrangement.

The totem-pole outputs of gates cannot be wired ANDed, because when one output is HIGH and the other LOW and they are wired ANDed, a large current flows from supply to ground through  $Q_3$  of the HIGH-state gate and  $Q_4$  of the LOW-state gate. This is because  $Q_4$  of the LOW-state gate acts as a very low resistance load on  $Q_3$  of the HIGH-state gate. This large current can easily damage any of these transistors. The situation is even worse when more than two TTL outputs are tied together. The open-collector gates, on the other hand, may be wired ANDed without any problem.

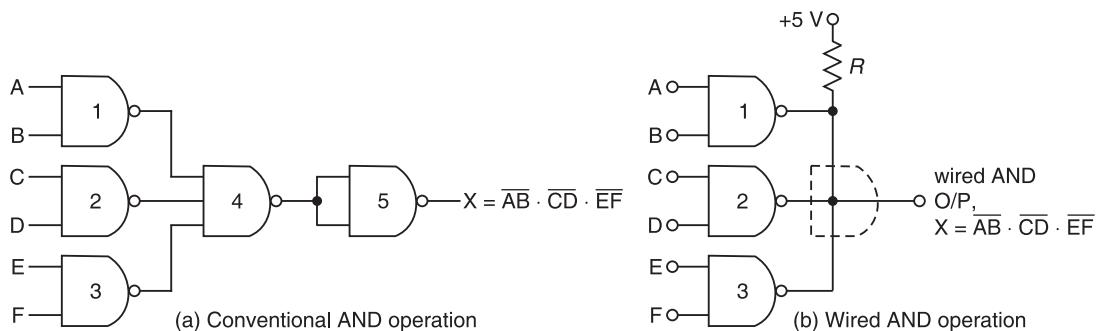


Figure 16.8 ANDing of TTL gates.

Because of the absence of pull-up transistors, the wired-AND connections significantly reduce switching speeds. However, they are useful in reducing the chip count of a system when speed is not a consideration.

### 16.5.2 Tri-state (3-state) TTL

The third TTL configuration is the tri-state configuration. It utilizes the advantage of high speed of operation of the totem-pole configuration and wire ANDing of the open-collector configuration. It is called the *tri-state* TTL, because it allows three possible output states: HIGH, LOW, and HIGH impedance (Hi-Z). In the Hi-Z state, both the transistors in the totem-pole arrangement are turned off, so that the output terminal is a HIGH impedance to ground or  $V_{CC}$ . In fact, the output is an open or floating terminal, that is, neither a LOW nor a HIGH. In practice, the output terminal is not an exact open circuit, but has a resistance of several  $M\Omega$  or more relative to ground and  $V_{CC}$ .

The circuit of a tri-state inverter is shown in Figure 16.9a. The tri-state operation is obtained by modifying the basic totem-pole circuit of Figure 16.5. The circuit has two inputs—A is the normal logic input and E is an enable input that can produce the Hi-Z. The traditional symbol is shown in Figure 16.9b.

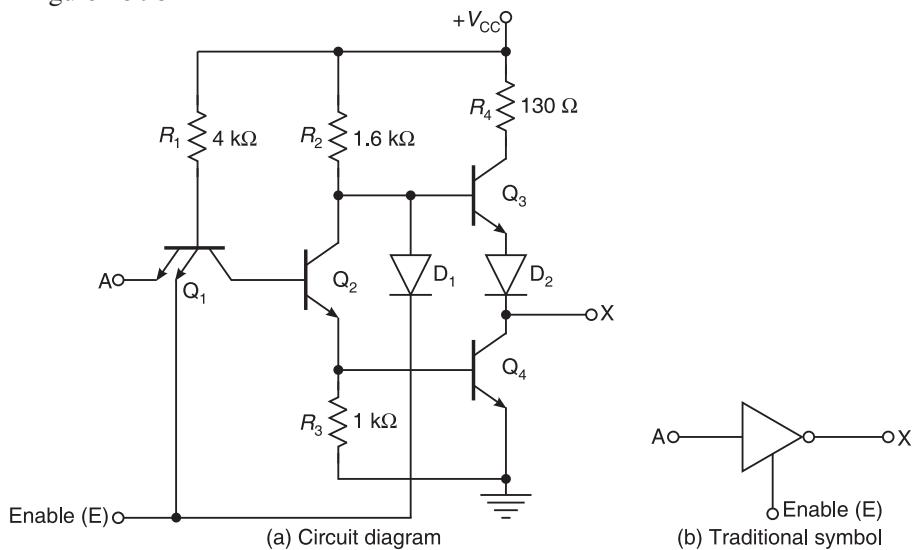


Figure 16.9 Tri-state TTL inverter.

### **The enabled state**

With  $E = 1$ , the circuit operates as a normal inverter because the high voltage at E has no effect on  $Q_1$  or  $Q_2$ . In this enabled condition, the output is simply the inversion of logic input A.

### **The disabled state (Hi-Z)**

When  $E = 0$ , the circuit goes into its Hi-Z state regardless of the state of logic input A. The LOW at E forward biases the emitter base junction of  $Q_1$  and shunts the current in  $R_1$  away from  $Q_2$ , so that,  $Q_2$  turns off, which in turn turns  $Q_4$  off. The LOW at E also forward biases diode  $D_1$  to shunt current away from the base of  $Q_3$ , and therefore,  $Q_3$  also turns off. With both totem-pole transistors in the non-conducting state, the output terminal is essentially an open circuit.

There are many ICs that are designed with tri-state outputs. The advantage of the tri-state configuration is that the outputs of the tri-state ICs can be connected together without sacrificing the switching speed. The traditional logic symbology has no special notation for tri-state outputs.

### **16.5.3 Buffer/Drivers**

Any logic circuit that is called a *buffer*, a *driver*, or a *buffer/driver* is designed to have a greater output current and/or voltage capability than that of an ordinary logic circuit. Buffer/driver ICs are available with totem-pole outputs, open-collector outputs, or tri-state outputs. Some tri-state buffers also invert the signal as it goes through. They are called *inverting tri-state buffers*.

## **16.6 TTL SUBFAMILIES**

### **16.6.1 Standard TTL, 74 Series**

The standard TTL ICs, i.e. 74 series, offer a combination of speed and power dissipation suited for many applications. The 54 series is the counterpart of the 74 series. They are functionally equivalent to the 74 series, but can be operated over wider temperature and voltage ranges, as required by military specifications. Several other TTL series discussed below have also been developed. The standard TTL is now rarely used in new systems.

### **16.6.2 Low Power TTL, 74L Series**

The low power TTL circuits, designated as the 74L series, have essentially the same basic circuit as the standard 74 series, except that all resistance values are increased ( $R_1 = 40 \text{ k}\Omega$ ,  $R_2 = 20 \text{ k}\Omega$ ,  $R_3 = 12 \text{ k}\Omega$ , and  $R_4 = 500 \Omega$ ). The larger resistors reduce the current and, therefore, the power requirement, but at the expense of reduction in speed. The power consumption of low power TTL is about 1/10 of that of standard TTL, but the standard TTL is more than three times faster than the low power TTL. The low power version is now not available in 7400 series. Low power Schottky TTL and CMOS versions of the 7400 series are now widely used instead.

### **16.6.3 High Speed TTL, 74H Series**

The high speed TTL circuits, designated as the 74H series, have essentially the same basic circuit as the standard 74 series, except that smaller resistance values ( $R_1 = 2.8 \text{ k}\Omega$ ,  $R_2 = 760 \Omega$ ,  $R_3 = 470 \Omega$ , and  $R_4 = 58 \Omega$ ) are used and the emitter follower transistor  $Q_3$  is replaced by a Darlington pair and emitter to base joining of Darlington pair ( $Q_5 - Q_3$ ) is connected to ground through a resistance of

$4\text{ k}\Omega$ . The switching speed of the 74H series is approximately two times more than that of the standard TTL, as also the power consumption. Newer Schottky versions are superior in both speed and power consumption.

#### 16.6.4 Schottky TTL, 74S Series

The standard TTL, low power TTL, and high speed TTL series operate using saturated switching. When a transistor is saturated, excess charge carriers will be stored in the base region and they must be removed before the transistor can be turned off. So, owing to storage time delay, the speed is reduced. The Schottky TTL 74S series reduces this storage time delay by not allowing the transistor to go into full saturation. This is accomplished by using a Schottky barrier diode (SBD) between the base and the collector of each transistor. Virtually, all modern TTL devices incorporate this so-called Schottky clamp. The SBD has a forward voltage of only 0.25 V. The circuits in the 74S series also use smaller resistance values to improve the speed of operation. The speed of the 74S series is twice that of the 74H series. Schottky TTL has more than three times the switching speed of standard TTL, at the expense of approximately doubling the power consumption. Figure 16.10 shows the circuit diagram of a two-input Schottky TTL NAND gate.

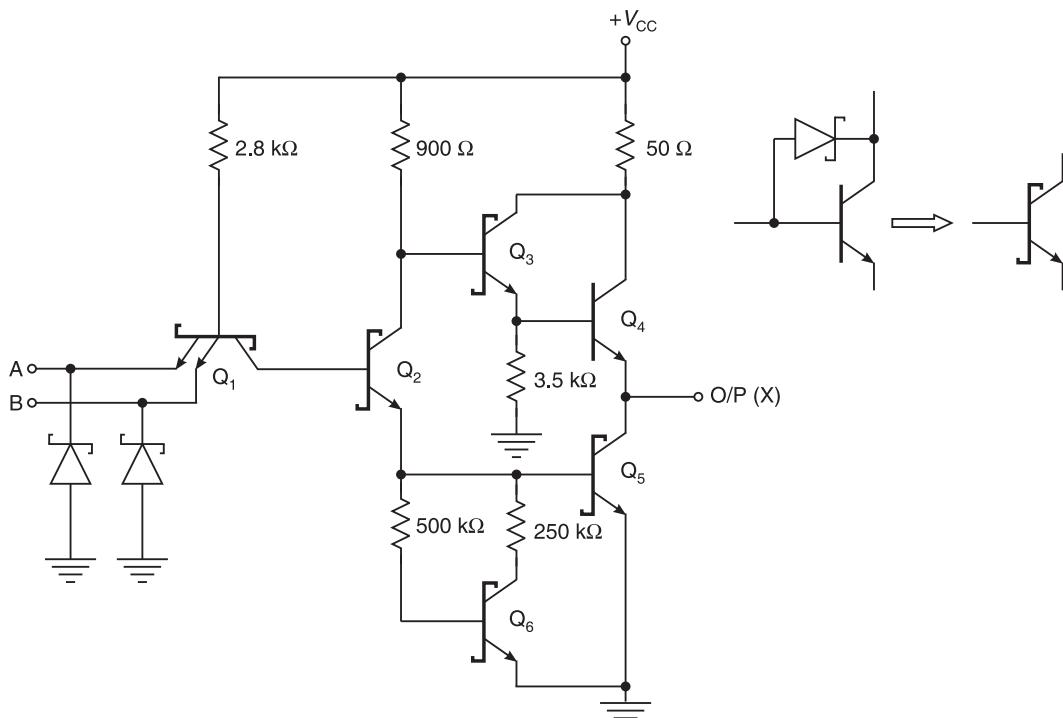


Figure 16.10 Circuit diagram of a two-input Schottky TTL NAND gate.

#### 16.6.5 Low Power Schottky TTL, 74LS Series

The 74LS series is a low-powered, slower-speed version of the 74S series. It uses the Schottky clamped transistor, but with larger resistance values than those in the 74S series. The larger resistance

values reduce the circuit power requirement but at the expense of reduction in speed. The switching speed of low power Schottky TTL is about the same as that of the standard TTL, but the power consumption is about 1/5 of the standard TTL. The 74LS NAND gate does not use the multiple emitter input transistor. Instead it uses diodes. This series is replacing the 74 series.

#### 16.6.6 Advanced Schottky TTL, 74AS Series

The 74AS series has developed owing to recent innovations in IC design. It provides considerable improvement in speed over the 74S series and at a much lower power requirement. It is the fastest TTL series and its speed-power product is significantly lower than that of the 74S series. Its fan-out is larger than that of the 74S series because of its lower input current requirement. It is twice as fast and consumes less than half as much power as the 74S series.

#### 16.6.7 Advanced Low Power Schottky TTL, 74ALS Series

The 74ALS series is a low power version of the advanced Schottky TTL. This series offers an improvement over the 74LS series in both speed and power dissipation. The 74ALS has the lowest speed-power product of all the TTL series, and it is very close to having the lowest gate power dissipation.

#### 16.6.8 F(fast)TTL, 74F Series

The 74F series is the newest and fastest TTL series. Devices in this series have the letter 'F' inserted in their designations.

The ALS and AS technologies are the recent enhancements in Schottky TTL circuitry. Among other refinements, the advanced Schottky devices are fabricated with an improved doping technique and Schottky-clamped transistors provide improved isolation. These enhancements reduce capacitance and, thus, improve switching times.

Also, it is a more complex circuit design which uses additional active devices to speed up switching, reduce power consumption, and increase fan-out.

The inverter IC in different TTL sub-families is 7404, 74L04, 74H04, 74LS04, 74AS04, 74ALS04, and 74F04.

#### 16.6.9 Typical TTL Series Characteristics

The typical characteristics of TTL subfamilies are shown in Table 16.2.

**Table 16.2** The typical characteristics of TTL subfamilies

Performance rating	74	74L	74H	74S	74LS	74AS	74ALS
Propagation delay (ns)	9	33	6	3	9.5	1.7	4
Power dissipation (mW)	10	1	23	20	2	8	1.2
Speed-power product (pJ)	90	33	138	60	19	13.6	4.8
Max. clock rate (MHz)	35	3	50	125	45	200	70
Fan-out (same series)	10	20	10	20	20	40	20
Noise margin (V)	0.4	0.4	0.4	0.7	0.7	0.5	0.5

**EXAMPLE 16.4** Determine the maximum average power dissipation and the maximum average propagation delay of a single gate of IC 7400.

**Solution**

From the data sheets of the 7400 NAND IC, the maximum values of  $I_{CCH}$  and  $I_{CCL}$  are 8 mA and 22 mA, respectively. The average  $I_{CC}$  is, therefore,

$$I_{CC}(\text{avg}) = \frac{I_{CCH} + I_{CCL}}{2} = \frac{(8 + 22)\text{mA}}{2} = 15 \text{ mA}$$

The average power is obtained by multiplying  $I_{CC}(\text{avg})$  by  $V_{CC}$ . These  $I_{CC}$  values are obtained when  $V_{CC}$  has its maximum value of 5.25 V. Thus, we have:

The power drawn from the complete IC is,  $P_D(\text{avg}) = 15 \text{ mA} \times 5.25 \text{ V} = 78.75 \text{ mW}$

The power drain of each NAND gate is,  $\frac{P_D(\text{avg})}{4} = \frac{78.75 \text{ mW}}{4} = 19.7 \text{ mW}$

The maximum propagation delays for a 7400 NAND gate are

$$t_{PLH} = 22 \text{ ns} \quad \text{and} \quad t_{PHL} = 15 \text{ ns}$$

so that the average propagation delay is

$$t_{pd}(\text{avg}) = \frac{(22 + 15)\text{ns}}{2} = 18.5 \text{ ns}$$

This is the worst case of maximum possible average propagation delay.

## 16.7 INTEGRATED INJECTION LOGIC (IIL OR I<sup>2</sup>L)

Integrated injection logic (IIL or I<sup>2</sup>L) or current injection logic (CIL) is the newest of the logic families, which is finding widespread use in LSI and VLSI circuits. It is not suitable for discrete gate ICs. The I<sup>2</sup>L logic gates are constructed using bipolar transistors only. The absence of resistors makes it possible to integrate a large number of gates on a single package. Complete microprocessors can be obtained on a single chip. The I<sup>2</sup>L circuits are easily fabricated and are economical. Their power consumption is also low. The speed-power product is constant and very small of the order of 4 pJ, comparable to advanced low power Schottky TTL. The I<sup>2</sup>L has,  $t_{pd} = 1 \text{ ns}$ ,  $P_D = 1 \text{ mW}$ , NM = 0.35 V, fan-out = 8, and the relative cost is very low.

In I<sup>2</sup>L, since the currents are constant, no transients are produced as in TTL and MOS. It can easily be integrated on the same chip with bipolar analog circuits such as op-amps. By programming the injector currents, the propagation delay and power dissipation can be varied over a wide range. The disadvantage is that, it requires one more step in its manufacturing process than those used in MOS.

### 16.7.1 I<sup>2</sup>L Inverter

Since discrete gates are not available in I<sup>2</sup>L, the operation of an I<sup>2</sup>L inverter can be explained by considering the inverter of Figure 16.11a that behaves in the same way as an I<sup>2</sup>L inverter. The p-n-p transistor  $Q_1$  serves as a constant current source that ‘injects’ current into node X. The direction in which the current flows after entering node X depends on the input level. A LOW input is a current sink. When the input is LOW, the injected current flows into the input, thus,

diverting current from the base of  $Q_2$ . Transistor  $Q_2$  is, therefore, OFF and the output is HIGH. If the input is HIGH, the injected current flows into the base of  $Q_2$  turning it ON and making the output LOW as shown in Figure 16.11b. Figure 16.11c shows an actual  $I^2L$  inverter. The output transistor has two collectors (sometimes three), making it equivalent to two transistors with parallel bases and emitters. Thus, it produces two equal outputs. Instead of a collector resistor, the outputs are connected directly to the inputs of other  $I^2L$  gates.

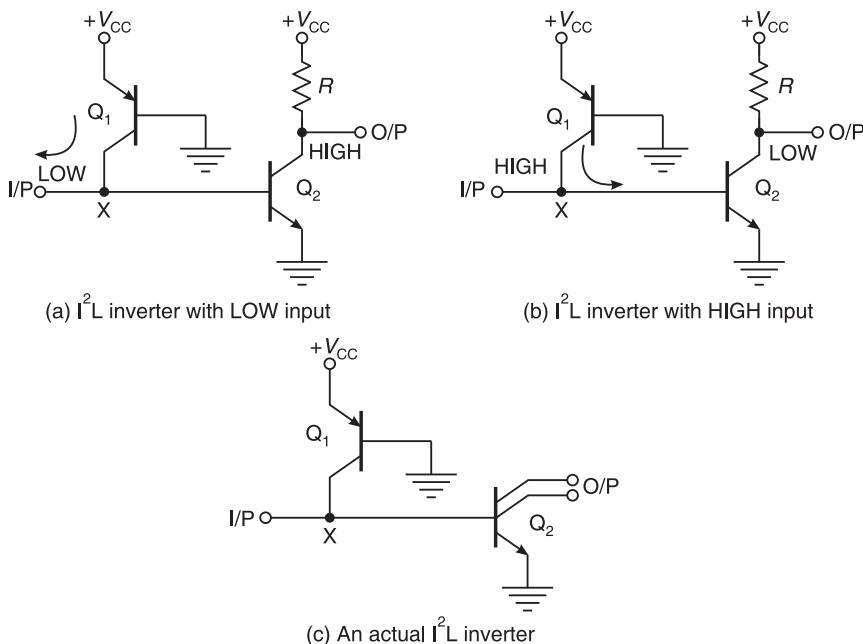


Figure 16.11  $I^2L$  inverters.

### 16.7.2 $I^2L$ NAND Gate

The  $I^2L$  NAND gate shown in Figure 16.12 is simply an inverter with inputs connected directly together at the inverter input. If, either input A, or input B, or both the inputs A and B are LOW (current sinks), the injected current flows into those inputs and  $Q_2$  remains OFF (HIGH). If both the inputs are HIGH, the injected current turns on  $Q_2$  making the output LOW. Thus, the NAND operation is performed. The transistor  $Q_1$  is called a *current injector transistor*, because when its emitter is connected to an external power source, it can supply current to the base of  $Q_2$ .

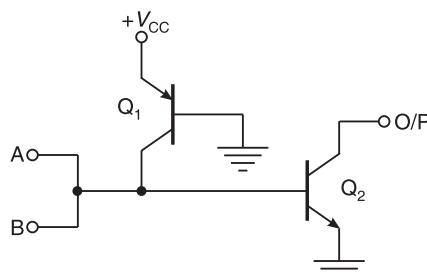
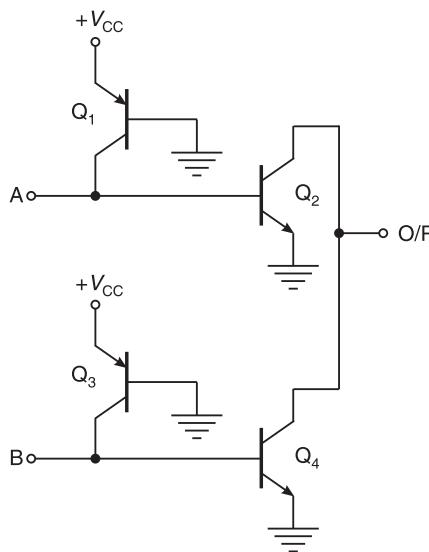


Figure 16.12 Two-input  $I^2L$  NAND gate.

### 16.7.3 I<sup>2</sup>L NOR Gate

The I<sup>2</sup>L NOR gate shown in Figure 16.13 is simply two inverters with their outputs connected together. If either or both the inputs are HIGH, the corresponding output transistor is ON and the output is a current sink. So, the output is LOW. If both the inputs are LOW, both the output transistors are OFF, and so, the output is HIGH. This is a NOR operation.



**Figure 16.13** Two-input I<sup>2</sup>L NOR gate.

## 16.8 Emitter-Coupled Logic (ECL)

Emitter-coupled logic (ECL), also called *current-mode logic* or *current-steering logic*, is the fastest of all logic families because of the following reasons:

1. It is a non-saturated logic, in the sense that the transistors are not allowed to go into saturation. So, storage time delays are eliminated and, therefore, the speed of operation is increased.
2. Currents are kept high, and the output impedance is so low that circuit and stray capacitances can be quickly charged and discharged.
3. The limited voltage swing.

The ECL is so named because of its use of BJTs that are coupled (joined) at their emitters. In ECL, the transistors are prevented from going into saturation when the input changes from LOW to HIGH, by choosing logic levels very close to each other. One disadvantage of having logic levels close to each other is that, it is difficult to achieve good noise immunity. Also, the power consumption is increased since the transistors are not saturated. But the advantage is that the current drawn from the supply is more steady and ECL gates do not experience large switching transients. The ECL family has considerably greater power consumption compared to other families.

The ECL operates on the principle of current switching, whereby a fixed bias current less than  $I_C(\text{sat})$  is switched from one transistor's collector to another. Because of this current-mode

operation, this logic form is also referred to as current-mode logic (CML). It is also called current-steering logic (CSL), because current is steered from one device to another. The ECL family is not as popular and widely used as the TTL and MOS, except in very high frequency applications where its speed is superior. It has the following drawbacks:

1. High cost
2. Low noise margin
3. High power dissipation
4. Its negative supply voltage and logic levels are not compatible with other logic families (making it difficult to use ECL in conjunction with TTL and MOS circuits)
5. Problem of cooling

Still, the ECL is used in superfast computers and high-speed special purpose applications. The ECL gates can be wired ORed, no noise spikes are generated, and complementary outputs are also available. The important characteristics of ECL gates are:

1. Transistors never saturate. So, speed is high with  $t_{pd} = 1$  ns.
2. Logic levels are negative,  $-0.9$  V for a logic 1 and  $-1.7$  V for a logic 0.
3. Noise margin is less, about  $250$  mV. This makes ECL unreliable for use in heavy industrial environment.
4. ECL circuits produce the output and its complement, and therefore, eliminate the need for inverters.
5. Fan-out is large because the output impedance is low. It is about 25.
6. Power dissipation per gate is large,  $P_D = 40$  mW.
7. The total current flow in ECL is more or less constant. So, no noise spikes will be internally generated.

#### 16.8.1 ECL OR/NOR Gate

Figure 16.14a shows a two-input ECL OR/NOR gate. Figure 16.14b shows its I/O characteristics. It has two outputs which are complements of each other. Transistors  $Q_2$  and  $Q_{1A}$  form a differential amplifier. Transistors  $Q_{1A}$  and  $Q_{1B}$  are in parallel. Transistors  $Q_3$  and  $Q_4$  are emitter followers whose emitter voltages are the same as the base voltages (less than  $0.8$  V base to emitter drops). Inputs are applied to  $Q_{1A}$  and  $Q_{1B}$ , and  $Q_2$  is supplied with constant  $-1.3$  V.

When the inputs A and B are both LOW, i.e.  $-1.7$  V,  $Q_2$  is more forward biased than  $Q_{1A}$  and  $Q_{1B}$ , and so,  $Q_2$  is ON and  $Q_{1A}$  and  $Q_{1B}$  are OFF. The value of  $R_2$  is such that current flowing through  $Q_2$  puts the collector at about  $-0.9$  V. Therefore, the emitter of  $Q_4$  is at,  $-0.9 - 0.8 = -1.7$  V, and so, the OR output is LOW. The base current of  $Q_3$  passing through  $R_1$  is very small. The value of  $R_1$  is such that this current puts the collectors of  $Q_{1A}$  and  $Q_{1B}$  at about  $-0.1$  V. So, the emitter of  $Q_3$  is at,  $-0.1 - 0.8 = -0.9$  V, that is, the NOR output is HIGH.

When A is HIGH, or B is HIGH, or both A and B are HIGH, the corresponding transistors are ON, because they are more forward biased than  $Q_2$ , and  $Q_2$  is OFF. So, the collectors of  $Q_{1A}$  and  $Q_{1B}$  are at  $-0.9$  V, which makes the NOR output  $= -0.9 - 0.8 = -1.7$  V, i.e. a logic 0. Only the small base current of  $Q_4$  flows through  $R_2$ . So, the collector of  $Q_2$  is approximately at  $-0.1$  V, and therefore, the OR output is,  $-0.1 - 0.8 = -0.9$  V, i.e. a logic 1. This shows that the above circuit works as an OR/NOR gate. Figure 16.14c shows the logic symbol of a two-input ECL OR/NOR gate.

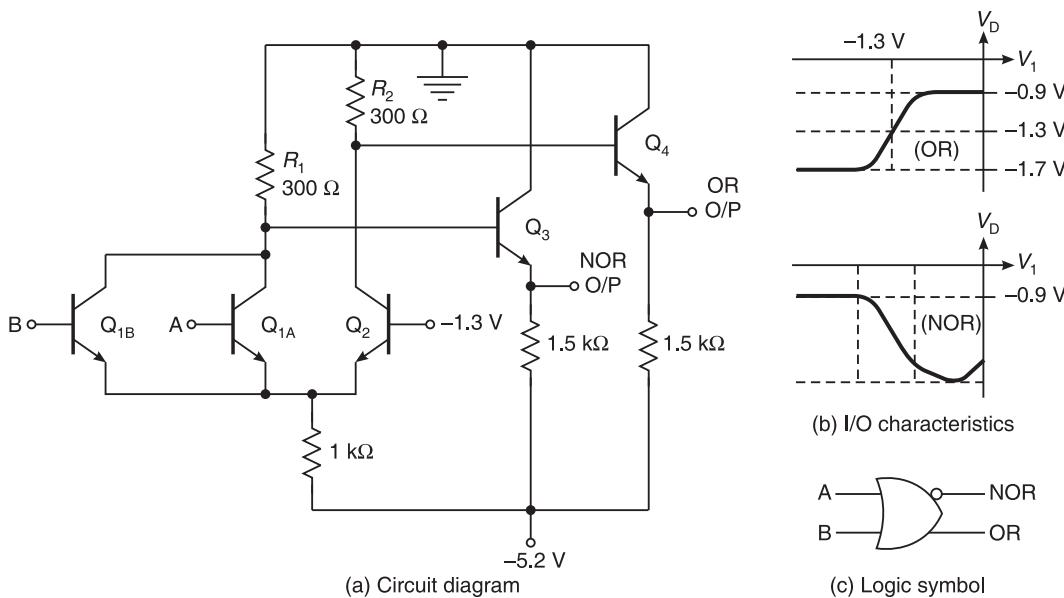


Figure 16.14 Two-input ECL OR/NOR gate.

One advantage of the differential input circuitry in ECL gates is that, it provides common mode rejection—power supply noise common to both sides of the differential configuration is effectively cancelled out (differenced out). Also, since the ECL output is produced at an emitter follower, the output impedance is desirably low. As a consequence, the ECL gates not only have a large fan-out, but also are relatively unaffected by capacitive loads. Some ECL gates are available with multiple outputs, that are derived from multiple emitter transistors in the emitter-follower output. For example, one OR/NOR gate may have two OR outputs and two NOR outputs.

### 16.8.2 ECL Subfamilies

There are many ECL subfamilies. They differ in characteristics such as propagation delay, power dissipation per gate, and speed–power product. The ECL subfamilies do not include as wide a range of general purpose logic gates as do TTL and CMOS families. They do, however, include many complex special purpose circuits used in high speed digital data transmission, arithmetic units, and memories.

The first ECL series marketed by Motorola was the MECL-I series. It was followed by MECL-II series. Both these series are now obsolete. The MECL-III series carrying MC1600 numbers, the MECL10K series carrying MC10000 numbers, and the recent MECL10KH series with MC10H000 numbers are in use presently. The MECL10KH series has a  $t_{pd}$  of 1 ns and  $P_D$  of 25 mW, giving a speed–power product of 25 pJ (least of all ECL series).

### 16.8.3 Wired OR Connections

The ECL gates are available with open-emitter outputs, that is, with resistors in the output emitter followers omitted. The open-emitter outputs can be connected together directly, and the common emitter output terminal may be connected through an external resistor to a negative supply voltage

( $-5.2\text{ V}$ ) as shown in Figure 16.15a to perform a wired OR operation. The transistors labelled  $Q_3$  are the output transistors of gates 1 and 2. The truth table of the wired OR gate is shown in Figure 16.15b.

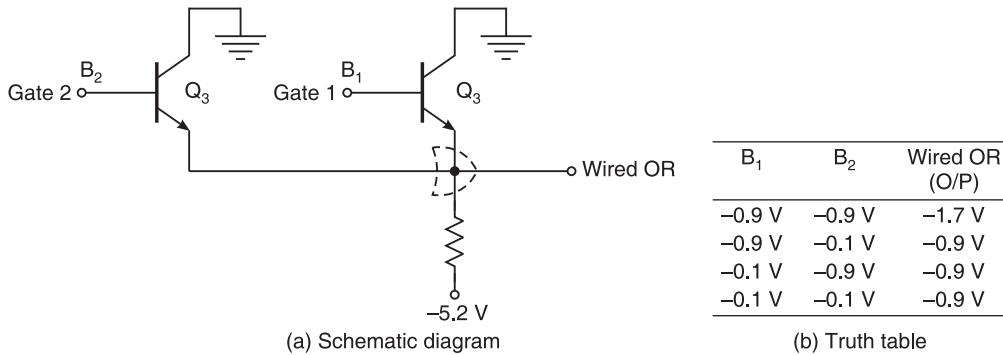


Figure 16.15 Wired OR operation of ECL gates.

When the bases of both the transistors are at  $-0.9\text{ V}$ , both the transistors conduct and make the common emitter voltage to be,  $-0.9\text{ V} - 0.8\text{ V} = -1.7\text{ V}$ . When both the bases are at  $-0.1\text{ V}$ , again both the transistors conduct and make the output voltage to be,  $-0.1\text{ V} - 0.8\text{ V} = -0.9\text{ V}$ . When only one base is at  $-0.1\text{ V}$  and the other at  $-0.9\text{ V}$ , the output transistor with  $-0.1\text{ V}$  base voltages conducts and makes the common emitter voltage  $-0.9\text{ V}$  preventing the second transistor from conducting. Hence, the circuit provides OR operation.

#### 16.8.4 Interfacing ECL Gates

Since ECL logic levels are so different from those of TTL and CMOS circuits, interfacing ECL gates with other logic families requires special level shifter circuits called *level translators*. Level translators are available in various ECL series to facilitate interfacing of ECL with other families.

**EXAMPLE 16.5** What are the logic levels at the output of the ECL gate shown in Figure 16.16?

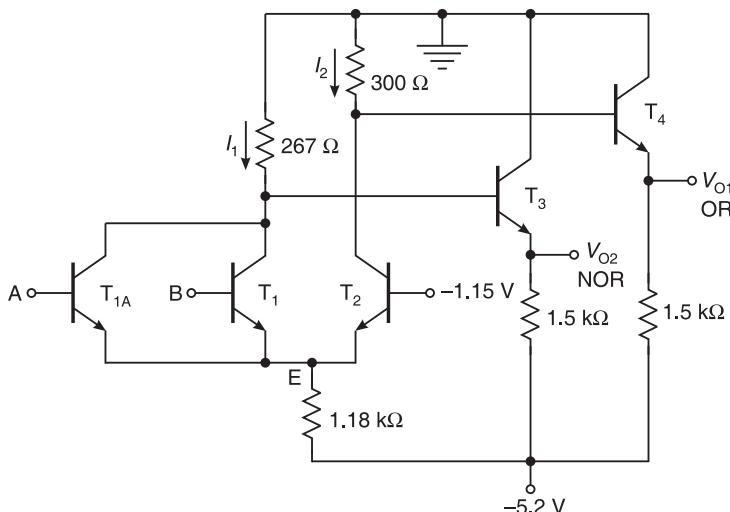


Figure 16.16 Example 16.5: ECL gate.

**Solution**

(a) To calculate the logic levels, the input voltage magnitudes need to be known. Since they are not given, assume them to be  $-0.8$  V (HIGH) and  $-1.5$  V (LOW).  $V_R$  ( $-1.15$  V) is the approximate average of the two input levels. When  $A = -0.8$  V and  $B = -0.8$  V, or  $A = -0.8$  V and  $B = -1.5$  V, or  $A = -1.5$  V and  $B = -0.8$  V,  $T_2$  is OFF and  $T_1$  or  $T_{1A}$  or both will be ON. Therefore,  $V_E = -0.8$  V  $- 0.7$  V  $= -1.5$  V and

$$I_E = \frac{[-1.5 - (-5.2)]V}{1.18 \text{ k}\Omega} \approx 3.1 \text{ mA}$$

Therefore,

$$I_1 = I_E \frac{h_{FE}}{1 + h_{FE}} \approx 3.1 \text{ mA}$$

$$V_{C1} = 0 - (267 \times 3.1 \times 10^{-3}) = -0.827 \text{ V}$$

$$V_{O2} = (-0.827 - 0.7)V = -1.527 \text{ V}$$

$V_{O2}$  is the OR output  $= -1.527$  V (logic 0)

When  $T_2$  is OFF,  $I_2$  is the small base current of  $T_4$  given by

$$I_2 = \frac{[0 - 0.7 - (-5.2)]}{[300 + 1.5 \times 10^3(1 + h_{FE})]\Omega} = \frac{4.5 \text{ V}}{(300 + 1500 \times 41)\Omega} = 0.0728 \text{ mA}$$

$$V_{C2} = 0 - (300 \times I_2) = (-300 \times 0.0728)V = -0.0218 \text{ V}$$

Therefore,

$$V_{O1} = (-0.0218 - 0.7)V = -0.7218 \text{ V (logic 1)}$$

When  $A = -1.5$  V and  $B = -1.5$  V, both  $T_1$  and  $T_{1A}$  are OFF and  $T_2$  is ON.

When  $T_1$  and  $T_{1A}$  are OFF,  $I_1$  is the small base current of  $T_3$  given by

$$I_1 = \frac{[0 - 0.7 - (-5.2)]}{[267 + 1.5 \times 10^3(1 + h_{FE})]\Omega} = \frac{4.5 \text{ V}}{(267 + 1500 \times 41)\Omega} = 0.07285 \text{ mA}$$

$$V_{C1} = 0 - (0.07285 \times 10^{-3} \times 267) = -0.0195 \text{ V}$$

Therefore,

$$V_{O2} = -0.0195 - 0.7 = -0.7195 \text{ V (logic 1)}$$

Since  $T_2$  is ON,  $V_E = -1.15 - 0.7 = -1.85$  V

Therefore,

$$I_E = \frac{[1.85 - (-5.2)]V}{1.18 \text{ k}\Omega} = 2.84 \text{ mA}$$

and

$$I_2 = I_E \frac{h_{FE}}{1 + h_{FE}} \approx I_E = 2.84 \text{ mA}$$

Therefore,

$$V_{C2} = 0 - (2.84 \times 10^{-3} \times 300) = -0.852 \text{ V}$$

$$V_{O1} = -0.852 - 0.7 = -1.56 \text{ V (logic 0)}$$

(b) To show that the transistors do not saturate, find the  $V_{CE}$  of the transistors.

When  $T_2$  is conducting, from the above calculations we see that its collector is at  $-0.852$  V and its emitter is at  $-1.52$  V. Therefore  $V_{CE} = -0.852 - (-1.52) \approx 1$  V. Since  $V_{CE} \approx 1$  V, the transistor is in the active region only.

Similarly, when  $T_1$  or  $T_{1A}$  or both are ON,  $V_{C1} = -0.847$  V and  $V_E = -1.5$  V.

$$\text{Therefore, } V_{CE} = -0.847 - (-1.5) = +0.653 \text{ V}$$

So, the transistor is in the active region only.

(c) Noise margins:

For an ECL gate, the limits of transition region are  $-1.1$  V and  $-1.25$  V.

In the problem, we got logic HIGH as  $-0.7218$  V and logic LOW as  $-1.52$  V.

Therefore,

$$\text{High level noise margin} < 1 = -0.73 \text{ V} - (-1.1 \text{ V}) = +0.37 \text{ V}$$

$$\text{Low level noise margin} < 0 = -1.25 \text{ V} - (-1.52 \text{ V}) = +0.27 \text{ V}$$

These noise margins are typical and not worst case values.

## 16.9 METAL OXIDE SEMICONDUCTOR (MOS) LOGIC

The MOS logic is so named because it uses metal oxide semiconductor field effect transistors (MOSFETs). Compared to the bipolar logic families, the MOS families are simpler and inexpensive to fabricate, require much less power, have a better noise margin, a greater supply voltage range, a higher fan-out, and require much less chip area. But they are slower in operating speed and are susceptible to static charge damage. For MOS logic,  $t_{pd} = 50$  ns,  $NM = 1.5$  V (for  $+5$  V supply),  $P_D = 0.1$  mW, and fan-out = 50 for frequencies greater than 100 Hz and it is virtually unlimited for dc or low frequencies. The propagation delay associated with MOS gates is large (50 ns) because of their high output resistance ( $100 \text{ k}\Omega$ ) and capacitive loading presented by the driven gates.

The MOS logic is the simplest to fabricate and occupies very small space, because it requires only one basic element—an NMOS or a PMOS transistor. It does not require other elements like resistors and diodes, which occupy large space. Because of its ease of fabrication and lower power dissipation per gate  $P_D$ , it is ideally suited for LSI, VLSI, and ULSI for dedicated applications such as large memories, calculator chips, large microprocessors, etc. The operating speed of MOS is slower than that of TTL, so, they are hardly used in SSI and MSI applications. The greater packing density of MOS ICs results in higher reliability because of the reduction in the number of external connections.

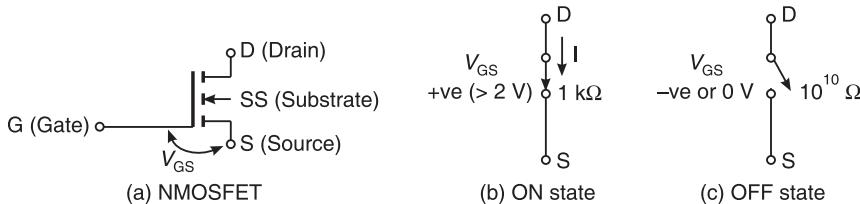
Because of the very high impedance present at a MOSFET's input, the MOS logic families are more susceptible to static charge damage. The CMOS family is less susceptible to static charge damage.

There are presently two general types of MOSFETs—*depletion type* and *enhancement type*. The MOS digital ICs use enhancement MOSFETs exclusively. The MOSFETs can be of NMOS type or PMOS type. Most modern MOSFET circuitry is constructed using NMOS devices, because they operate at about three times the speed of their PMOS counterparts, and also have twice the packing density of PMOS.

Both NMOS and PMOS have greater packing density than that of CMOS, and are therefore, more economical than CMOS. The CMOS family has the greatest complexity and the lowest packing density of all the MOS families, but it possesses the important advantages of higher speed and much lower power dissipation. The CMOS can be operated at high voltage resulting in improved noise margin.

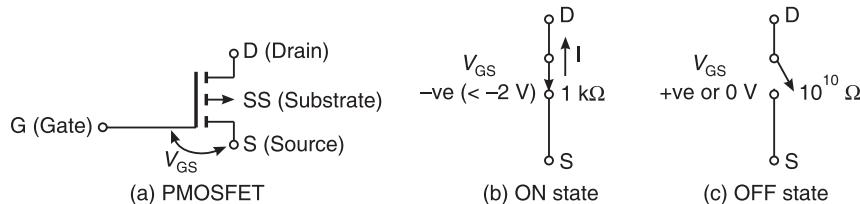
### 16.9.1 Symbols and Switching Action of NMOS and PMOS

Figure 16.17a shows the circuit symbol of NMOSFET. Figure 16.17b shows its equivalent as a closed switch when it is ON and Figure 16.17c shows its equivalent as an open switch when it is OFF.



**Figure 16.17** Circuit symbol and ON and OFF equivalents of NMOSFET.

Figure 16.18a shows the circuit symbol of PMOSFET. Figure 16.18b shows its equivalent as a closed switch when it is ON and Figure 16.18c shows its equivalent as an open switch when it is OFF.

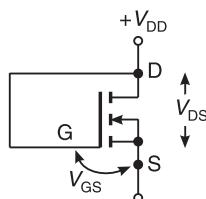


**Figure 16.18** Circuit symbol and ON and OFF equivalents of PMOSFET.

The arrow in the symbols of MOSFETs indicates either P or N channel. In the channel, the broken line between the source and the drain indicates that normally there is no conducting channel between these electrodes. The separation between the gate and the other terminals indicates the existence of very high resistance ( $10,000\text{ M}\Omega$ ) between the gate and the channel. The switch in a MOSFET is between the drain and source terminals. The gate-to-source voltage  $V_{GS}$  controls the switch. In an N-channel MOSFET, switch closes and current flows from drain to source when  $V_{GS}$  is positive, and switch opens when  $V_{GS}$  is negative or zero w.r.t. the source.

### 16.9.2 Resistor

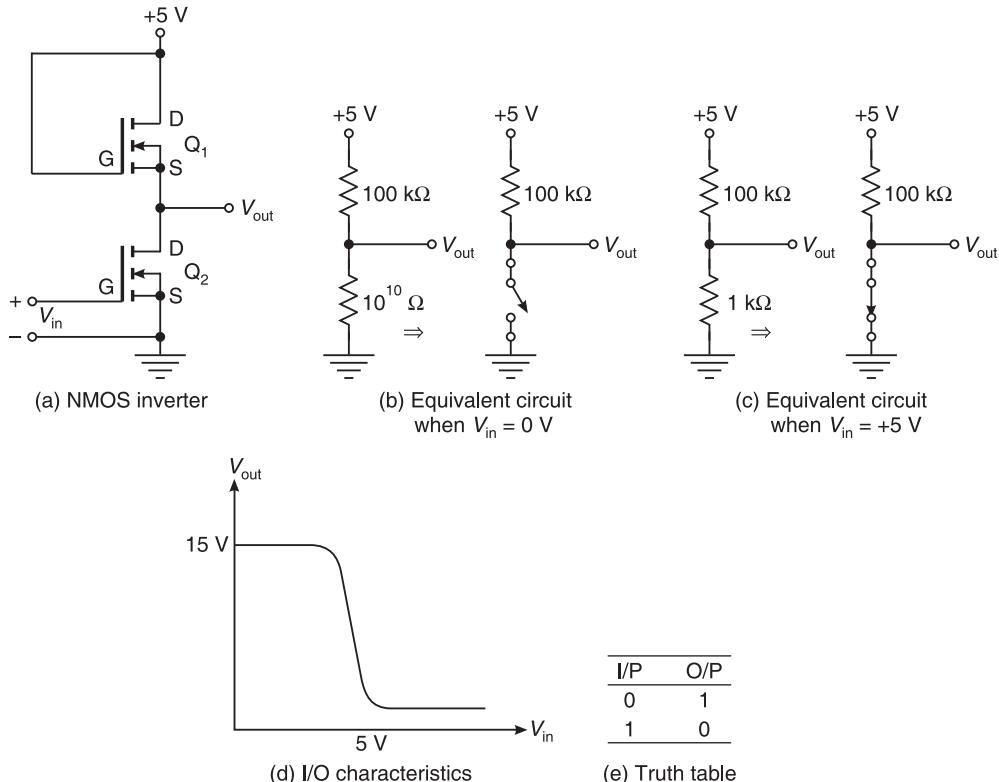
A MOS transistor can be connected as a resistor as shown in Figure 16.19. The value of the resistance presented by a resistor-connected NMOS device depends on the current through it. The gate is permanently connected to  $+5\text{ V}$ , and so, it is always in the ON state and the transistor acts as a resistor of value  $R_{ON}$ . The load resistor is designed to have a narrower channel, so, its  $R_{ON}$  is much greater than the  $R_{ON}$  of the switching transistor. Typically, its  $R_{ON} = 100\text{ k}\Omega$ .



**Figure 16.19** NMOS connected as a resistor.

### 16.9.3 NMOS Inverter

The basic NMOS inverter shown in Figure 16.20 contains two N-channel MOSFETs.  $Q_1$  is called the load MOSFET and  $Q_2$  the switching MOSFET.  $Q_2$  will switch from ON to OFF in response to  $V_{in}$ . These two MOSFETs can be considered as resistors and the circuit as a potential divider.



**Figure 16.20** Circuit diagram and equivalent circuits for various inputs of the NMOS inverter.

- When  $V_{in} = 0 \text{ V}$ ,  $Q_2$  is OFF. So, its  $R_{OFF} = 10^{10} \Omega$ , and the equivalent circuit (b) results. Therefore,

$$V_{out} = \frac{V_{DD}R_{OFF}(Q_2)}{R_{ON}(Q_1) + R_{OFF}(Q_2)} \approx \frac{5 \times 10^{10}}{100 \times 10^3 + 10^{10}} \approx 5 \text{ V}$$

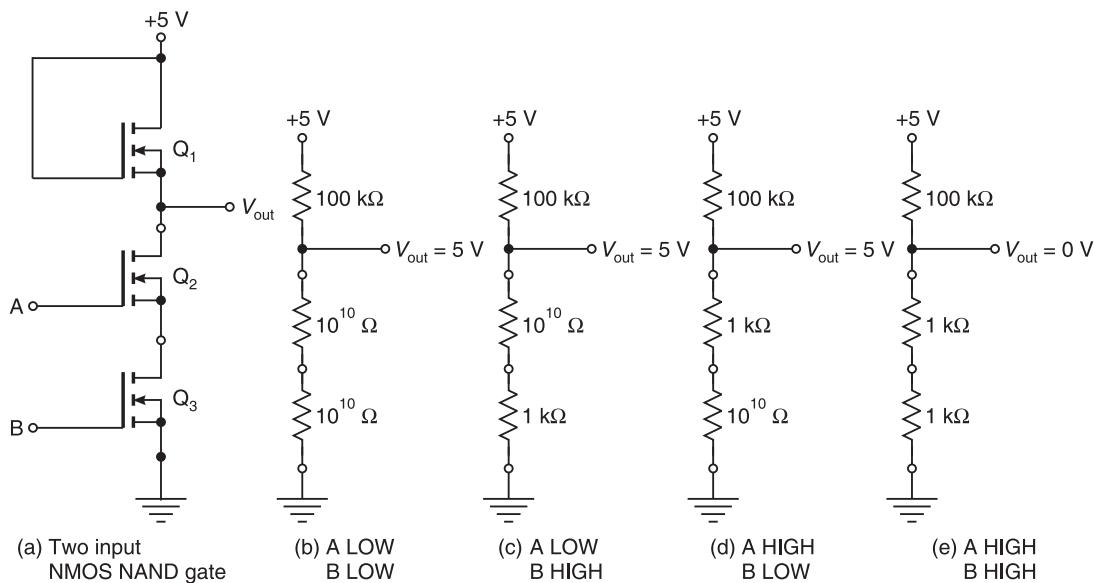
- When  $V_{in} = 5 \text{ V}$ ,  $Q_2$  is ON. So, its  $R_{ON} = 1 \text{k}\Omega$ , and the equivalent circuit (c) results. Therefore,

$$V_{out} = \frac{V_{DD}R_{ON}(Q_2)}{R_{ON}(Q_1) + R_{ON}(Q_2)} = \frac{5 \times 1}{100 + 1} \approx 0 \text{ V}$$

This shows that the above circuit acts as an inverter. The I/O characteristics and the truth table are shown in Figures 16.20d and e, respectively.

### 16.9.4 NMOS NAND Gate

Figure 16.21 shows an NMOS two-input NAND gate and its equivalent circuits for different possible combinations of inputs in terms of resistance values of transistors in ON and OFF positions.



**Figure 16.21** Circuit diagram and equivalent circuits for various inputs of the NMOS NAND gate.

In the NMOS NAND gate shown,  $Q_1$  is acting as a load resistor and  $Q_2$  and  $Q_3$  as switches controlled by input levels A and B, respectively.

- When both A and B are 0 V, both  $Q_2$  and  $Q_3$  are OFF. So, the equivalent circuit (b) results with  $V_{out} = +5$  V.
- When  $A = 0$  V and  $B = +5$  V,  $Q_2$  is OFF and  $Q_3$  is ON. So, the equivalent circuit (c) results with  $V_{out} = +5$  V.
- When  $A = +5$  V and  $B = 0$  V,  $Q_2$  is ON and  $Q_3$  is OFF. So, the equivalent circuit (d) results with  $V_{out} = +5$  V.
- When  $A = +5$  V and  $B = +5$  V, both  $Q_2$  and  $Q_3$  are ON. So, the equivalent circuit (e) results with  $V_{out} = 0$  V.

Thus, the above circuit works as a positive logic two-input NAND gate.

### 16.9.5 NMOS NOR Gate

Figure 16.22 shows an NMOS two-input NOR gate and its equivalent circuits for different possible combinations of inputs in terms of resistance values of transistors in ON and OFF positions.  $Q_1$  is the resistor-connected NMOS transistor that serves as a load and  $Q_2$  and  $Q_3$  are the switching transistors controlled by the inputs A and B, respectively.

- When A is LOW and B is LOW,  $Q_2$  is OFF and  $Q_3$  is OFF. So, the equivalent circuit (b) results with  $V_{out} = +5$  V.

- When A is LOW and B is HIGH,  $Q_2$  is OFF and  $Q_3$  is ON. So, the equivalent circuit (c) results with  $V_{out} = 0$  V.
- When A is HIGH and B is LOW,  $Q_2$  is ON and  $Q_3$  is OFF. So, the equivalent circuit (d) results with  $V_{out} = 0$  V.
- When A is HIGH and B is HIGH,  $Q_2$  is ON and  $Q_3$  is ON. So, the equivalent circuit (e) results with  $V_{out} = 0$  V.

Thus, the above circuit works as a positive logic two-input NOR gate. The truth table is shown in Figure 16.22f.

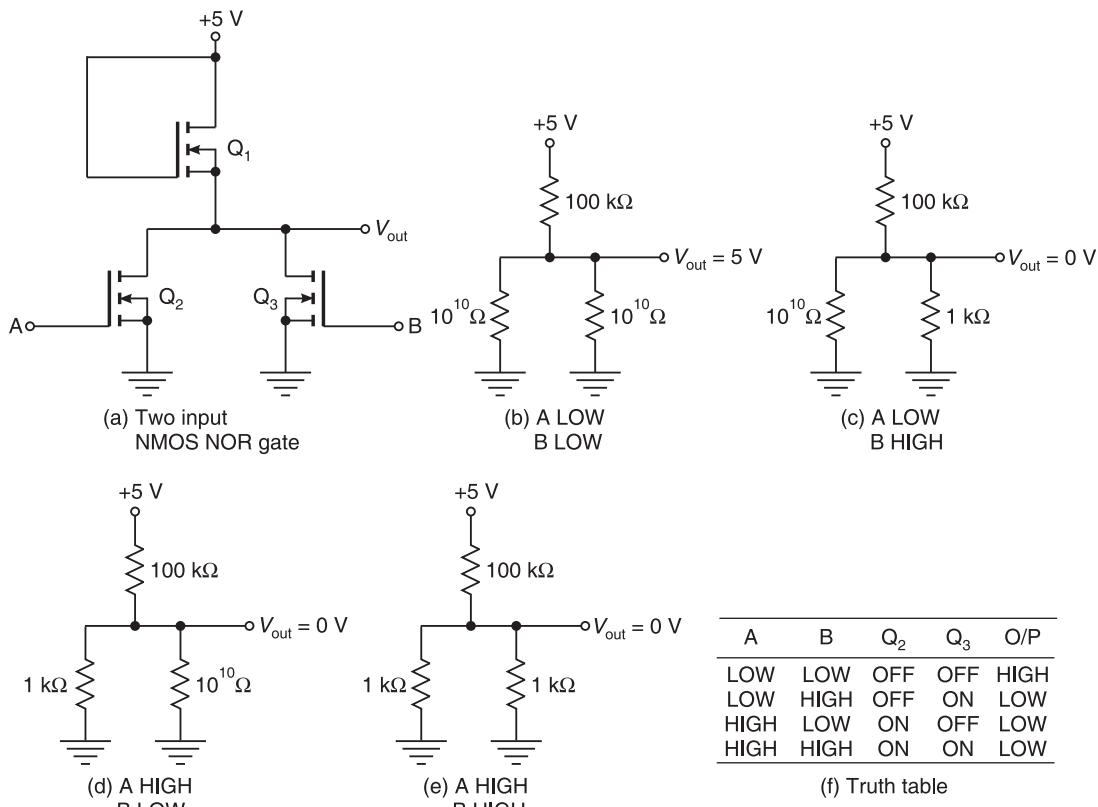


Figure 16.22 Circuit diagram and equivalent circuits for various inputs of the NMOS NOR gate.

## 16.10 COMPLEMENTARY METAL OXIDE SEMICONDUCTOR (CMOS) LOGIC

The CMOS logic family uses both P and N channel MOSFETs in the same circuit to realize several advantages over the PMOS and NMOS families. The CMOS family is faster and consumes less power than the other MOS families. These advantages are offset somewhat by the increased complexity of the IC fabrication process and a lower packing density. The CMOS can be operated at higher voltages resulting in improved noise immunity. It is widely used for general purpose logic circuitry. The CMOS technology has been used to construct small, medium, and large scale

ICs for a wide variety of applications ranging from general-purpose logic to microprocessors. Because of its extremely small power consumption, it is useful for applications in watches and calculators. The CMOS, however, cannot yet compete with MOS in applications requiring the utmost in LSI. The CMOS has very high input resistance. Thus, it draws almost zero current from the driving gate, and therefore, its fan-out is very high. Its output resistance is small ( $1\text{ k}\Omega$ ) compared to that of NMOS ( $100\text{ k}\Omega$ ). Hence, it is faster than NMOS. The speed of CMOS decreases with increase in load. In CMOS, there is always a very high resistance between the  $V_{DD}$  terminal and ground, because of the MOSFET in the current path. Hence, its power consumption is very low. The noise margin of CMOS is the same in both the LOW and HIGH states and it is 30% of  $V_{DD}$ , indicating that noise margin increases with an increase in power supply voltage. So in noisy environments, CMOS with large  $V_{DD}$  is preferred. However, an increase in  $V_{DD}$  results in the corresponding increase in  $P_D$ . The CMOS loses some of its advantages at high frequencies.

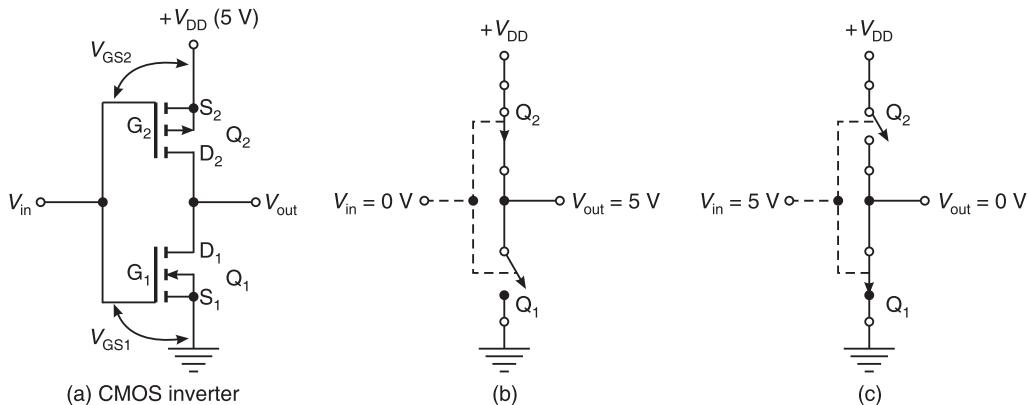
In MSI, the CMOS is also competitive with TTL. The CMOS fabrication process is simpler than that of the TTL and it has greater packing density, thereby permitting more circuitry in a given area and reducing the cost per function. The CMOS uses only a fraction of the power needed even for low power TTL and is, thus, ideally suited for applications requiring battery power or battery backup power. The CMOS is, however, generally slower than TTL.

### 16.10.1 CMOS Inverter

Figure 16.23 shows a CMOS inverter and its equivalent circuits for different inputs. It consists of an NMOS transistor  $Q_1$  and a PMOS transistor  $Q_2$ . The input is connected to the gates of both the devices and the output is at the drain of both the devices. The positive supply voltage is connected to the source of the PMOS transistor  $Q_2$ , and the source of  $Q_1$  is grounded.

- When  $V_{in} = 0\text{ V}$  (LOW),  $V_{GS2} = -5\text{ V}$ , and  $V_{GS1} = 0\text{ V}$ . So,  $Q_2$  is ON and  $Q_1$  is OFF. Therefore, the switching circuit (b) results with  $V_{out} = 5\text{ V}$ .
- When  $V_{in} = +5\text{ V}$  (HIGH),  $V_{GS2} = 0\text{ V}$  and  $V_{GS1} = +5\text{ V}$ . So,  $Q_2$  is OFF and  $Q_1$  is ON. Therefore, the switching circuit (c) results with  $V_{out} = 0\text{ V}$ .

Thus, the above circuit acts as an inverter.



**Figure 16.23** Circuit diagram and equivalent circuits for various inputs of the CMOS inverter.

### 16.10.2 CMOS NAND Gate

Figure 16.24 shows a CMOS two-input NAND gate and its equivalent circuits for various input combinations. Here,  $Q_1$  and  $Q_2$  are parallel-connected PMOS transistors, and  $Q_3$  and  $Q_4$  are series-connected NMOS transistors.

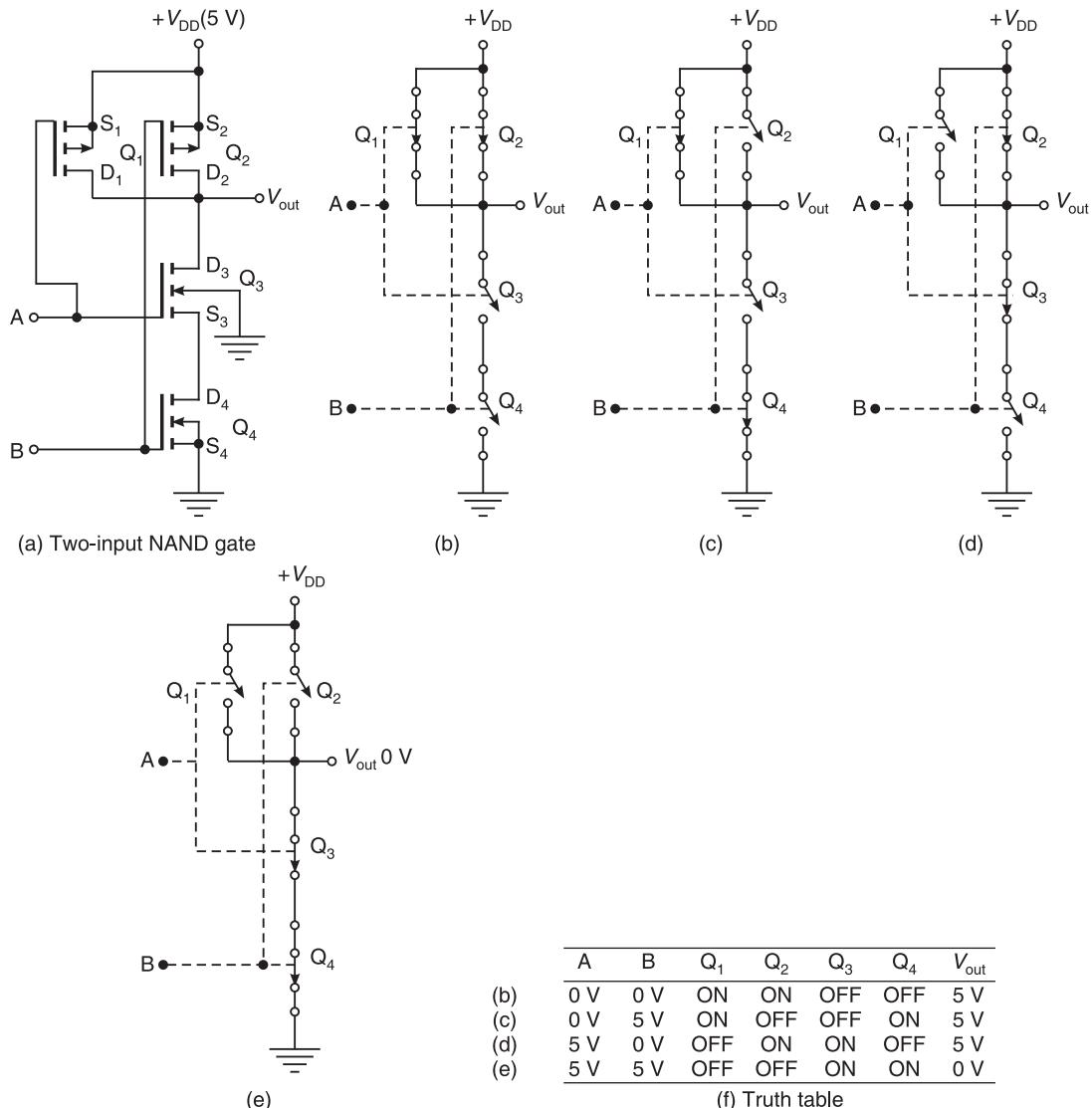


Figure 16.24 Circuit diagram and equivalent circuits for various inputs of the CMOS NAND gate.

- When  $A = 0 \text{ V}$  and  $B = 0 \text{ V}$ ,  $V_{GS1} = V_{GS2} = -5 \text{ V}$ ,  $V_{GS3} = V_{GS4} = 0 \text{ V}$ . So,  $Q_1$  is ON,  $Q_3$  is OFF,  $Q_2$  is ON and  $Q_4$  is OFF. Thus, the switching circuit (b) results with  $V_{out} = +5 \text{ V}$ .
- When  $A = 0 \text{ V}$  and  $B = +5 \text{ V}$ ,  $V_{GS1} = -5 \text{ V}$ ,  $V_{GS2} = 0 \text{ V}$ ,  $V_{GS3} = 0 \text{ V}$ ,  $V_{GS4} = 5 \text{ V}$ . So,  $Q_1$  is ON,  $Q_3$  is OFF,  $Q_2$  is OFF and  $Q_4$  is ON. Thus, the switching circuit (c) results with  $V_{out} = +5 \text{ V}$ .

- When  $A = +5\text{ V}$  and  $B = 0\text{ V}$ ,  $V_{GS1} = 0\text{ V}$ ,  $V_{GS2} = -5\text{ V}$ ,  $V_{GS3} = 5\text{ V}$ ,  $V_{GS4} = 0\text{ V}$ . So,  $Q_1$  is OFF,  $Q_3$  is ON,  $Q_2$  is ON and  $Q_4$  is OFF. Thus, the switching circuit (d) results with  $V_{out} = +5\text{ V}$ .
- When  $A = +5\text{ V}$  and  $B = +5\text{ V}$ ,  $V_{GS1} = V_{GS2} = 0\text{ V}$ ,  $V_{GS3} = V_{GS4} = 5\text{ V}$ . So,  $Q_1$  is OFF,  $Q_3$  is ON,  $Q_2$  is OFF and  $Q_4$  is ON. Thus, the switching circuit (e) results with  $V_{out} = 0\text{ V}$ .

Thus, the circuit works as a two-input NAND gate. The truth table is shown in Figure 16.24f.

### 16.10.3 CMOS NOR Gate

Figure 16.25 shows a CMOS two-input NOR gate and its equivalent circuits for various input combinations. Here, the NMOS transistors  $Q_3$  and  $Q_4$  are connected in parallel and the PMOS transistors  $Q_1$  and  $Q_2$  in series.

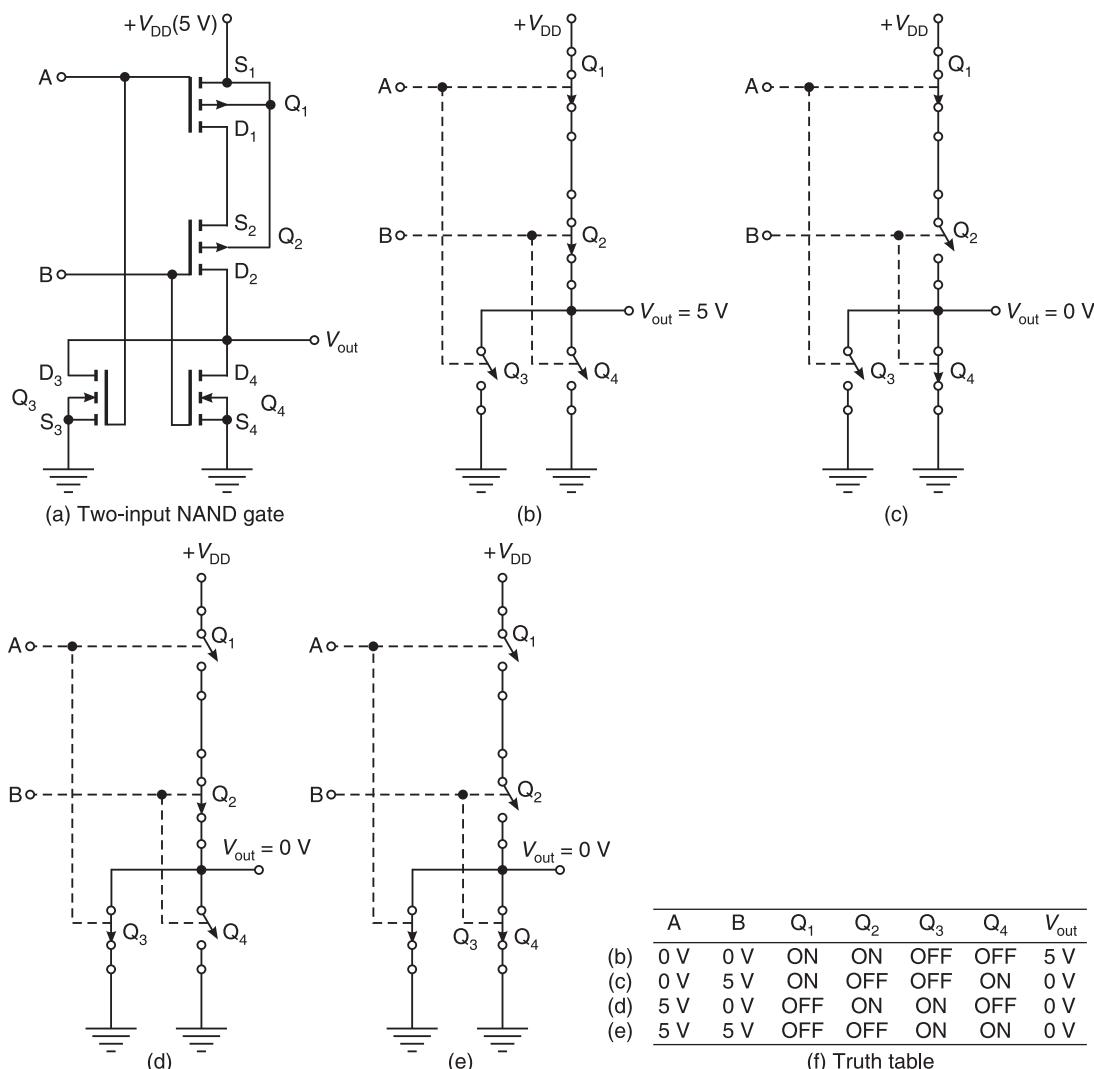


Figure 16.25 Circuit diagram and equivalent circuits for various inputs of the CMOS NOR gate.

The operation of the CMOS NOR gate can be explained as follows:

- When  $A = 0\text{ V}$  and  $B = 0\text{ V}$ ,  $V_{GS1} = V_{GS2} = -5\text{ V}$ ,  $V_{GS3} = V_{GS4} = 0\text{ V}$ . So,  $Q_1$  and  $Q_2$  are ON, and  $Q_3$  and  $Q_4$  are OFF. Thus, the equivalent circuit (b) results with  $V_{out} = +5\text{ V}$ .
- When  $A = 0\text{ V}$  and  $B = +5\text{ V}$ ,  $V_{GS1} = -5\text{ V}$ ,  $V_{GS2} = 0\text{ V}$ ,  $V_{GS3} = 0\text{ V}$ ,  $V_{GS4} = 5\text{ V}$ . So,  $Q_1$  and  $Q_4$  are ON, and  $Q_2$  and  $Q_3$  are OFF. Thus, the equivalent circuit (c) results with  $V_{out} = 0\text{ V}$ .
- When  $A = +5\text{ V}$  and  $B = 0\text{ V}$ ,  $V_{GS1} = 0\text{ V}$ ,  $V_{GS2} = -5\text{ V}$ ,  $V_{GS3} = 5\text{ V}$ ,  $V_{GS4} = 0\text{ V}$ . So,  $Q_1$  and  $Q_4$  are OFF, and  $Q_2$  and  $Q_3$  are ON. Thus, the equivalent circuit (d) results with  $V_{out} = 0\text{ V}$ .
- When  $A = +5\text{ V}$  and  $B = +5\text{ V}$ ,  $V_{GS1} = V_{GS2} = 0\text{ V}$ ,  $V_{GS3} = V_{GS4} = 5\text{ V}$ . So,  $Q_1$  and  $Q_2$  are OFF, and  $Q_3$  and  $Q_4$  are ON. Thus, the equivalent circuit (e) results with  $V_{out} = 0\text{ V}$ .

The above analysis shows that the circuit works as a two-input NOR gate. The truth table is shown in Figure 16.25f.

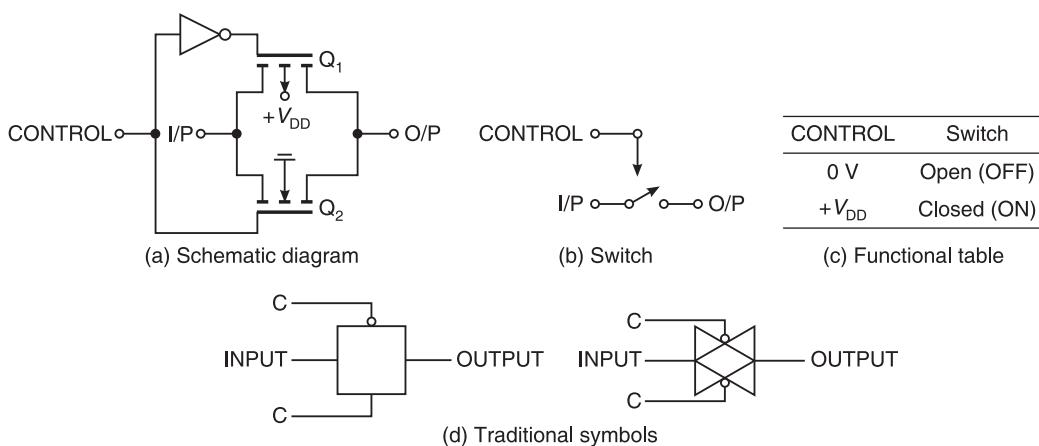
#### 16.10.4 Buffered and Unbuffered Gates

Some metal gate CMOS circuits are available in buffered and unbuffered versions. The gates in buffered circuits have CMOS inverters in series with their outputs to suppress switching transients and to improve the sharpness of the voltage transition at the output. The gates discussed above are unbuffered gates.

#### 16.10.5 Transmission Gate

A transmission gate is simply a digitally controlled CMOS switch. When the switch is open (OFF), the impedance between its terminals is very large. It is used to implement special logic functions. Since the CMOS gate can transmit signals in both directions, it is called a *bilateral transmission gate*. It is also called a bilateral switch. It is useful for digital and analog applications. The TTL and ECL gates are essentially unidirectional.

Figure 16.26 shows the schematic diagram and logic symbols of a CMOS transmission gate. The NMOS and PMOS transistors are connected in parallel. So, both polarities of input voltages can be switched. The CONTROL signal is connected to the NMOSFET and its inverse is connected to the PMOSFET. When the CONTROL is HIGH, the gate of PMOSFET  $Q_1$  is LOW and the gate



**Figure 16.26** Circuit diagram and logic symbols of the CMOS transmission gate.

of NMOSFET  $Q_2$  is HIGH. If the input (data) is LOW,  $V_{GS1}$  is 0 V and  $V_{GS2}$  is positive. So,  $Q_1$  is OFF and  $Q_2$  is ON. If the input is HIGH,  $V_{GS1}$  is negative and  $V_{GS2}$  is 0 V. So,  $Q_1$  is ON and  $Q_2$  is OFF. Thus, there is always one conducting path from input to output when the CONTROL is HIGH.

On the other hand, when the CONTROL is LOW, the gate of PMOSFET  $Q_1$  is HIGH and the gate of NMOSFET  $Q_2$  is LOW. If the input (data) is LOW,  $V_{GS1}$  is positive and  $V_{GS2}$  is 0 V. Therefore,  $Q_1$  is OFF and  $Q_2$  is also OFF. If the input (data) is HIGH,  $V_{GS1}$  is 0 V and  $V_{GS2}$  is negative. So, again  $Q_1$  is OFF and  $Q_2$  is also OFF. Thus, there is no conducting path from input to output when the CONTROL is LOW.

So, we can conclude that when the CONTROL is HIGH, the circuit acts as a closed switch and allows the transmission of the signal from input to output. When the CONTROL is LOW, the circuit acts as an open switch and blocks the transmission of the signal from input to output. The CONTROL acts as an active-HIGH enabling signal. Active-LOW enabling is possible, if the CONTROL is connected to the gate of PMOS and to the gate of NMOS.

Since the input and output terminals can be interchanged, the circuit can also transmit signals in the opposite direction. Hence, it acts as a bilateral switch.

#### 16.10.6 Open Drain and High Impedance Outputs

The CMOS logic gates are available with open-drain outputs similar to their TTL counter-parts with open-collector outputs. In these devices, the output stage consists only of an N-channel MOSFET whose drain is unconnected, since the upper P-channel MOSFET has been eliminated. An external pull-up resistor is needed to produce a HIGH state voltage level. Like open-collector outputs, the open-drain outputs can be wired ANDed. Figure 16.27a shows two inverters at the output of a CMOS gate that are used to provide the buffering. As shown in Figure 16.27b, the open-drain output is obtained by omitting the PMOS transistor in the output inverter. Diode  $D_1$  is connected internally to provide protection from electro-static discharge.

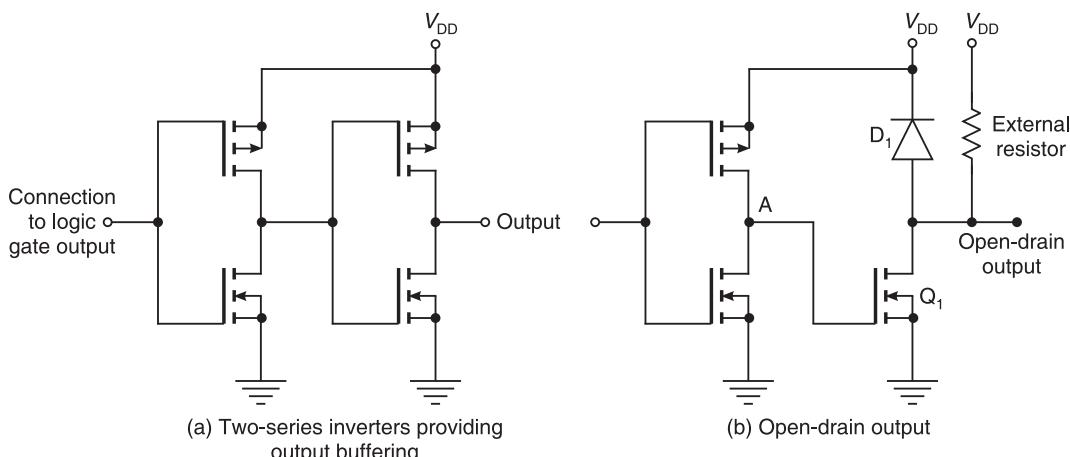


Figure 16.27 Open-drain CMOS inverter.

### 16.10.7 Interfacing CMOS and TTL Devices

Specially designed ICs called level shifters are available to make devices from different logic families compatible with each other. The 74C series of CMOS is compatible pin for pin and function for function with TTL devices having the same number.

### 16.10.8 CMOS Series

**4000/14000 series:** It was the first CMOS series. The original 4000 series is now the 4000A series. The 4000B series is an improved version of the 4000A. The 4000B series has a higher output current capability than that of the 4000A series. The 4000A and 4000B series are still widely used despite the emergence of the new CMOS series.

**74C series:** This series is compatible pin for pin and function for function with TTL devices having the same number. Many, but not all, functions that are available in TTL are also available in this CMOS series. This makes it possible to replace some TTL circuits by an equivalent CMOS design. The performance characteristics of the 74C series are about the same as those of the 4000 series.

**74HC series (High speed series):** This is an improved version of the 74C series. High speed (10 times that of the 74C series) and higher output current capability are its main features. The speed of the devices in this series is compatible with that of the 74LS TTL series.

**74 HCT series:** This is also a high speed CMOS series and it is designed to be voltage compatible with TTL devices. In other words, it can be directly driven by a TTL output.

### 16.10.9 Operating and Performance Characteristics of CMOS

**Supply voltage:** The 4000 and 74C series can operate with  $V_{DD}$  values ranging from 3 to 15 volts. The 74HC and 74HCT series can operate with  $V_{DD}$  values ranging from 2 to 6 volts.

**Voltage levels:** When a CMOS output drives only a CMOS input and as a CMOS gate has an extremely high input resistance, the current drawn is almost zero and, therefore, the output voltage levels will be very close to zero for LOW state and  $V_{DD}$  for HIGH state, i.e.  $V_{OL}(\text{max}) = 0 \text{ V}$ ,  $V_{OH}(\text{min}) = V_{DD}$ . Usually, the input voltage levels are expressed as percentage of  $V_{DD}$  values, for example,  $V_{IL}(\text{max}) = 30\% \text{ of } V_{DD}$ ,  $V_{IH}(\text{min}) = 70\% \text{ of } V_{DD}$ .

**Power dissipation:** When a CMOS circuit is in a static state, its power dissipation per gate is extremely small, but it increases with increase in operating frequency and supply voltage level. For dc, CMOS power dissipation is only 2.5 nW per gate when  $V_{DD} = 5 \text{ V}$ , and it increases to 10 nW per gate when  $V_{DD} = 10 \text{ volts}$ . With a  $V_{DD}$  of 10 V at a frequency of 100 KPPS, power dissipation is 0.1 mW/gate, and at 1 MHz,  $P_D = 1 \text{ mW}$ .

**Noise margins:** Since for a CMOS gate,  $V_{OL}(\text{max}) = 0 \text{ V}$ ,  $V_{OH}(\text{min}) = V_{DD}$  and  $V_{IL}(\text{max})$  is 30% of  $V_{DD}$  and  $V_{IH}(\text{min})$  is 70% of  $V_{DD}$ , the low level and high level noise margins will be the same (30% of  $V_{DD}$ ) and increase with an increase in the value of  $V_{DD}$ . Of course, the higher values of  $V_{DD}$  result in higher power dissipations.

$$V_{NH} = V_{OH}(\text{min}) - V_{IH}(\text{min}) = V_{DD} - 70\% \text{ of } V_{DD} = 30\% \text{ of } V_{DD}$$

$$V_{NL} = V_{IL}(\text{max}) - V_{OL}(\text{max}) = 30\% \text{ of } V_{DD} - 0 \text{ V} = 30\% \text{ of } V_{DD}$$

**Fan-out:** The CMOS fan-out depends on the permissible maximum propagation delay. For low frequencies ( $\leq 1$  MHz), the fan-out is 50, and for high frequencies it will be less.

**Switching speed:** The speed of the CMOS gate increases with increase in  $V_{DD}$ . The 4000 series has  $t_{pd} = 50$  ns at  $V_{DD} = 5$  V and  $t_{pd} = 25$  ns at  $V_{DD} = 10$  V. The increase in  $V_{DD}$  results in increase in power dissipation too.

**Unused inputs:** The CMOS inputs should never be left disconnected. All CMOS inputs have to be tied either to a fixed voltage level (0 V or  $V_{DD}$ ) or to another input.

**Static charge susceptibility:** The high input resistance of CMOS inputs makes CMOS gates prone to static charge build-up, that can produce voltages large enough to break down the dielectric insulation between the MOSFET gate and the channel. Most of the newer CMOS devices are protected against static charge damage by the inclusion of protective zener diodes on each input.

## 16.11 DYNAMIC MOS LOGIC

When power consumption and physical size are the prime design considerations as in digital watches and calculators, dynamic MOS logic is usually the family selected to meet these requirements. Each transistor used in a dynamic MOS circuit is identical to the other, and each can be fabricated in a very small amount of space on a chip. Consequently, large and very large scale integrations are possible.

In dynamic MOS logic, power consumption is minimized by relying on the inherent capacitance of the MOS transistors to store logic levels, i.e. to remain charged or discharged—and by using clock signals to turn on transistors for very brief intervals of time only. The clock signals turn transistors on to allow the capacitance to recharge or discharge at periodic intervals. Since a transistor is OFF during most of any given time interval, the average power consumption is quite small.

The NMOS transmission gate shown in Figure 16.28a is a fundamental component of dynamic logic circuits. Because the NMOSFET is completely symmetrical, the drain and source terminals are indistinguishable, i.e. current can flow in either direction. In dynamic logic applications, there is a shunt capacitance at each of these terminals identified in the figure as  $C_1$  and  $C_2$ .

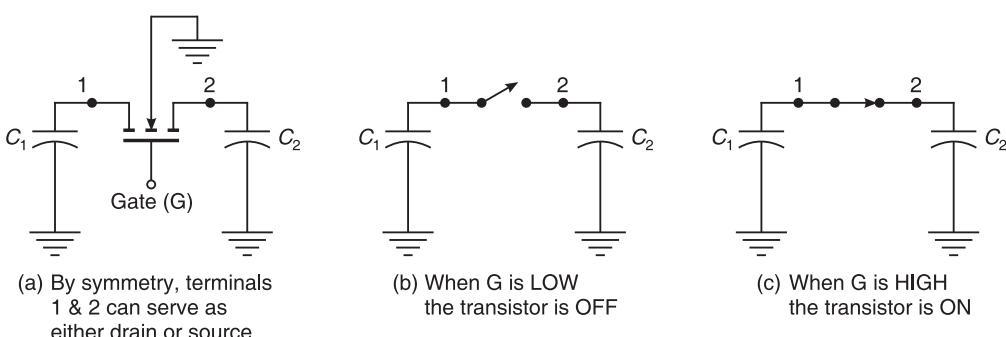


Figure 16.28 NMOS transmission gate.

When the gate terminal G is LOW, the transistor will be OFF irrespective of the potentials at drain and source, i.e. irrespective of charges on  $C_1$  and  $C_2$ , because the gate-to-source voltage may

be either 0 or negative. Once the transistor is OFF, it acts as an open switch as shown in Figure 16.28b and the charges on capacitors remain as they are. That is, no transfer of charge takes place, and therefore, no signal transmission takes place.

When G is HIGH, the transistor is ON and acts as a closed switch as shown in Figure 16.28c. If capacitors  $C_1$  and  $C_2$  are charged to the same level, no transfer of charge takes place. But if one capacitor is charged and the other discharged, transfer of charge takes place from one capacitor to the other, i.e. the input is transmitted to the output.

### 16.11.1 Dynamic MOS Inverter

Figure 16.29 shows a dynamic MOS inverter. The capacitance shown by dotted lines represents the inherent device (interelectrode) capacitance. The  $\phi_1$  and  $\phi_2$  are the control signals that are used to control the ON and OFF of  $Q_2$  and  $Q_3$ . The two together are called a two-phase non-overlapping clock, because  $\phi_1$  and  $\phi_2$  are never both HIGH at the same time. As in the case of a normal MOS inverter,  $Q_1$  acts as a switching transistor and  $Q_2$  as a load resistor. The only difference is that, in this case  $Q_2$  acts as an active load only when clock  $\phi_1$  is HIGH. The rest of the time (i.e. when  $\phi_1$  is LOW),  $Q_2$  is OFF and does not allow any current to pass through it. The transistor  $Q_3$  acts as a transmission gate, i.e. it transfers charge only when clock  $\phi_2$  is HIGH. The rest of the time (i.e. when  $\phi_2$  is LOW), no transfer of charge takes place.

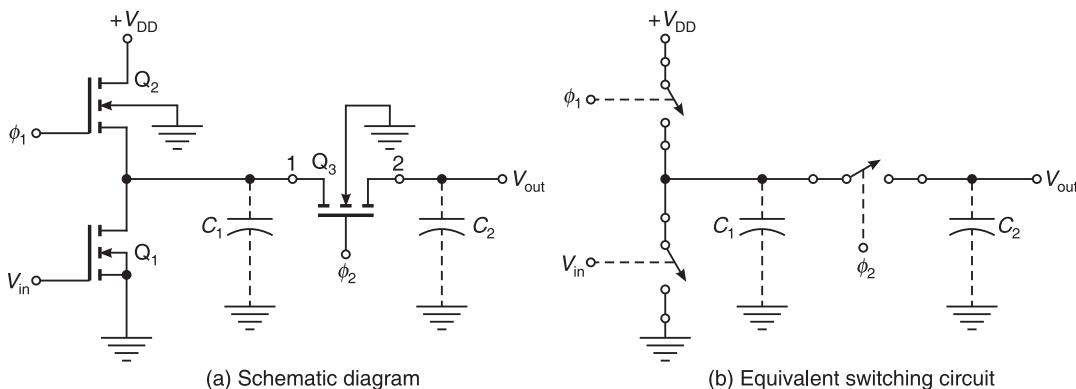


Figure 16.29 Dynamic MOS inverter.

When  $V_{in}$  is LOW,  $Q_1$  is OFF. When  $\phi_1$  goes HIGH,  $Q_2$  conducts and  $C_1$  is charged, but when  $\phi_1$  goes LOW, there is no path for  $C_1$  to discharge, and so,  $C_1$  remains charged. When  $\phi_2$  goes HIGH, this charge on  $C_1$  is transferred to  $C_2$ , and so,  $V_{out}$  goes HIGH. Thus, a LOW at the input results in a HIGH at the output.

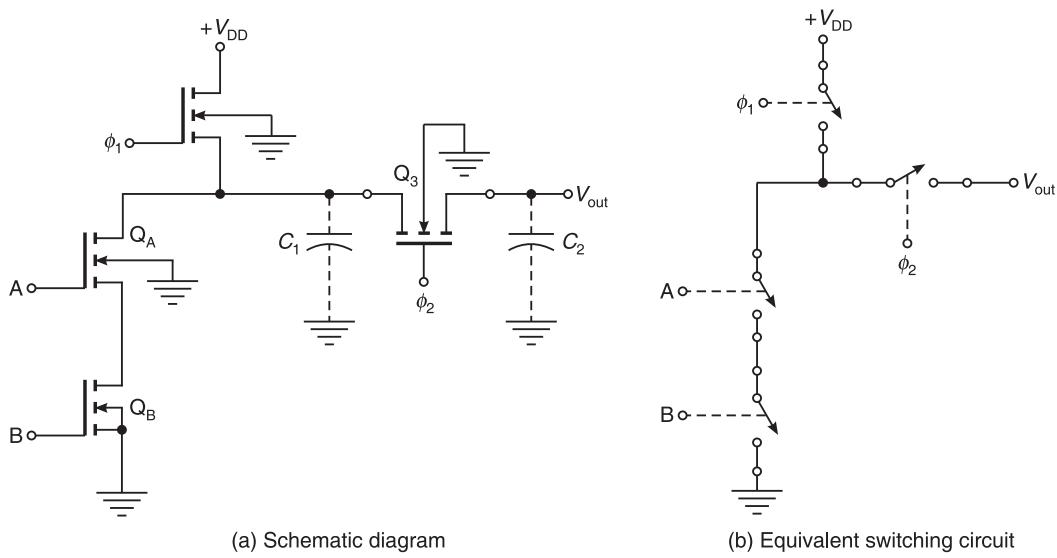
Suppose  $V_{in}$  is HIGH, when  $\phi_1$  goes HIGH,  $Q_2$  conducts and  $Q_1$  also turns on. So,  $C_1$  cannot charge. When  $\phi_2$  goes HIGH,  $Q_3$  acts as a closed switch and  $C_2$  discharges into  $C_1$ . So,  $V_{out}$  goes LOW.  $V_{out}$  remains LOW when  $\phi_2$  is LOW. Thus, a HIGH at the input results in a LOW at the output. Therefore, the above circuit acts as an inverter.

The output of a dynamic logic gate is ‘valid’ only when  $\phi_2$  is HIGH. Thus, we can say that the gates are sampled at the frequency of  $\phi_2$ . A sampled output becomes the input to other gates, whose responses become available only at the next sampling time. The disadvantage of dynamic

logic is the complexity added by the clocking requirements. The capacitors need to be recharged periodically so that the charge on the capacitors does not decay very much. This process of recharging is called *refreshing*. The minimum clock frequency is, therefore, determined by the amount of time taken by the capacitance to decay significantly. A typical period is 1 ms, giving a minimum clock frequency of 1 kHz.

### 16.11.2 Dynamic NAND Gate

Figure 16.30 shows a dynamic two-input MOS NAND gate and its equivalent switching circuit. The only difference between this and the static NMOS NAND gate is that, the load MOSFET is clocked by  $\phi_1$  and a transmission gate is added at the output and the outputs are clocked through the transmission gate by  $\phi_2$ .



**Figure 16.30** Dynamic MOS NAND gate.

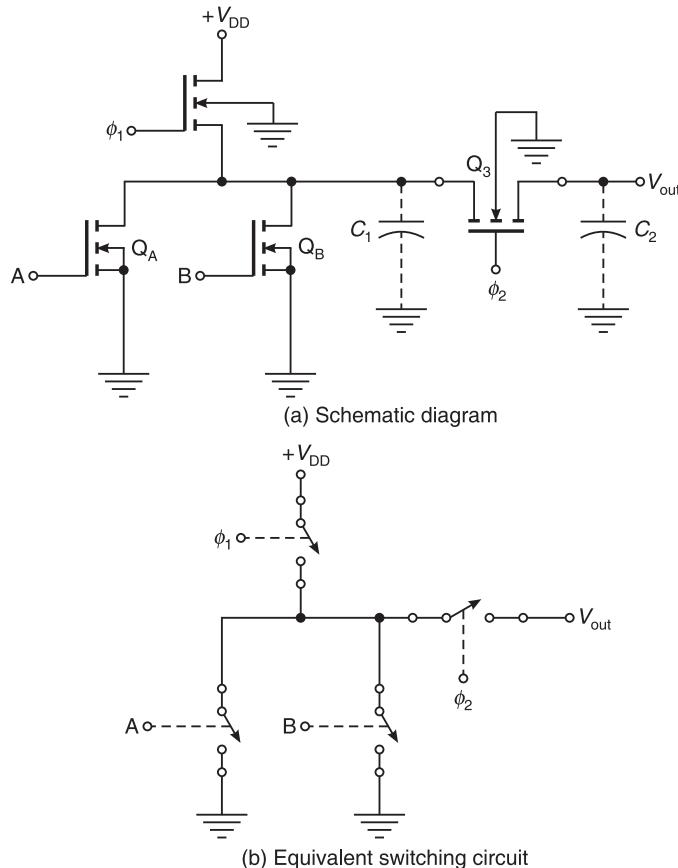
When  $\phi_1$  goes HIGH,  $C_1$  is charged according to the NAND logic of inputs A and B and when  $\phi_2$  goes HIGH, this charge is transferred from  $C_1$  to  $C_2$ . So, the output  $V_{out}$  follows the NAND logic. When either A is LOW, or B is LOW, or both A and B are LOW, the corresponding MOSFETs ( $Q_A$  and  $Q_B$ ) will be OFF and no current passes through them. Thus,  $C_1$  is charged when  $\phi_1$  goes HIGH, and this charge on  $C_1$  is transferred to  $C_2$ , when  $\phi_2$  goes HIGH. Therefore, the output goes HIGH ( $C_1$  remains discharged after  $\phi_2$  goes LOW).

Only when both A and B are HIGH,  $Q_A$  and  $Q_B$  will turn on when  $\phi_1$  goes HIGH and, therefore, no current flows through  $C_1$  and it does not charge and remains in the discharged condition only. When  $\phi_2$  goes HIGH,  $C_2$  discharges into  $C_1$ , and so the output goes LOW. Hence, this circuit works as a two-input NAND gate.

### 16.11.3 Dynamic NOR Gate

Figure 16.31 shows a dynamic two-input MOS NOR gate and its equivalent switching circuit. The only difference between this and the static NMOS NOR gate is that, the load MOSFET is clocked

by  $\phi_1$  and a transmission gate is added at the output and the outputs are clocked through the transmission gate by  $\phi_2$ .



**Figure 16.31** Dynamic MOS NOR gate.

When  $\phi_1$  goes HIGH,  $C_1$  is charged according to the NOR logic of inputs A and B, and when  $\phi_2$  goes HIGH, this charge is transferred from  $C_1$  to  $C_2$ . So, the output  $V_{\text{out}}$  follows the NOR logic.

When both A and B are LOW,  $Q_A$  and  $Q_B$  will be OFF. So,  $C_1$  charges when  $\phi_1$  goes HIGH and this charge on  $C_1$  is transferred to  $C_2$  when  $\phi_2$  goes HIGH, and so, the output goes HIGH. ( $C_1$  remains discharged, after  $\phi_2$  goes LOW).

When either A is HIGH or B is HIGH or both A and B are HIGH, either  $Q_A$  or  $Q_B$  or both  $Q_A$  and  $Q_B$  will turn on when  $\phi_1$  goes HIGH, keeping  $C_1$  in the discharged condition only. When  $\phi_2$  goes HIGH, the charge on  $C_2$  is transferred to  $C_1$ , and so, the output goes LOW. Hence, this circuit works as a two-input NOR gate.

## 16.12 INTERFACING

Interfacing means connecting the output(s) of one circuit or system to the input(s) of another system with different electrical characteristics.

There are a number of logic families, each having its own strong points. In designing more complex digital systems, the designers utilize different logic families for different parts of the system in order to take advantage of the strong points of each family. When the designed parts are assembled, since the electrical characteristics of different logic families vary widely, interfacing circuits or logic level translators are used to connect the driver circuit belonging to one family to the load circuit belonging to another family.

### 16.12.1 TTL to ECL

The TTL is the most widely used logic family, but its speed of operation is not very high. The ECL is the fastest family. In some applications, the rate at which input data is to be handled may be much lower than the rate at which the output data is to be handled. Therefore, it becomes necessary to interconnect the two different logic systems, such as TTL and ECL. One such application is in the time division multiplexing of M digital signals to form a single digital signal. Although, the bit rate of each of the M signals may be handled using TTL, the bit rate of the composite signal is M times faster and may require ECL to process it.

### 16.12.2 ECL to TTL

Sometimes, the input data is at a faster rate, but the output data is at a slower rate like in demultiplexers. An ECL to TTL logic translator will be of use in such cases.

### 16.12.3 TTL to CMOS

The MOS and CMOS gates are slower than the TTL gates, but consume less space. Hence, there is an advantage in using TTL and MOS devices in combination.

The input current values of CMOS are extremely low compared with the output current capabilities of any TTL series. Thus, TTL has no problem in meeting the CMOS input current requirements. So, a level translator is used to raise the level of the output voltage of the TTL gate to an acceptable level for CMOS. The arrangement is shown in Figure 16.32a, where the TTL output is connected to a +5 V source with a pull-up resistor. The presence of the pull-up resistor will cause the TTL output to rise to approximately +5 V in the HIGH state, thereby providing an adequate CMOS input.

If the TTL has to drive a high voltage CMOS, the pull-up resistor cannot be used to raise the level of the TTL output to the level of the CMOS input, since the TTL is sensitive to voltage levels. In such a case, an open collector buffer can be used to interface TTL to a high voltage CMOS as shown in Figure 16.32b.

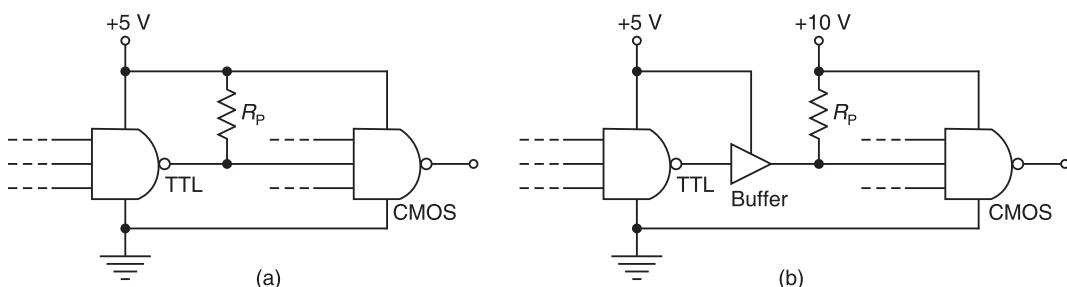
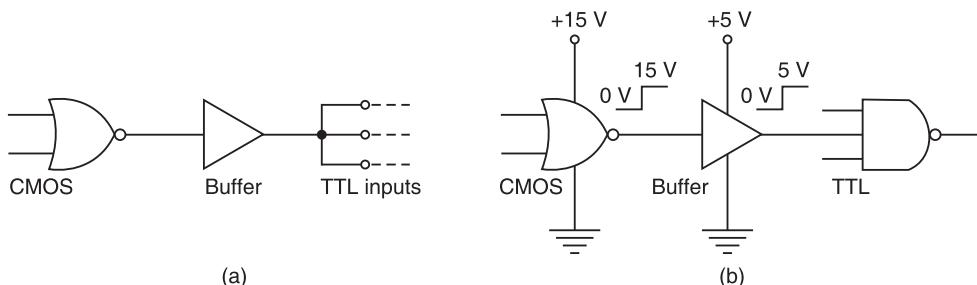


Figure 16.32 TTL to CMOS interfacing.

#### 16.12.4 CMOS to TTL

The CMOS output can supply enough voltage and current to satisfy the TTL input requirements in the HIGH state. Hence, no special consideration is required for the HIGH state. But the TTL input current requirements at LOW state cannot be met directly. Therefore, an interface circuit with a LOW input current requirement and a sufficiently high output current rating is required. A CMOS buffer serves this purpose. The arrangement is shown in Figure 16.33a.

When a high voltage CMOS has to drive a TTL gate, a voltage level translator that converts the high voltage input to a +5 V output is used between CMOS and TTL as shown in Figure 16.33b.



**Figure 16.33** CMOS to TTL interfacing.

### SHORT QUESTIONS AND ANSWERS

1. What are the merits of ICs?  
A. The merits of ICs are: Use of ICs
  - (a) Reduces the overall size of the digital system drastically,
  - (b) Reduces the cost of the digital system,
  - (c) Improves the reliability of the system by reducing the number of external connections from one device to another, and
  - (d) Greatly reduces the power consumption of digital systems.
2. What are the limitations of ICs?  
A. The limitations of ICs are:
  - (a) ICs cannot handle large voltages or currents.
  - (b) Electrical devices like precision resistors, inductors, transformers and large capacitors cannot be implemented on chips.
  - (c) They are mainly suitable for low power applications only.
3. Using which technologies are ICs fabricated?  
A. Presently ICs are fabricated using TTL, ECL, IIL, MOS and CMOS technologies.
4. Name the technologies which use bipolar transistors.  
A. TTL, ECL and IIL technologies use bipolar transistors.
5. Name the technologies which use unipolar transistors.  
A. MOS and CMOS technologies use unipolar transistors.

**6.** Define the terms:

- |                          |                       |
|--------------------------|-----------------------|
| (a) threshold voltage    | (b) propagation delay |
| (c) power dissipation    | (d) fan-in            |
| (e) fan-out              | (f) noise margin      |
| (g) speed power product. |                       |

- A. (a) The threshold voltage is defined as that voltage at the input of a gate which causes a change in the state of the output from one logic level to the other.  
(b) The propagation delay of a gate is defined as the time taken by the pulse to propagate from input to output.  
(c) The power dissipation of a gate is defined as the power required by the gate to operate with 50% duty cycle at a specified frequency.  
(d) The fan-in of a logic gate is defined as the number of inputs that the gate is designed to handle.  
(e) The fan-out of a logic gate is defined as the maximum number of similar gates that the output of the gate can drive without impairing its normal operation.  
(f) The noise margin is defined as the maximum noise signal that can be added to the input signal of a digital circuit without causing an undesirable change in the circuit output.  
(g) The speed power product of a logic gate is defined as the product of the gate propagation delay and the gate power dissipation.

**7.** Which technologies are obsolete?

- A. The RTL, DCTL, DTL and HTL technologies are obsolete.

**8.** Define standard load.

- A. A standard load is defined as the amount of current needed by an input of another gate of the same logic family.

**9.** Which is the most popular and most widely used digital IC family?

- A. TTL is the most popular and most widely used digital IC family.

**10.** Name the three types of TTL gates.

- A. The three types of TTL gates are:  
(a) totem-pole type,  
(b) open-collector type, and  
(c) tri-state type.

**11.** What are the three possible output states of a tri-state IC?

- A. The three possible output states of a tri-state IC are: LOW, HIGH and HIGH impedance state.

**12.** When does a TTL circuit act as a current sink and source?

- A. A TTL circuit acts as a current sink in the low state and as a current source in the high-state.

**13.** What do you mean by Schottky TTL? Why is it faster than standard TTL?

- A. Schottky TTL is one which uses transistors having a Schottky barrier diode (SBD) between the base and the collector of each transistor. It is more than three times faster than standard TTL because in this the transistors are not allowed to go fully into saturation and also it uses smaller resistors.

**14.** Which TTL series is most suitable at high frequencies?

- A. F (fast) TTL, 74F series is the most suitable at high frequencies.

**15.** Which gates are suitable for the wired AND operation?

- A. TTL open collector gates are suitable for the wired AND operation.

**16.** What are the merits and demerits of TTL?

- A. Good speed, low manufacturing cost, wide range of circuits and the availability in SSI and MSI are the merits of TTL. Tight  $V_{cc}$  tolerance, relatively high power consumption, moderate packing density, generation of noise spikes and susceptibility to power transients are the demerits of TTL.

**17.** How many TTL subfamilies are there? Name them.

- A. There are eight TTL subfamilies. They are:

- (a) standard TTL,
- (b) high speed TTL,
- (c) low power TTL,
- (d) Schottky TTL,
- (e) low power Schottky TTL,
- (f) advanced Schottky TTL,
- (g) advanced low power Schottky TTL, and (h) F(fast) TTL.

**18.** What are the advantages and disadvantages of totem-pole configuration?

- A. The totem-pole configuration has the advantages of high speed and low power dissipation, but the disadvantages of generation of current spikes and the inability to be wire ANDed.

**19.** Which is the non-saturated logic?

- A. ECL is the non-saturated logic.

**20.** Which logic gives complementary outputs?

- A. Emitter coupled logic (ECL) gives complementary outputs.

**21.** Which logic is preferred in superfast computers?

- A. Emitter coupled logic (ECL) is preferred in superfast computers.

**22.** Which logic gates are suitable for wired OR operation?

- A. ECL open emitter gates are suitable for wired OR operation.

**23.** What are the drawbacks of ECL?

- A. The drawbacks of ECL are:

- (a) high cost,
- (b) low noise margin,
- (c) high power dissipation,
- (d) its negative supply voltage and logic levels are not compatible with other logic families, and
- (e) Problem of cooling.

**24.** What are the merits of ECL?

- A. The merits of ECL are:

- (a) The speed of operation is very high.
- (b) The current drawn from the supply is steadier and they do not experience large switching transients.

**25.** What are the characteristics of ECL gates?

- A. The important characteristics of ECL gates are:

- (a) Transistors never saturate, so speed is high with  $t_{pd} = 1$  ns.
- (b) Logic levels are negative,  $-0.9$  V for logic 1 and  $-1.7$  V for logic 0.
- (c) Noise margin is less, about 250 mV. This makes ECL unreliable for use in heavy industrial environments.
- (d) ECL circuits produce the output and its complement and therefore eliminate the need for invertors.
- (e) Fan-out is large because the output impedance is low. It is about 25.
- (f) Power dissipation per gate is large,  $P_D = 40$  mW.
- (g) The total current flow in ECL is more or less constant. No noise spikes will be internally generated.

## 904 FUNDAMENTALS OF DIGITAL CIRCUITS

26. Why does the MOS family mostly use NMOS devices?
  - A. The MOS family mostly uses NMOS devices because they operate at about three times the speed of their PMOS counterparts, and also have twice the packing density of PMOS.
27. Why are MOS ICs especially sensitive to static charge?
  - A. MOS ICs are sensitive to static charge because of the very high impedance at the MOSFET's input.
28. What are the advantages of MOS families over bipolar families?
  - A. Compared to the bipolar families, the MOS families are simpler and inexpensive to fabricate, require much less power, have a better noise margin, a greater supply voltage range, a higher fan-out and require much less chip area.
29. What are the disadvantages of MOS families compared to bipolar families?
  - A. Compared to bipolar families the MOS families are slower in operating speed and are susceptible to static charge damage.
30. What are the parameters of MOS?
  - A. For MOS logic:  $t_{pd} = 50$  ns, NM = 1.5 V (for +5 V supply),  $P_D = 0.1$  mW, and fan-out = 50 for frequencies greater than 100 Hz and virtually unlimited for dc or low frequencies.
31. What are the two types of MOSFETs and which type is used in MOS ICs?
  - A. The two types of MOSFETs are:
    - (a) depletion type, and
    - (b) enhancement type. The MOS digital ICs use enhancement MOSFETs exclusively.
32. What are the merits and demerits of MOS logic compared to TTL?
  - A. The MOS logic is the simplest to fabricate and has high packing density and low power dissipation per gate, but is more susceptible to static charge damage and it is slow compared to TTL.
33. Where are MOS ICs used?
  - A. MOS ICs are ideally suited for LSI, VLSI and ULSI for dedicated applications such as large memories, calculator chips, large microprocessors, etc. The operating speed of MOS is slower than that of TTL, so they are hardly used in SSI and MSI applications.
34. What are the merits and demerits of CMOS?
  - A. The demerits and merits of CMOS are: the CMOS family has the greatest complexity and lowest packing density, but it possesses the important advantages of higher speed and much lower dissipation. It can be operated at high voltages resulting in improved noise immunity.
35. Where is CMOS technology used?
  - A. The CMOS technology is used to construct small, medium, and large scale ICs for a wide variety of applications ranging from general purpose logic to microprocessors.
36. Which ICs are used in watches and calculators? And why?
  - A. CMOS ICs are used in watches and calculators because of their extremely low power consumption.
37. Why is the fan-out of CMOS very high?
  - A. The CMOS has very high input resistance. Thus it draws almost zero current from the driving gate, and therefore its fan-out is very high.
38. How do you compare CMOS with TTL?
  - A. The CMOS fabrication process is simpler than that of TTL and it has greater packing density. The CMOS uses only a fraction of the power needed even for low power TTL. The CMOS is however generally slower than TTL.
39. For which applications is CMOS ideally suited?
  - A. The CMOS is ideally suited for applications requiring battery power or battery backup power.

- 40.** Which is the fastest logic family? And the slowest family?  
A. ECL is the fastest logic family and MOS is the slowest logic family.
- 41.** Which family has the highest packing density? And the lowest packing density?  
A. IIL has the highest packing density and ECL has the lowest packing density.
- 42.** Which logic family consumes the maximum power? And the least power?  
A. ECL family consumes the maximum power and CMOS family consumes the least power.
- 43.** Which logic family is the simplest to fabricate? And most complex to fabricate?  
A. The MOS logic family is the simplest to fabricate and the TTL family is the most complex to fabricate.
- 44.** Which logic family has the highest fan-out? And the least fan-out?  
A. The CMOS family has the highest fan-out and the IIL family has the least fan-out.
- 45.** Which logic family has the highest noise margin? And the least noise margin?  
A. The CMOS family has the highest noise margin and the ECL family has the least noise margin.
- 46.** What are level shifters?  
A. Level shifters are specially designed ICs which are used to make devices from different logic families compatible with each other.
- 47.** Which CMOS series is compatible pin for pin with TTL?  
A. The CMOS 74C series is compatible pin for pin and function for function with TTL devices having the same number.
- 48.** What do you mean by interfacing? Why is it required?  
A. Interfacing means connecting the output(s) of one circuit or system to the input(s) of another system with different electrical characteristics.  
There are a number of logic families, each having its own strong points. In designing more complex digital systems, the designers utilize different logic families for different parts of the system in order to take advantage of the strong points of each family. When the designed parts are assembled, since the electrical characteristics of different logic families vary widely, interfacing circuits or logic level translators are used to connect the driver circuit belonging to one family to the load circuit belonging to another family.
- 49.** What is a transmission gate?  
A. A transmission gate is simply a digitally controlled CMOS switch. It is a bilateral device.
- 50.** Why is CMOS gate called a bilateral transmission gate?  
A. Since the CMOS gate can transmit signals in both directions, it is called a bilateral transmission gate.
- 51.** Which logic family is suitable for SSI and MSI? For LSI and VLSI? And for VLSI and ULSI?  
A. TTL is the most suitable for SSI and MSI. CMOS can also be used for SSI and MSI. MOS is more suitable for LSI and VLSI. IIL and MOS are suitable for VLSI and ULSI.
- 52.** When is dynamic MOS logic selected?  
A. When physical size and power consumption are the prime design considerations as in digital watches and calculators, dynamic MOS logic is selected.
- 53.** What is the disadvantage of dynamic logic?  
A. The disadvantage of dynamic logic is the complexity added by the clocking requirements. The capacitors need to be recharged periodically so that the charge on the capacitors does not decay very much.

**54.** What do you mean by refreshing?

A. The process of recharging capacitors in dynamic logic is called refreshing.

**55.** Why is power consumption quite low in dynamic MOS logic?

A. The power consumption is quite low in dynamic MOS logic because the transistors are off during most of any given time interval.

### REVIEW QUESTIONS

1. With the help of a neat diagram, explain the working of a two-input TTL NAND gate.
2. With the help of a neat diagram, explain the working of a two-input ECL OR/NOR gate.
3. With the help of a neat diagram, explain the working of IIL NAND and NOR gates.
4. Show that in a totem pole TTL NAND gate transistors  $Q_3$  and  $Q_4$  cannot conduct simultaneously.
5. With the help of a neat circuit diagram, explain the working of (a) a MOS inverter, (b) a two-input MOS NAND gate, and (c) a two-input MOS NOR gate.
6. With the help of a neat circuit diagram, explain the working of (a) a CMOS inverter, (b) a two-input CMOS NAND gate, and (c) a two-input CMOS NOR gate.
7. With the help of the neat circuit diagram explain the working of a transmission gate.
8. Write short notes on dynamic logic.
9. With the help of circuit diagrams explain the working of (a) dynamic MOS inverter, (b) dynamic NAND gate, and (c) dynamic NOR gate.
10. Write short notes on interfacing of various logic families.
11. Compare different logic families.
12. What are the merits and demerits of various logic families?

### FILL IN THE BLANKS

1. The IC technologies which use bipolar transistors are \_\_\_\_\_, \_\_\_\_\_, and \_\_\_\_\_.
2. The IC technologies which use unipolar transistors are \_\_\_\_\_ and \_\_\_\_\_.
3. The \_\_\_\_\_ voltage is defined as that voltage at the input of a gate which causes a change in the state of the output from one logic level to the other.
4. The \_\_\_\_\_ of a gate is defined as the time taken by the pulse to propagate from input to output.
5. The \_\_\_\_\_ of a gate is defined as the power required by the gate to operate with 50% duty cycle at a specified frequency.
6. The \_\_\_\_\_ of a logic gate is defined as the number of inputs that the gate is designed to handle.
7. The \_\_\_\_\_ of a logic gate is defined as the maximum number of similar gates that the output of the gate can drive without impairing its normal operation.
8. The \_\_\_\_\_ is defined as the maximum noise signal that can be added to the input signal of a digital circuit without causing an undesirable change in the circuit output.
9. The \_\_\_\_\_ of a logic gate is defined as the product of the gate propagation delay and the gate power dissipation.

10. A \_\_\_\_\_ is defined as the amount of current needed by an input of another gate of the same logic family.
11. \_\_\_\_\_ is the most popular and most widely used digital IC family.
12. \_\_\_\_\_ is the fastest of the saturated logic families.
13. The \_\_\_\_\_, \_\_\_\_\_, and \_\_\_\_\_ are the three types of TTL gates.
14. The three possible output states of a tri-state TTL are \_\_\_\_\_, \_\_\_\_\_, and \_\_\_\_\_.
15. \_\_\_\_\_ series is the most suitable for high frequencies.
16. \_\_\_\_\_ is the fastest logic family.
17. The \_\_\_\_\_ family has got both the logic levels negative.
18. \_\_\_\_\_ is a non-saturated logic.
19. The MOS family mostly uses \_\_\_\_\_ devices.
20. The two types of MOSFETs are (a) \_\_\_\_\_ and (b) \_\_\_\_\_.
21. The MOS digital ICs use \_\_\_\_\_ MOSFETs exclusively.
22. MOS ICs are ideally suited for \_\_\_\_\_, \_\_\_\_\_, and \_\_\_\_\_.
23. The \_\_\_\_\_ technology is used to construct small, medium and large scale ICs for a wide variety of applications.
24. \_\_\_\_\_ ICs are used in watches and calculators.
25. \_\_\_\_\_ is ideally suited for applications involving battery power and battery backup power.
26. The \_\_\_\_\_ family consumes maximum power and the \_\_\_\_\_ family consumes the least power.
27. The \_\_\_\_\_ family has the highest fan-out and the \_\_\_\_\_ family has the least fan-out.
28. The \_\_\_\_\_ family has the highest noise margin and the \_\_\_\_\_ family has the least.
29. \_\_\_\_\_ gates are suitable for wired AND operation.
30. \_\_\_\_\_ gates are suitable for wired OR operation.
31. The advantages of totem-pole configuration are \_\_\_\_\_ and \_\_\_\_\_.
32. \_\_\_\_\_ is most suitable for SSI and MSI, and \_\_\_\_\_ can also be used for SSI and MSI.
33. \_\_\_\_\_ is more suitable for LSI and VLSI.
34. \_\_\_\_\_ and \_\_\_\_\_ are suitable for VLSI and ULSI.
35. A \_\_\_\_\_ gate is simply a digitally controlled CMOS switch.
36. The \_\_\_\_\_ logic is the simplest to fabricate and occupies very little space.
37. \_\_\_\_\_ ICs are hardly used in SSI and MSI applications because of their slower speed.
38. The recharging of capacitors in dynamic MOS logic is called \_\_\_\_\_.
39. The \_\_\_\_\_ gate is called a bilateral transmission gate.
40. The interfacing circuits are also called logic \_\_\_\_\_.
41. \_\_\_\_\_ is called the figure of merit of an IC family.
42. \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_, and \_\_\_\_\_ logic families are now obsolete.
43. There are \_\_\_\_\_ TTL subfamilies.
44. \_\_\_\_\_ outputs cannot be wired ANDed whereas \_\_\_\_\_ outputs can be.
45. Low power TTL uses \_\_\_\_\_ resistors whereas high speed TTL uses \_\_\_\_\_ resistors.
46. When power consumption and physical size are prime considerations \_\_\_\_\_ logic is preferred.
47. \_\_\_\_\_ are specially designed ICs which are used to make devices from different logic families compatible with each other.
48. \_\_\_\_\_ logic is preferred in superfast computers.

49. \_\_\_\_\_ CMOS series is compatible pin for pin and function for function with TTL devices having the same number.

50. \_\_\_\_\_ is the newest of the logic families.

51. The \_\_\_\_\_ logic family is preferred in noisy environments.

52. The \_\_\_\_\_ of CMOS gate increases with increase in  $V_{DD}$ .

## **OBJECTIVE TYPE QUESTIONS**

- 16.** The logic family with both logic levels negative is  
(a) TTL                    (b) ECL                    (c) CMOS                    (d) MOS
- 17.** The logic family which consumes least power is  
(a) TTL                    (b) ECL                    (c) MOS                    (d) CMOS
- 18.** The number of subfamilies TTL has is  
(a) 4                      (b) 8                      (c) 6                      (d) 10
- 19.** The logic family with highest packing density is  
(a) TTL                    (b) IIL                    (c) CMOS                    (d) MOS
- 20.** The logic family which is simplest to fabricate is  
(a) TTL                    (b) ECL                    (c) MOS                    (d) CMOS
- 21.** The logic family ideally suited for LSI/VLSI/ULSI applications is  
(a) TTL                    (b) ECL                    (c) MOS                    (d) CMOS
- 22.** The newest of the logic families is the  
(a) TTL                    (b) ECL                    (c) IIL                    (d) CMOS

# 17

## ANALOG-TO-DIGITAL AND DIGITAL-TO-ANALOG CONVERTERS

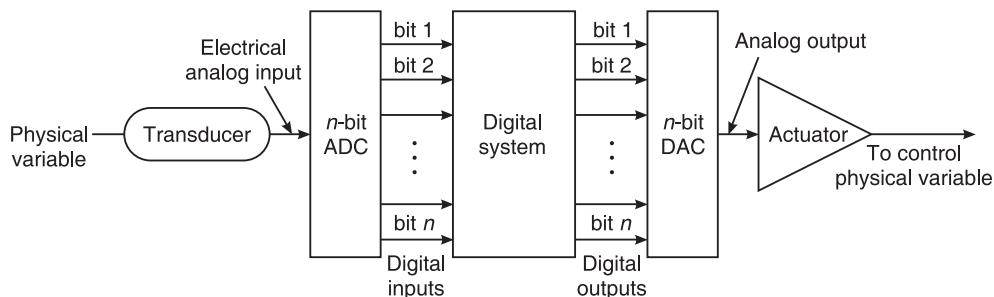
### 17.1 INTRODUCTION

An analog quantity is one that can take on any value over a continuous range of values. It represents an exact value. Most physical variables are analog in nature. Temperature, pressure, light and sound intensity, position, rotation, speed, etc. are some examples of analog quantities.

A digital quantity takes on only discrete values. The value is expressed in a digital code such as a binary or BCD number.

When a physical process is monitored or controlled by a digital system such as a digital computer, the physical variables are first converted into electrical signals using transducers, and then these electrical analog signals are converted into digital signals using analog-to-digital converters (ADCs). These digital signals are processed by a digital computer and the output of the digital computer is converted into analog signals using digital-to-analog converters (DACs). The output of the DAC is modified by an actuator and the output of the actuator is applied as the control variable.

Figure 17.1 shows how ADCs and DACs function as interfaces between a completely digital system such as a digital computer and the analog world. This function has become increasingly more important as inexpensive microcomputers are being widely used for process control.



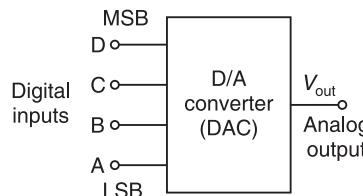
**Figure 17.1** Interfacing a digital computer to the analog world.

## 17.2 DIGITAL-TO-ANALOG (D/A) CONVERSION

Basically, D/A conversion is the process of converting a value represented in digital code, such as straight binary or BCD, into a voltage or current which is proportional to the digital value. Figure 17.2 shows the symbol for a typical 4-bit D/A converter. Each of the digital inputs A, B, C, and D can assume a value 0 or a 1, therefore, there are  $2^4 = 16$  possible combinations of inputs. For each input number 0000, 0001, ..., 1111, the D/A converter outputs a unique value of voltage. The analog output voltage  $V_{\text{out}}$  is proportional to the input binary number, that is,

$$\text{Analog output} = K \times \text{digital input}$$

where  $K$  is the proportionality factor and is a constant value for a given DAC. The analog output can, of course, be current or voltage.

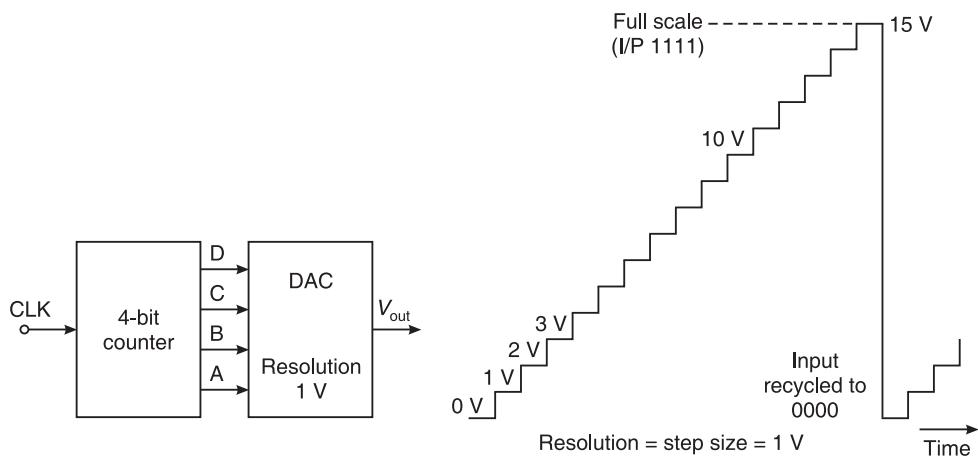


**Figure 17.2** Block diagram of a 4-bit DAC.

Strictly speaking, the output of a DAC is not a true analog quantity, because it can take on only specific values. In that sense, it is actually digital. Thus, the output of a DAC is a 'pseudo-analog' quantity. By increasing the number of input bits, the number of possible output values can be increased and also the step size (the difference between two successive output values) can be reduced, thereby producing an output that is more like an analog quantity. Figure 17.3 shows the output waveform of a DAC when it is fed by a 4-bit binary counter.

When the binary counter is continually recycled through its 16 states by applying the clock signal, the DAC output will be a staircase waveform with a step size of 1 V. When the counter is at 0000, the output of the DAC is minimum (0 V). When the counter is at 1111, the output of the DAC is maximum (15 V). This is the full-scale output.

Digital-to-analog and analog-to-digital conversions form the very important aspects of digital data processing. Digital-to-analog conversion is a straightforward process and is considerably easier than the A/D conversion. In fact, a DAC is usually an integral part of any ADC.



**Figure 17.3** Output waveform of a DAC fed by a binary counter.

### 17.2.1 Parameters of DAC

**Resolution (step size).** The resolution of a DAC is defined as the smallest change that can occur in an analog output as a result of a change in the digital input. The resolution of a DAC is also defined as the reciprocal of the number of discrete steps in the full-scale output of the DAC. The resolution is always equal to the weight of the LSB and is also referred to as the *step size*. The resolution or step size is the size of the jumps in the staircase waveform. The step size is the amount by which  $V_{out}$  will change as the digital input value is changed from one value to the next. For the waveform shown in Figure 17.3, the resolution or step size = 1 V. The step size of the DAC is the same as the proportionality factor in the DAC input-output relationship.

Although resolution can be expressed as the amount of voltage or current per step, it is also useful to express it as a percentage of the full-scale output as

$$\% \text{ resolution} = \frac{\text{step size}}{\text{full scale}} \times 100\%$$

Since, full-scale = number of steps  $\times$  step size, resolution can be expressed as

$$\% \text{ resolution} = \frac{1}{\text{total number of steps}} \times 100\%$$

In general, for an  $N$ -bit DAC, the number of different levels will be  $2^N$  and the number of steps will be  $2^N - 1$ . The greater the number of bits, the greater will be the number of steps and the smaller will be the step size, and therefore, the finer will be the resolution. Of course, the cost of the DAC increases with the number of input bits.

**Accuracy.** The accuracy of a DAC is usually specified in terms of its full-scale error and linearity error, which are normally expressed as a percentage of the converter's full-scale output. The full-scale error is the maximum deviation of the DAC's output from its expected (ideal) value, expressed as a percentage of the full-scale. The linearity error is the maximum deviation of the analog output from the ideal output. The accuracy and resolution of a DAC must be compatible.

**Settling time.** The operating speed of a DAC is usually specified by giving its settling time. It is defined as the total time between the instant when the digital input changes and the time that the output enters a specified error band for the last time, usually  $\pm 1/2$  LSB around the final value after the change in digital input. It is measured as the time for the DAC output to settle within  $\pm 1/2$  step size of its final value. Generally, DACs with a current output will have shorter settling times than those with voltage outputs.

**Offset voltage.** Ideally, the output of a DAC should be zero when the binary input is zero. In practice, however, there is a very small output voltage under this situation called the *offset voltage*. This offset error, if not corrected, will be added to the expected DAC output for all input cases.

**Monotonicity.** A DAC is said to be monotonic if its output increases as the binary input is incremented from one value to the next. This means that the staircase output will have no downward steps as the binary input is incremented from 0 to full-scale value. The DAC is said to be non-monotonic, if its output decreases when the binary input is incremented.

**EXAMPLE 17.1** Determine the resolution of (a) a 6-bit DAC and that of (b) a 12-bit DAC in terms of percentage.

**Solution**

(a) For the 6-bit DAC,

$$\% \text{ resolution} = \frac{1}{2^N - 1} \times 100 = \frac{1}{2^6 - 1} \times 100 = \frac{1}{63} \times 100 = 1.587\%$$

(b) For the 12-bit DAC,

$$\% \text{ resolution} = \frac{1}{2^N - 1} \times 100 = \frac{1}{2^{12} - 1} \times 100 = \frac{1}{4095} \times 100 = 0.0244\%$$

**EXAMPLE 17.2** A 6-bit DAC has a step size of 50 mV. Determine the full-scale output voltage and the percentage resolution.

**Solution**

With 6 bits, there will be  $2^6 - 1 = 63$  steps of size 50 mV each.

The full-scale output will, therefore, be

$$63 \times 50 \text{ mV} = 3.15 \text{ V}$$

$$\% \text{ resolution} = \frac{50 \text{ mV}}{3.15 \text{ V}} \times 100 = \frac{1}{63} \times 100 = 1.587\%$$

**EXAMPLE 17.3** An 8-bit DAC produces  $V_{\text{out}} = 0.05 \text{ V}$  for a digital input of 00000001. Find the full-scale output. What is the resolution? What is  $V_{\text{out}}$  for an input of 00101010?

**Solution**

$$\text{Full-scale output} = \text{Step size} \times \text{Number of steps}$$

$$= 0.05(2^8 - 1) = 0.05 \times 255 = 12.75 \text{ V}$$

$$\% \text{ resolution} = \frac{1}{2^N - 1} \times 100 = \frac{1}{255} \times 100 = 0.392\%$$

$$V_{\text{out}} \text{ for an input of } 00101010 = 42 \times 0.05 = 2.10 \text{ V}$$

**EXAMPLE 17.4** A certain 6-bit DAC has a full-scale output of 2 mA and a full-scale error of  $\pm 0.5\%$ . What is the range of possible outputs for an input of 100000?

**Solution**

The step size is  $2 \text{ mA}/63 = 31.7 \mu\text{A}$ . Since  $100000 = 32_{10}$ , the ideal output should be  $32 \times 31.7 = 1014 \mu\text{A}$ . The error can be as much as  $\pm 0.5\% \times 2 \text{ mA} = 10 \mu\text{A}$ .

Thus the actual output can deviate by this amount from the ideal value of  $1014 \mu\text{A}$ , and therefore, the actual output can be anywhere from  $1004 \mu\text{A}$  to  $1024 \mu\text{A}$ .

### 17.2.2 DAC Using BCD Input Code

The DAC we considered earlier used a binary input code. Many DACs use a BCD input code, where 4-bit code groups are used for each decimal digit. Figure 17.4 shows the diagram of an 8-bit (2-digit) converter of this type. Each 4-bit code group can range from 0000 to 1001, and so the BCD inputs represent decimal numbers from 00 to 99. Within each code group, the weights of the different bits are in the normal binary proportions (1, 2, 4, 8), but the relative weights of each code group are different by a factor of 10. Figure 17.4 shows the relative weights of the various bits. Note that the bits that make up the BCD code for the most significant digit (MSD) have a relative weight that is 10 times that of the corresponding bits of the LSD.

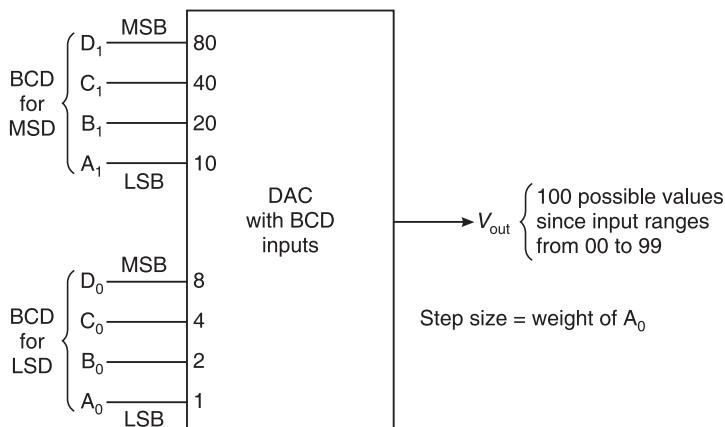


Figure 17.4 DAC using BCD input code.

**EXAMPLE 17.5** If the weight of A<sub>0</sub> is 0.2 V in Figure 17.4, find the following values:

- (a) step size, (b) full-scale output, (c) percentage resolution, and (d) V<sub>out</sub> for D<sub>1</sub>C<sub>1</sub>B<sub>1</sub>A<sub>1</sub> = 0110 and D<sub>0</sub>C<sub>0</sub>B<sub>0</sub>A<sub>0</sub> = 0100.

**Solution**

- (a) Step size is the weight of the LSB of the LSD, i.e. 0.2 V.
- (b) There are 99 steps since there are two BCD digits. Thus, the full-scale output is  $99 \times 0.2 \text{ V} = 19.8 \text{ V}$ .
- (c) The resolution is

$$\frac{\text{step size}}{\text{full-scale output}} \times 100\% = \frac{0.2}{19.8} \times 100\% \approx 1\%$$

The exact weights of the various bits in the number in volts are:

$$\text{MSD: } D_1 = 16, C_1 = 8, B_1 = 4, A_1 = 2$$

$$\text{LSD: } D_0 = 1.6, C_0 = 0.8, B_0 = 0.4, A_0 = 0.2$$

- (d) One way to find  $V_{\text{out}}$  for a given input is to add the weights of all the bits that are 1s. Thus, for an input of 01100100, we have

$$\begin{array}{ccc} C_1 & B_1 & C_0 \\ V_{\text{out}} = 8 \text{ V} + 4 \text{ V} + 0.8 \text{ V} = 12.8 \text{ V} \end{array}$$

As the BCD input code represents  $64_{10}$  and the step size is 0.2 V,  $V_{\text{out}}$  can also be determined as

$$V_{\text{out}} = (0.2 \text{ V}) \times 64 = 12.8 \text{ V}$$

- EXAMPLE 17.6** A certain 12-bit BCD DAC has a full-scale output of 19.98 V. Determine  
(a) the percentage resolution and (b) the converter's step size.

#### Solution

- (a) 12 BCD bits correspond to three decimal digits, i.e. decimal numbers from 000 to 999.

Therefore, the output of the DAC has 999 possible steps from 0 V to 19.98 V. Thus, we have

$$\% \text{ resolution} = \frac{1}{\text{number of steps}} \times 100\% = \frac{1}{999} \times 100\% \approx 0.1\%$$

$$(b) \text{Step size} = \frac{\text{full-scale output}}{\text{number of steps}} = \frac{19.98 \text{ V}}{999} = 0.02 \text{ V}$$

### 17.2.3 Bipolar DACs

So far we have assumed that the binary input to a DAC is an unsigned number and the DAC output is a positive voltage or current. Such types of DACs are called unipolar DACs.

Bipolar DACs are designed to produce both positive and negative values. This is generally done by using the binary input as a signed number with the MSB as the sign bit (0 for + and 1 for -). Negative input values are often represented in 2's complement form, although the true magnitude form is also used by some DACs. For example, suppose we have a 6-bit bipolar DAC that uses the 2's complement system and has a resolution of 0.2 V. The binary input values range from 100000 (-32) to 011111(+31) to produce analog outputs in the range from -6.4 V to +6.2 V. There are 63 steps ( $2^N - 1$ ) of 0.2 V between these negative and positive limits.

## 17.3 THE R-2R LADDER TYPE DAC

The  $R$ - $2R$  ladder type DAC is the most popular DAC. It uses a ladder network containing series-parallel combinations of two resistors of values  $R$  and  $2R$ . Hence the name. The operational amplifier configured as voltage follower is used to prevent loading. Figure 17.5 shows the circuit diagram of a  $R$ - $2R$  ladder type DAC having a 4-bit digital input. When a digital signal  $D_3D_2D_1D_0$  is applied at the input terminals of the DAC, an equivalent analog signal is produced at the output terminal. The operation of the DAC is as follows:

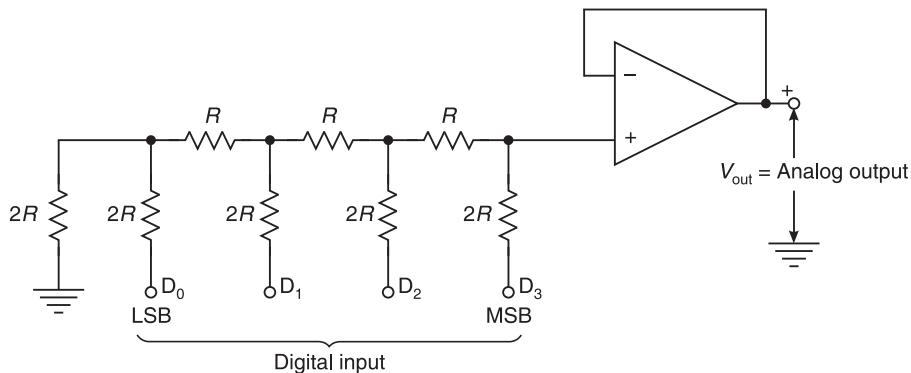
Figure 17.5 *R-2R* ladder type DAC.**Case 1: When the input is 1000**

Figure 17.6 illustrates the procedure to calculate  $V_{\text{out}}$  when the input is 1000. At the left end of the ladder,  $2R$  is in parallel with  $2R$ , so that the combination is equivalent to  $R$ . This  $R$  is in series with another  $R$  giving  $2R$ . This  $2R$  in parallel with another  $2R$  is equivalent to  $R$ . Continuing in this manner, we ultimately find that  $R_{\text{eq}} = 2R$ . The output voltage,  $V_{\text{out}} = E/2$ .

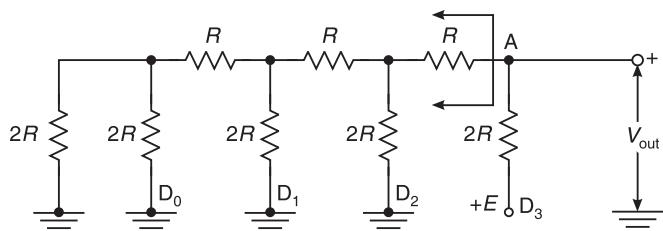
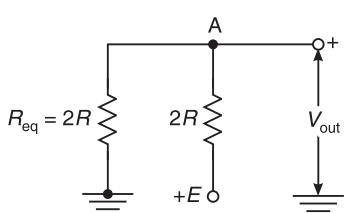
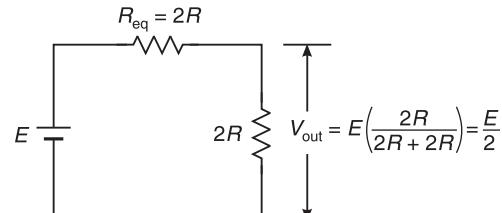
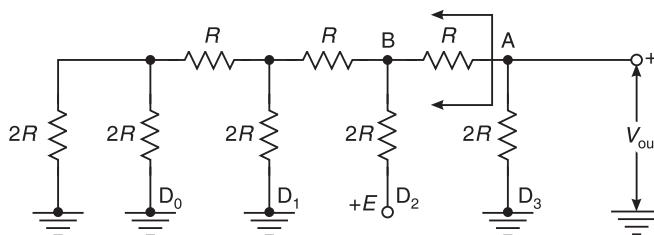
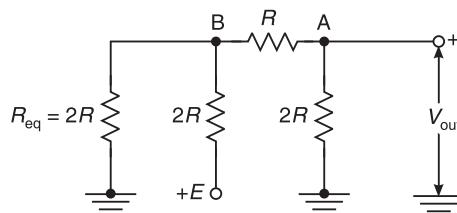
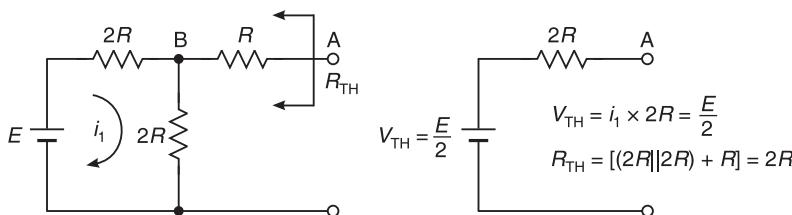
(a) When the input is 1000;  $D_0$ ,  $D_1$  and  $D_2$  are grounded (0 V) and  $D_3 = +E$ .(b) The circuit equivalent to (a) when the circuit to the left of A is replaced by its equivalent resistance  $R_{\text{eq}}$ .(c) Calculation of  $V_{\text{out}}$  using the voltage divider rule.Figure 17.6 Calculation of  $V_{\text{out}}$  when the input is 1000.**Case 2: When the input is 0100**

Figure 17.7 illustrates the procedure to calculate  $V_{\text{out}}$  when the input is 0100. Here, we find that to the left of terminal B,  $R_{\text{eq}} = 2R$ . The output voltage,  $V_{\text{out}} = E/4$ .

**Case 3: When the input is 0010**

Figure 17.8 illustrates the procedure to calculate  $V_{\text{out}}$  when the input is 0010. Here,  $R_{\text{eq}}$  to the left of C is  $2R$ . The output voltage,  $V_{\text{out}} = E/8$ .

(a) When the input is 0100;  $D_0$ ,  $D_1$  and  $D_2$  are grounded (0 V) and  $D_3 = +E$ .(b) The circuit equivalent to (a) when the circuit to the left of B is replaced by its equivalent resistance  $R_{eq}$ .

(c) Thevenin's equivalent to the left of A.

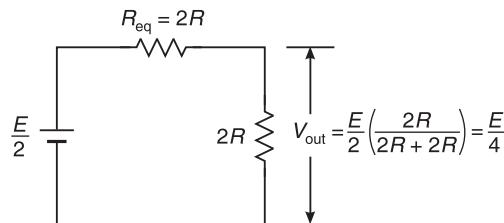
(d) Calculation of  $V_{out}$  using the voltage divider rule.**Figure 17.7** Calculation of  $V_{out}$  when the input is 0100.**Case 4: When the input is 0001**

Figure 17.9 illustrates the procedure to calculate  $V_{out}$  when the input is 0001. The output voltage,  $V_{out} = E/16$ .

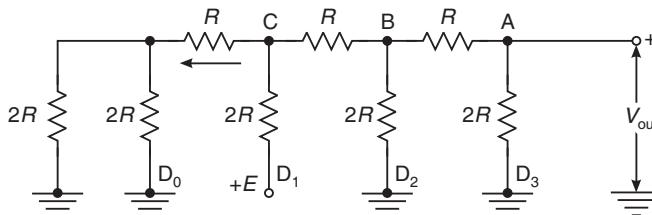
In general, when the  $D_n$  input is a 1 and all other inputs are a 0, the output is  $V_{out} = E/2^{N-n}$ , where  $N$  is the total number of binary inputs. For example, if  $E = 5$  V, then the output voltage when the input is 0100 is

$$\frac{5}{2^{4-2}} = \frac{5}{4} = 1.25 \text{ V}$$

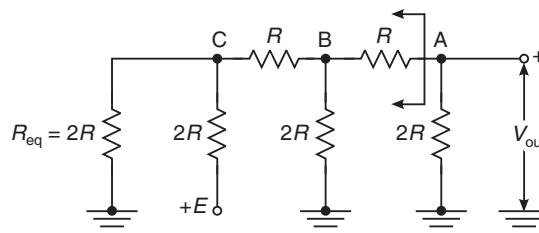
To find the output voltage corresponding to any input combination, apply the principle of superposition and simply add the voltages produced by the inputs where 1s are applied.

Typical values for  $R$  and  $2R$  are  $10\text{ k}\Omega$  and  $20\text{ k}\Omega$ , respectively. For accurate conversion, the output of the  $R-2R$  ladder network is connected to a high impedance circuit to prevent loading. An op-amp configured as a voltage follower can be used for this purpose as shown in Figure 17.5.

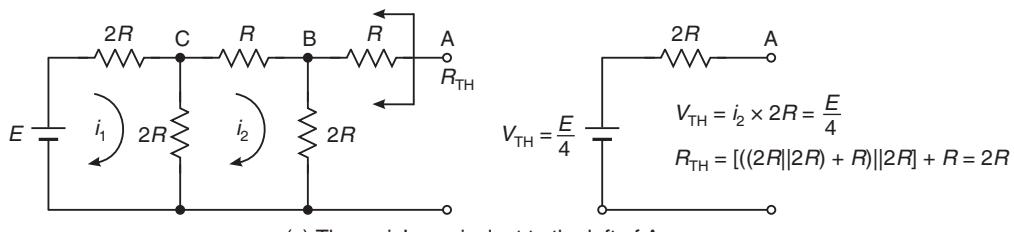
The principal advantage of this converter is that resistors of only two values are required. Therefore, standard resistors can be used.



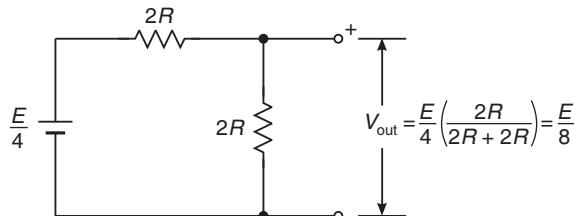
(a) When the input is 0010;  $D_0$ ,  $D_2$  and  $D_3$  are grounded (0 V) and  $D_1 = +E$ .



(b) The circuit equivalent to (a) when the circuit to the left of C is replaced by  $R_{eq} = 2R$ .

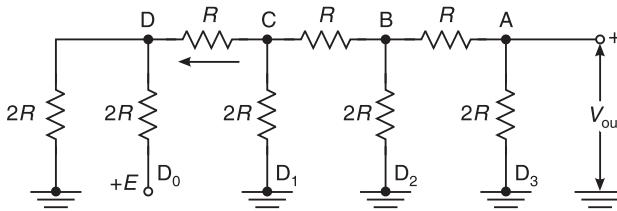
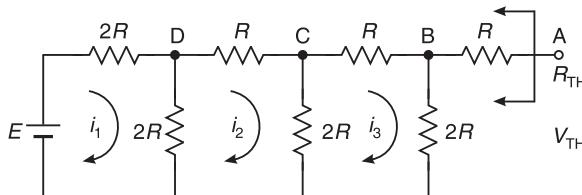


(c) Thevenin's equivalent to the left of A.

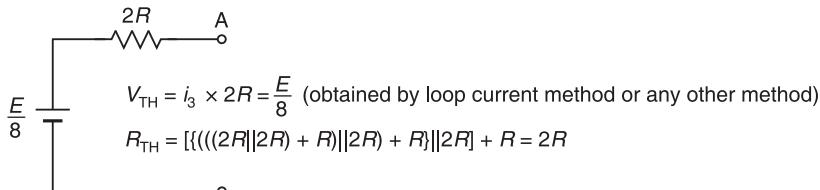


(d) Calculation of  $V_{out}$  using the voltage divider rule.

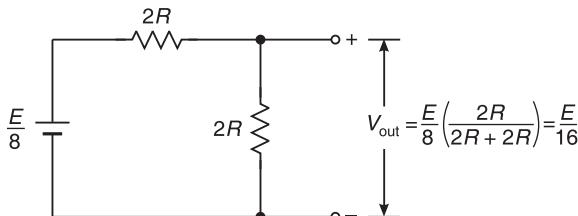
**Figure 17.8** Calculation of  $V_{out}$  when the input is 0010.

(a) When the input is 0001;  $D_1, D_2$  and  $D_3$  are grounded (0 V) and  $D_0 = +E$ .

(b) The circuit equivalent to (a).



(c) Thevenin's equivalent to the left of A.

(d) Calculation of  $V_{\text{out}}$  using the voltage divider rule.**Figure 17.9** Calculation of  $V_{\text{out}}$  when the input is 0001.

**EXAMPLE 17.7** What are the output voltages caused by logic 1 in each bit position in an 8-bit ladder if the input level for 0 is 0 V and that for 1 is + 10 V?

**Solution**

$$\text{The output voltage level caused by the } D_n \text{ bit} = \frac{E}{2^{N-n}}$$

$$\text{The output voltage level caused by the } D_7 \text{ bit (MSB)} = \frac{E}{2^{8-7}} = \frac{E}{2^1} = \frac{10}{2} = 5 \text{ V}$$

$$\text{The output voltage level caused by the } D_6 \text{ bit} = \frac{E}{2^2} = \frac{10}{4} = 2.5 \text{ V}$$

The output voltage level caused by the  $D_5$  bit =  $\frac{E}{2^3} = \frac{10}{8} = 1.25$  V

The output voltage level caused by the  $D_4$  bit =  $\frac{E}{2^4} = \frac{10}{16} = 0.625$  V

The output voltage level caused by the  $D_0$  bit (LSB) =  $\frac{E}{2^8} = \frac{10}{256} = 0.03906$  V

**EXAMPLE 17.8** What is the resolution of a 9-bit DAC, which uses a ladder network? What is this resolution expressed as a percentage? If the full-scale output voltage of this converter is + 5 V, what is the resolution in volts?

### Solution

The LSB in a 9-bit system has a weight of 1/512. Thus, this converter has a resolution of 1 part in 512. The resolution expressed as a percentage is,  $(1/512) \times 100 \approx 0.2\%$ . The voltage resolution is obtained by multiplying the weight of the LSB by the full-scale output voltage. Thus, the resolution in volts is,  $(1/512) \times 5\text{ V} = 9.8\text{ mV}$ .

## 17.4 THE WEIGHTED-RESISTOR TYPE DAC

The diagram of the weighted-resistor DAC is shown in Figure 17.10. The operational amplifier is used to produce a weighted sum of the digital inputs, where the weights are proportional to the weights of the bit positions of inputs. Since the op-amp is connected as an inverting amplifier, each input is amplified by a factor equal to the ratio of the feedback resistance divided by the input resistance to which it is connected. The MSB  $D_3$  is amplified by  $R_f/R$ ,  $D_2$  is amplified by  $R_f/2R$ ,  $D_1$  is amplified by  $R_f/4R$ , and  $D_0$ , the LSB is amplified by  $R_f/8R$ .

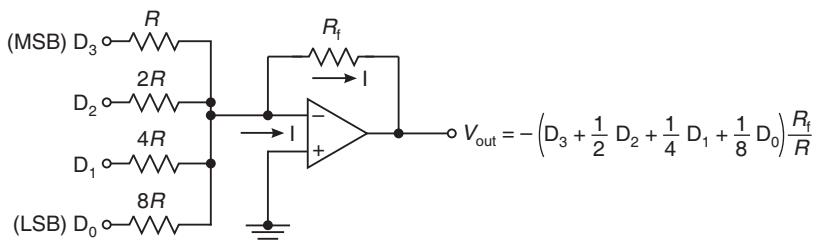


Figure 17.10 Weighted-resistor type DAC.

The inverting terminal of the op-amp in Figure 17.10 acts as a virtual ground. Since the op-amp adds and inverts,

$$V_{\text{out}} = - \left( D_3 + \frac{D_2}{2} + \frac{D_1}{4} + \frac{D_0}{8} \right) \times \left( \frac{R_f}{R} \right)$$

The main disadvantage of this type of DAC is, that a different-valued precision resistor must be used for each bit position of the digital input.

**EXAMPLE 17.9** For the 4-bit weighted-resistor DAC shown in Figure 17.10, determine (a) the weight of each input bit if the inputs are 0 V and 5 V, (b) the full-scale output, if  $R_f = R = 1 \text{ k}\Omega$ . Also, find the full-scale output if  $R_f$  is changed to  $500 \Omega$ .

**Solution**

- (a) The MSB passes with a gain of 1, so, its weight = 5 V; the next bit passes with a gain of  $1/2$ , so, its weight =  $5/2 = 2.5$  V; the following bit passes with a gain of  $1/4$ , so, its weight =  $5/4 = 1.25$  V; the LSB passes with a gain of  $1/8$ , so, its weight =  $5/8 = 0.625$  V.  
 (b) Therefore, the full scale output

$$\begin{aligned} (\text{when } R_f = R = 1 \text{ k}\Omega) &= -\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8}\right) \times 5 \\ &= -9.375 \text{ V} \end{aligned}$$

The full-scale output, when  $R_f$  is changed to  $500 \Omega$  is

$$V_{\text{out}} = -\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8}\right) \times \left(\frac{5}{2}\right) = -4.6875 \text{ V}$$

**EXAMPLE 17.10** Determine the output of the DAC in Figure 17.11a, if the sequence of the 4-bit numbers in Figure 17.11b is applied to the inputs.  $D_0$  is the LSB.

**Solution**

First, let us determine the output current  $I$  for each of the weighted inputs. Since the inverting input (-) of the op-amp is at 0 V (virtual ground) and a binary 1 corresponds to +5 V, the current through any of the input resistors is 5 V divided by the resistance value. Thus,

$$\begin{aligned} I_0 &= \frac{5 \text{ V}}{400 \text{ k}\Omega} = 0.0125 \text{ mA}; & I_1 &= \frac{5 \text{ V}}{200 \text{ k}\Omega} = 0.025 \text{ mA} \\ I_2 &= \frac{5 \text{ V}}{100 \text{ k}\Omega} = 0.05 \text{ mA}; & I_3 &= \frac{5 \text{ V}}{50 \text{ k}\Omega} = 0.1 \text{ mA} \end{aligned}$$

The input impedance of the op-amp is extremely large; therefore, the current into the op-amp is zero. Thus, the current,  $I_0 + I_1 + I_2 + I_3$ , has to go through the feedback resistance  $R_f$ . Since one end of  $R_f$  is at 0 V (virtual ground), the drop across  $R_f$  equals the output voltage.

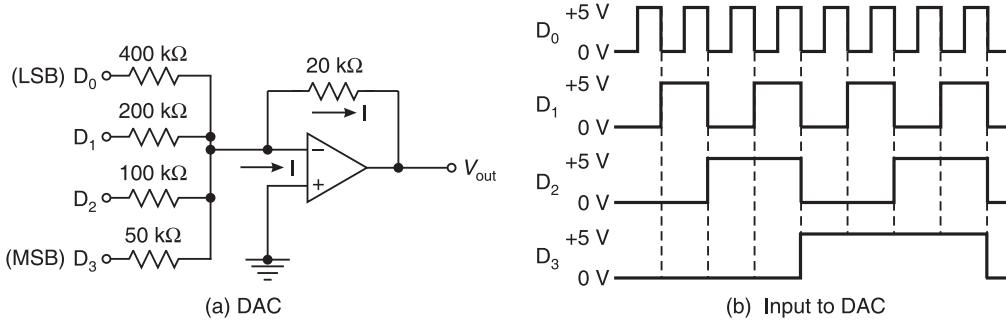


Figure 17.11 Example 17.10: DAC.

The output voltage due to input  $D_0$  is,  $20 \text{ k}\Omega (-0.0125 \text{ mA}) = -0.25 \text{ V}$

The output voltage due to input  $D_1$  is,  $20 \text{ k}\Omega (-0.025 \text{ mA}) = -0.5 \text{ V}$

The output voltage due to input  $D_2$  is,  $20 \text{ k}\Omega (-0.05 \text{ mA}) = -1 \text{ V}$

The output voltage due to input  $D_3$  is,  $20 \text{ k}\Omega (-0.1 \text{ mA}) = -2 \text{ V}$

From Figure 17.11b, the first input code is 0001. For this, the output voltage is  $-0.25 \text{ V}$ . The next input code is 0010, for which the output voltage is  $-0.5 \text{ V}$ . The next code is 0011 for which the output voltage is  $-0.75 \text{ V}$ , and so on. Each successive binary code increases the output voltage by  $-0.25 \text{ V}$ . So, for this particular straight binary sequence of the inputs, the output is a staircase waveform going from  $0 \text{ V}$  to  $-3.75 \text{ V}$  in  $-0.25 \text{ V}$  steps. The output waveform is shown in Figure 17.12.

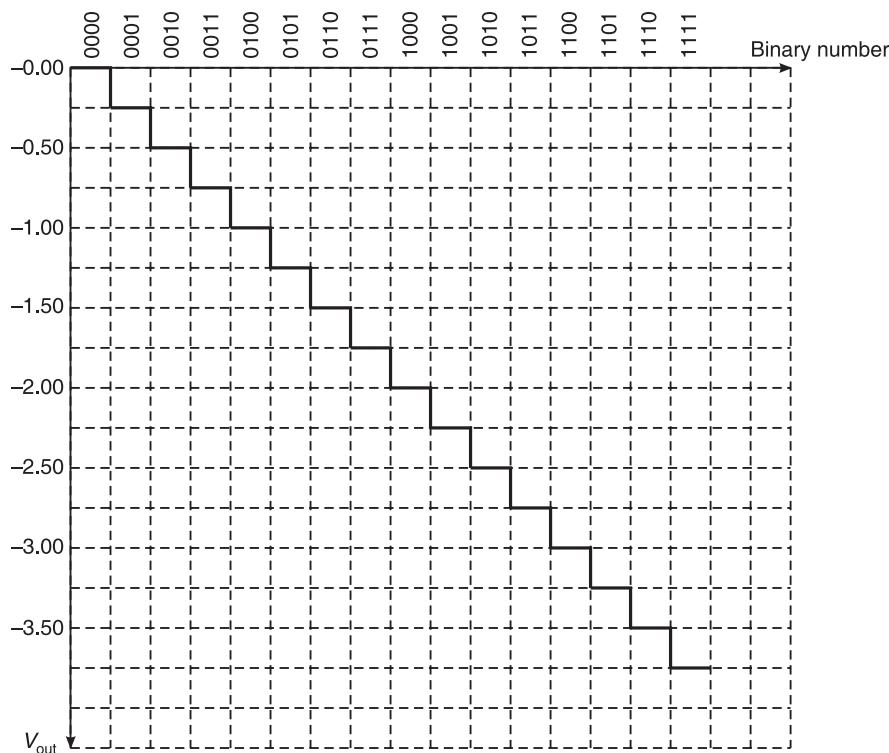


Figure 17.12 Example 17.10: Output waveform.

**EXAMPLE 17.11** Design a 4-bit weighted-resistor DAC whose full-scale output voltage is  $-5 \text{ V}$ . The logic levels are  $1 = +5 \text{ V}$  and  $0 = 0 \text{ V}$ . What is the output voltage when the input is 1101?

### Solution

The full-scale output voltage is the output voltage when the input voltage is maximum, i.e. 1111. Thus,

$$\text{The full-scale output} = -\left(5 \text{ V} + \frac{5 \text{ V}}{2} + \frac{5 \text{ V}}{4} + \frac{5 \text{ V}}{8}\right) \times \frac{R_f}{R} = -5 \text{ V}$$

Therefore,

$$9.375 \times \frac{R_f}{R} = 5$$

Choosing  $R_f = 10 \text{ k}\Omega$ , and since  $\frac{R_f}{R} = \frac{5}{9.375}$ , we have

$$R = R_f \times \frac{9.375}{5} = 18.75 \text{ k}\Omega$$

Thus,

$$2R = 37.5 \text{ k}\Omega; \quad 4R = 75 \text{ k}\Omega; \quad 8R = 150 \text{ k}\Omega$$

When the input is 1101, the output voltage is

$$V_{\text{out}} = - \left( 5 \text{ V} + \frac{5 \text{ V}}{2} + 0 + \frac{5 \text{ V}}{8} \right) \times \frac{R_f}{R} = -8.125 \times \frac{10}{18.75} = -4.333 \text{ V}$$

## 17.5 THE SWITCHED CURRENT-SOURCE TYPE DAC

The  $R-2R$  ladder network DAC and the weighted-resistor DAC can be regarded as switched voltage-source DACs, because when an input position goes HIGH, the HIGH voltage is effectively switched into the circuit, where it is summed up with other input voltages. Most integrated circuit DACs, however, use some form of current switching rather than voltage switching, since currents can be switched in and out of the circuit faster than voltages. In the former type, the binary inputs are used to open and close switches that connect and disconnect internally generated currents. The currents are weighted according to the bit positions they represent and are summed up in an operational amplifier.

Figure 17.13 shows the circuit diagram of a current-switching type DAC. Note that an  $R-2R$  ladder is connected to a voltage source  $E_{\text{REF}}$ . The current in the first  $2R$  resistor from supply is given by  $I_3 = E_{\text{REF}}/2R$ , because  $E_{\text{REF}}$  is directly applied across  $2R$ . The current in the second  $2R$  is given by  $E_{\text{REF}}/4R$  because the equivalent resistance to the right of the second  $2R$  is  $2R$  and, so, the current  $E_{\text{REF}}/(R + R)$  coming into the first  $R$  is equally divided between the second  $2R$  and the  $2R$  to its right.

In general, the current that flows in each  $2R$  resistor is given by,  $I_n = (E_{\text{REF}}/R) \times 1/2^{N-n}$  where  $n = 0, 1, \dots, N-1$  is the subscript for the current created by input  $D_n$  and  $N$  is the total number of inputs. Thus, each current is weighted by the bit position it represents.

The switches that connect the currents either to ground or to the input of the operational amplifier are controlled by the digital input. The op-amp sums up all those currents whose corresponding digital inputs are HIGH. The amplifier also serves as a voltage-to-current converter. It is connected in an inverting configuration and its output is  $V_{\text{out}} = -I_T R$ , where  $I_T$  is the sum of the currents that have been switched to its input.

If  $E_{\text{REF}}$  is an externally variable voltage, the output of the DAC is proportional to the product of the variable  $E_{\text{REF}}$  and the variable digital input. In that case, the circuit is called a *multiplying* DAC and the output represents the product of an analog input  $E_{\text{REF}}$  and a digital input.

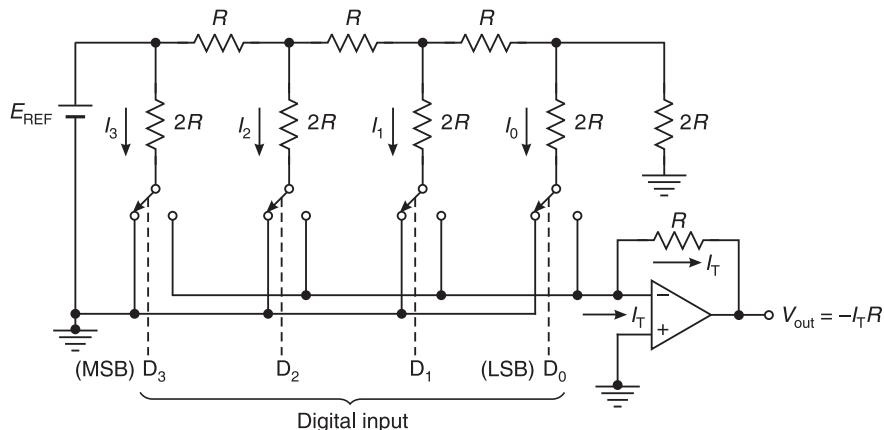


Figure 17.13 The 4-bit switched current-source DAC.

**EXAMPLE 17.12** The switched current-source DAC in Figure 17.13 has  $R = 5 \text{ k}\Omega$  and  $E_{\text{REF}} = 10 \text{ V}$ . Find the total current delivered to the amplifier and the output voltage when the digital input is 1101.

**Solution**

Since the digital input is 1101, the total current into the amplifier, i.e.  $I_T$  is given by

$$\begin{aligned} I_T &= I_3 + I_2 + I_0 \\ &= \left( \frac{E_{\text{REF}}}{2R} \right) + \left( \frac{E_{\text{REF}}}{4R} \right) + \left( \frac{E_{\text{REF}}}{16R} \right) \\ &= \left( \frac{10}{10} + \frac{10}{20} + \frac{10}{80} \right) = \frac{130}{80} = \frac{13}{8} \text{ mA} \end{aligned}$$

Therefore, the output voltage,

$$V_{\text{out}} = -I_T \times R = \left( -\frac{13}{8} \times 5 \right) = -8.125 \text{ V}$$

**EXAMPLE 17.13** In a 4-bit DAC, for a digital input of 0100 an output current of 10 mA is produced. What will be the output current for a digital input of 1011?

**Solution**

$$\begin{aligned} \text{Output current} &= 10 \text{ mA for a digital input of } 0100, \text{ i.e. } 4_{10}. \\ \text{Analog output} &= K \times \text{digital input} \end{aligned}$$

Therefore,

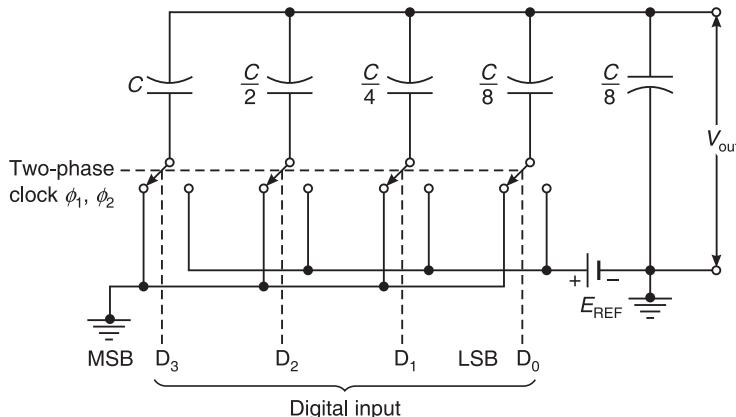
$$K = \frac{10 \text{ mA}}{4} = 2.5 \text{ mA}$$

Hence the output current for a digital input of 1011 is,  $11_{10} \times 2.5 = 27.5 \text{ mA}$ .

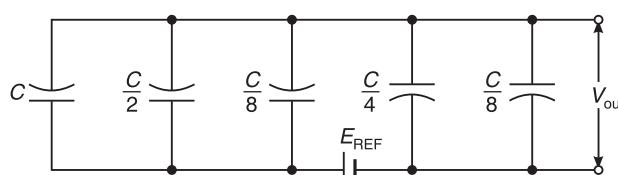
## 17.6 THE SWITCHED-CAPACITOR TYPE DAC

The switched-capacitor type DAC employs weighted capacitors instead of weighted resistors. In this method, charged capacitors form a capacitive voltage divider whose output is proportional to the sum of the binary inputs.

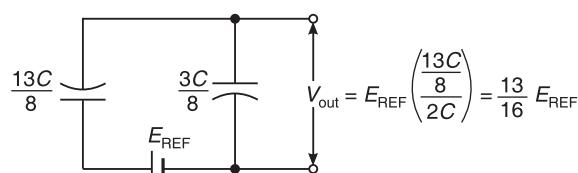
Figure 17.14a shows an example of a 4-bit switched-capacitor DAC. Note that the capacitance values have binary weights. A two-phase clock is used to control switching of the capacitors. When  $\phi_1$  goes HIGH, all capacitors are switched to ground and discharged. When  $\phi_2$  goes HIGH, those capacitors where the digital inputs are HIGH are switched to  $E_{REF}$ , whereas those whose inputs are LOW remain grounded. Figure 17.14b shows the equivalent circuit when  $\phi_2$  is HIGH and the digital input is 1101.



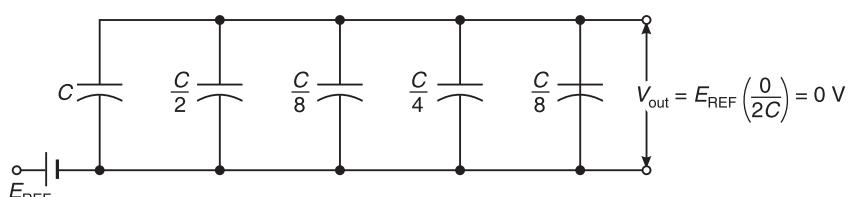
(a) All capacitors are switched to ground by  $\phi_1$ . Those capacitors whose digital inputs are HIGH are switched to  $E_{REF}$  by  $\phi_2$ .



(b) Equivalent circuit when the input is 1101. The capacitors switched to  $E_{REF}$  are in parallel as are the ones connected to ground.



(c) Circuit equivalent to (b). The output is determined by a capacitor voltage divider.



(d) Equivalent circuit when the input is 0000.

**Figure 17.14** The switched-capacitor type DAC.

We see that the capacitors whose digital inputs are a 1, are in parallel and the capacitors whose digital inputs are a 0, are in parallel with  $C/8$ . The circuit is redrawn in Figure 17.14c, where each set of the parallel capacitors is replaced by its equivalent capacitance. The output of the capacitive voltage divider is

$$V_{\text{out}} = E_{\text{REF}} \left( \frac{\frac{13}{8}C}{2C} \right) = \frac{13}{16} E_{\text{REF}}$$

In general, for any binary input,

$$V_{\text{out}} = \left( \frac{C_{\text{EQ}}}{2C} \right) \times E$$

where  $2C$  is the sum of all the capacitance values in the circuit and  $C_{\text{EQ}}$  is the sum of all the capacitors whose digital inputs are HIGH.

The analog output is proportional to the digital input. When the input is 0000, the positive terminal of  $E_{\text{REF}}$  is effectively open-circuited as shown in Figure 17.14d; so, the output is 0 V.

Switched-capacitor technology has been developed for implementing analog functions in integrated circuits, particularly MOS circuits. It is used to construct filters, amplifiers, and many other special devices. The principal advantage of this technology is that, small capacitors of the order of a few picofarads can be constructed in the integrated circuits to perform the function of the much larger capacitors that are normally needed in low-frequency analog circuits.

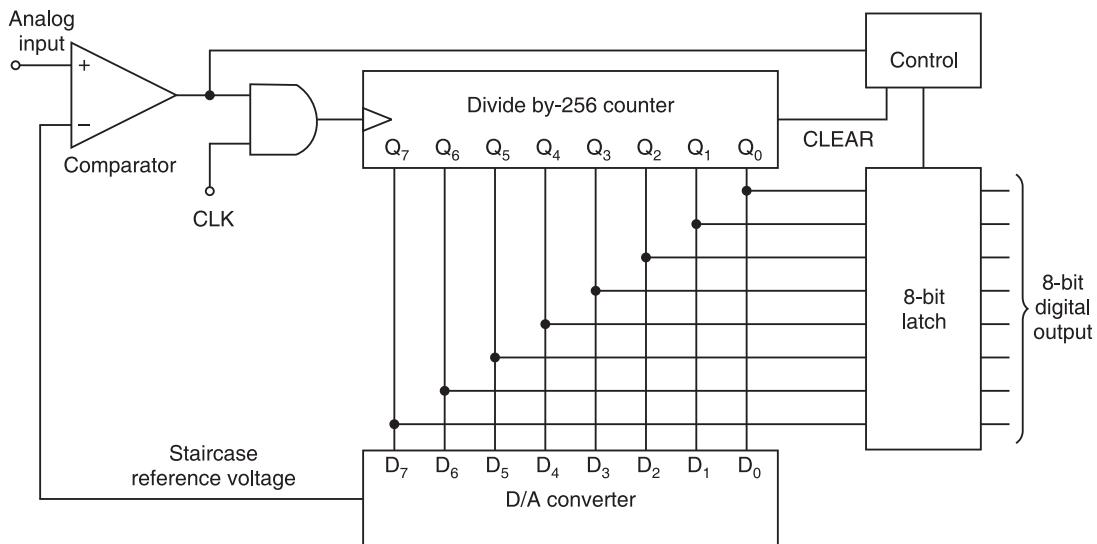
## 17.7 ANALOG-TO-DIGITAL CONVERSION

An analog-to-digital converter (A/D converter or ADC) produces a digital output that is proportional to the value of the input analog signal. When an analog signal is processed by a digital system, an ADC is used to convert the analog voltage to a digital form suitable for processing by a digital system.

## 17.8 THE COUNTER-TYPE A/D CONVERTER

This is the simplest type of the A/D converter. It employs a binary counter, a voltage comparator, a control circuit, an AND gate, latches, and a D/A converter as shown in Figure 17.15. It is also called a *digital ramp* ADC, because the waveform at the output of the DAC is a step-by-step ramp (actually a staircase). The analog signal to be converted is applied to the non-inverting terminal of the op-amp comparator. The output of the DAC is applied to the inverting terminal of the op-amp. Whenever the analog input signal is greater than the DAC output, the output of the op-amp is HIGH and whenever the output of the DAC is greater than the analog signal, the output of the comparator is LOW. The comparator output serves as an active-low end of the conversion signal.

Assume that initially the counter is reset and, therefore, the output of the DAC is zero. Since the analog input is larger than the initial output of the DAC, the output of the comparator is HIGH and, therefore, the AND gate is enabled and, so, the clock pulses are transmitted to the counter and the counter advances through its binary states. These binary states are converted



**Figure 17.15** Logic diagram of the counter-type ADC.

into reference analog voltage (which is in the form of a step) by the DAC. The counter continues to advance from one state to the next, producing successively larger steps in the reference voltage. When the staircase output voltage reaches the value of the analog signal, the comparator outputs a LOW, and the AND gate is disabled; so, the clock pulses do not reach the counter and the counter stops. The count it reached is the digital output proportional to the analog input. The control logic loads the binary count into the latches and resets the counter, thus, beginning another count sequence to sample the input value. The cycle thus repeats itself.

The resolution of this ADC is equal to the resolution of the DAC it contains. The resolution can also be thought of as the built-in error and is often referred to as the *quantization error*. Thus,

$$\text{Resolution} = \frac{\text{FSR}}{2^N}$$

where FSR is the full-scale reading and  $N$  is the number of bits in the counter.

As in the DAC, the accuracy is not related to the resolution, but is dependent on the accuracy of the circuit components such as comparator, the DAC's precision resistors, etc.

Figure 17.16 illustrates the output of a 4-bit DAC in an ADC over several cycles when the analog input is a slowly varying voltage.

The principal disadvantage of this type of converter is that, the conversion time depends on the magnitude of the analog input. The larger the input, the more will be the number of clock pulses that must pass to reach the proper count, and, therefore, the larger will be the conversion time. For each conversion, the counter has to start from reset only and count up to the point at which the staircase reference voltage reaches the analog input voltage. This type of converter is considered quite slow in comparison with the other types.

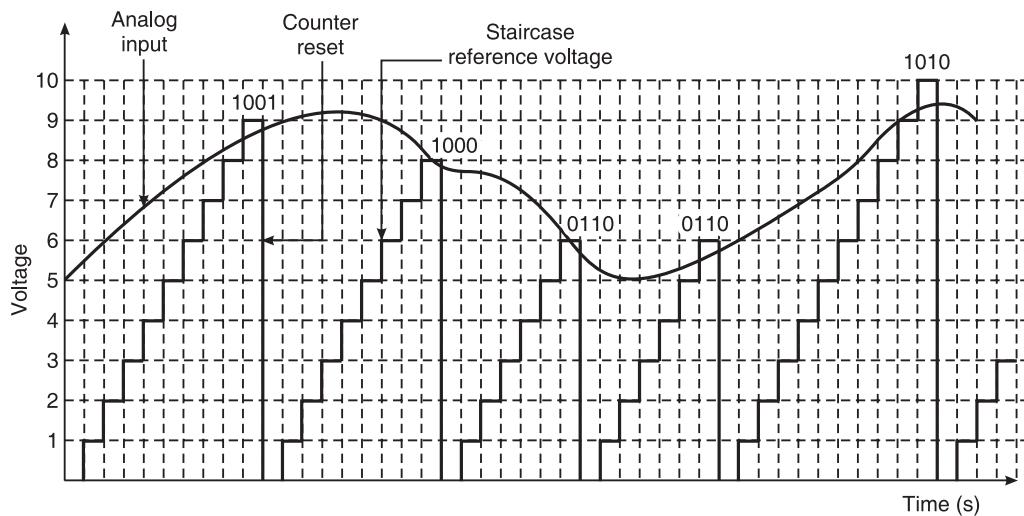


Figure 17.16 Output waveform of the counter-type ADC.

## 17.9 THE TRACKING-TYPE A/D CONVERTER

The counter-type ADC is slow, because the counter resets itself after each conversion. The tracking-type ADC uses an up/down counter and is faster than the counter-type ADC, because the counter is not reset after each sample, but tends to track the analog input, i.e. counts up or down from its last count to its new count. Thus, the total number of clock pulses required to perform a conversion is proportional to the change in the analog input between counts, rather than to its magnitude. Since the count more or less keeps up with the changing analog input, this type of converter is called a *tracking converter*.

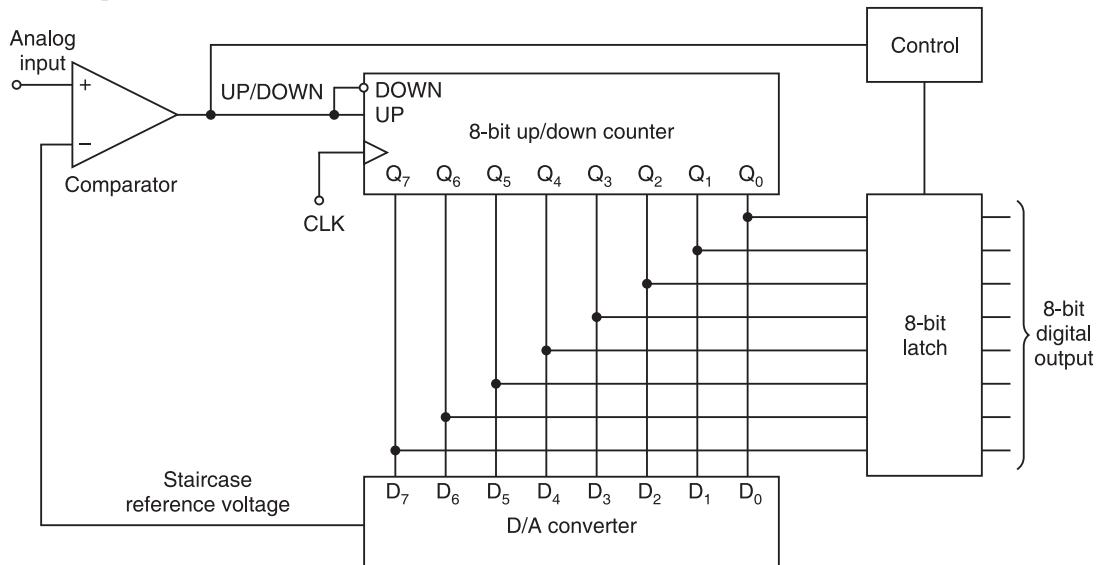
Figure 17.17 shows the logic diagram of a tracking type-ADC. As long as the D/A output reference voltage is less than the analog input, the comparator output is HIGH putting the counter in the up mode and causing it to produce an up-sequence of binary counts. This causes an increasing value of the reference voltage out of the D/A converter, which continues until the staircase reaches the value of the input analog voltage. When the reference voltage equals the analog input, the comparator output switches LOW and puts the counter in the down mode, causing it to back up one count. If the analog input is now decreasing, the counter will continue to back down in its sequence and effectively track the input. If the analog input remains constant, the counter keeps on changing from up-to-down-to-up continuously and, therefore, the output of the ADC keeps on oscillating about the constant analog input. This is a disadvantage of this type of converter. Figure 17.18 shows the output waveform of a tracking-type ADC.

The conversion time of this ADC is the time interval between the starting of the conversion and the time the comparator outputs a LOW (stopping the count). That is, the conversion time is the time required to convert a single analog input to a digital output.

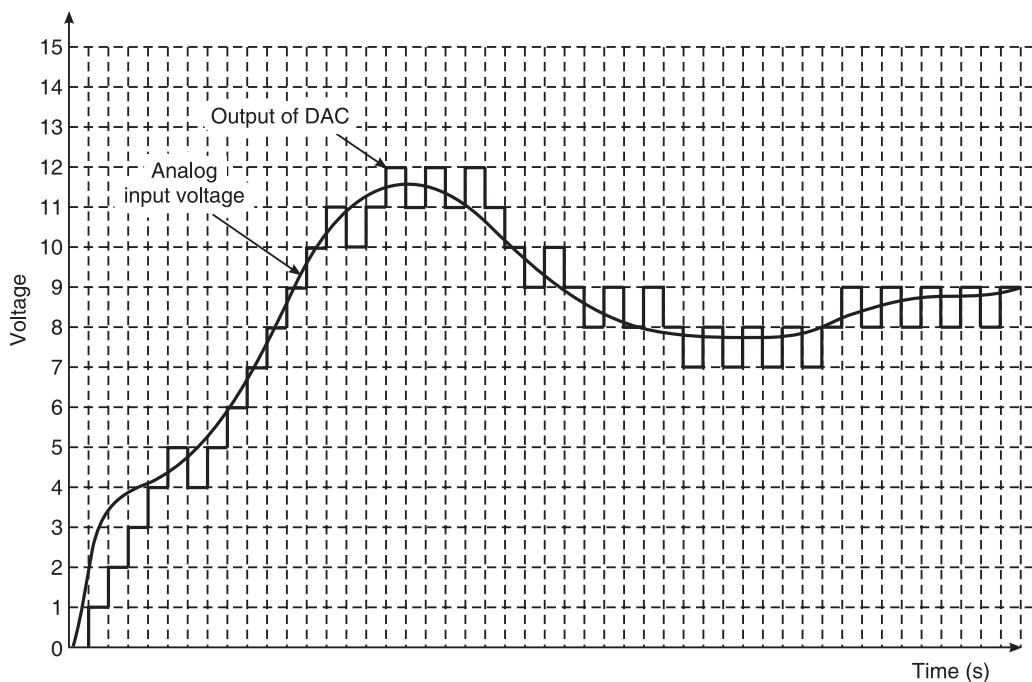
$$\begin{aligned} t_c(\max) &= (2^N - 1) \text{ clock cycles} \\ &= (2^N - 1) \times \text{time for 1 cycle} \end{aligned}$$

$$\text{The average conversion time} = \frac{t_c(\max)}{2}$$

The ADC must perform at a rate equal to at least twice the frequency of the highest component of the input.



**Figure 17.17** Logic diagram of the tracking-type ADC.



**Figure 17.18** Output waveform of the tracking-type ADC.

**EXAMPLE 17.14** Determine the maximum conversion time that an ADC can have, if it is used to convert signals in the range of 1 kHz to 50 kHz.

**Solution**

Since the highest input frequency is 50 kHz, conversions should be performed at the rate of  $2 \times 50 \times 10^3 = 100 \times 10^3$  conversions/s. The maximum allowable conversion time is, therefore, equal to  $1/(100 \times 10^3) = 10 \mu\text{s}$ .

**EXAMPLE 17.15** An ADC has a total conversion time of 200  $\mu\text{s}$ . What is the highest frequency that its analog input should be allowed to contain?

**Solution**

The highest frequency that the analog signal can contain is

$$\frac{1}{2 \times \text{conversion time}} = \frac{1}{2 \times 200 \mu\text{s}} = 2.5 \text{ kHz}$$

## 17.10 THE FLASH-TYPE A/D CONVERTER

The flash (or simultaneous or parallel) type A/D converter is the fastest type of A/D converter. This type of converter utilizes the parallel differential comparators that compare reference voltages with the analog input voltage. The main advantage of this type of converter is that the conversion time is less, but the disadvantage is that, an  $n$ -bit converter of this type requires  $2^n - 1$  comparators,  $2^n$  resistors, and a priority encoder. Figure 17.19 shows a 3-bit flash type A/D converter which

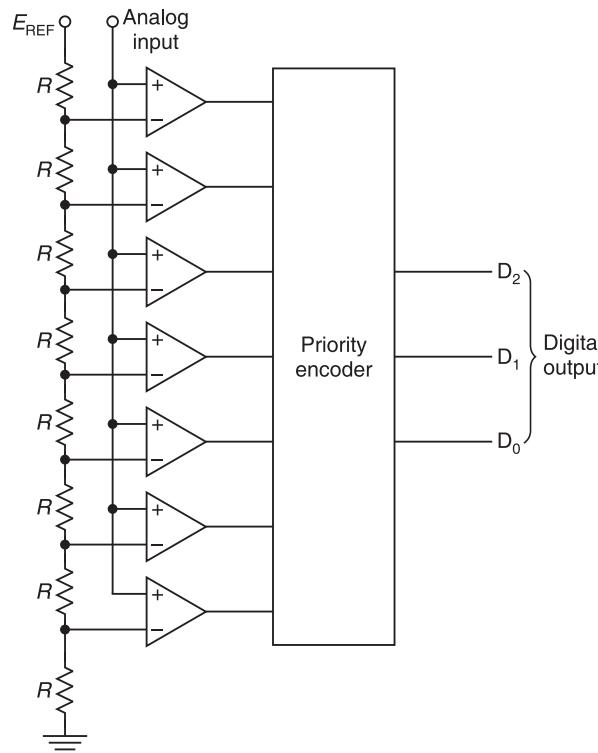


Figure 17.19 The flash-type ADC.

requires  $7(= 2^3 - 1)$  comparators. A reference voltage  $E_{\text{REF}}$  is connected to a voltage divider that divides it into seven equal increment levels. Each level is compared to the analog input by a voltage comparator. For any given analog input, one comparator and all those below it will have a HIGH output. All comparator outputs are connected to a priority encoder, which produces a digital output corresponding to the input having the highest priority, which in this case is the one that represents the largest input. Thus, the digital output represents the voltage that is closest in value to the analog input.

The voltage applied to the inverting terminal of the uppermost comparator in Figure 17.19 is (by voltage divider action),

$$\left(\frac{7R}{7R + R}\right) \times E_{\text{REF}} = \frac{7}{8} \times E_{\text{REF}}$$

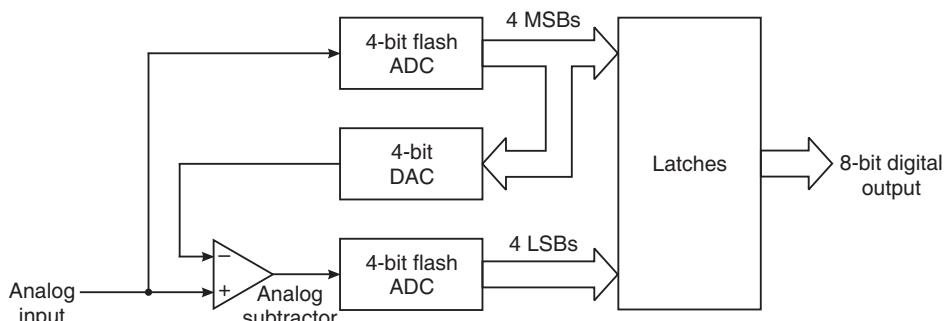
Similarly, the voltage applied to the inverting terminal of the second comparator is

$$\left(\frac{6R}{7R + R}\right) \times E_{\text{REF}} = \frac{6}{8} \times E_{\text{REF}}$$

and so forth. The increment between voltages is  $\frac{1}{8} \times E_{\text{REF}}$ .

The flash converter uses no clock signal, because there is no timing or sequencing period. The conversion takes place continuously. The only delays in the conversion are in the comparators and the priority encoders.

Figure 17.20 shows the block diagram of a modified flash A/D converter. To perform an 8-bit conversion, it requires two 4-bit flash converters. So, an 8-bit conversion can be done by using  $30[= 2 \times (2^4 - 1)]$  comparators instead of  $255(= 2^8 - 1)$  comparators. One 4-bit flash converter is used to produce the four most significant bits (MSBs). Those four bits are converted back to an analog voltage by a D/A converter and this voltage is subtracted from the analog input. The difference between the analog input and the analog voltage corresponding to the four most significant bits, is an analog voltage corresponding to the four least significant bits (LSBs). Therefore, that voltage is converted to the four least significant bits by another 4-bit flash converter.



**Figure 17.20** Modified flash ADC.

**EXAMPLE 17.16** Determine the digital output of a 3-bit simultaneous A/D converter for the analog input signal and the sampling pulses (encoder enable) shown in Figure 17.21.  $V_{\text{REF}} = + 8 \text{ V}$ .

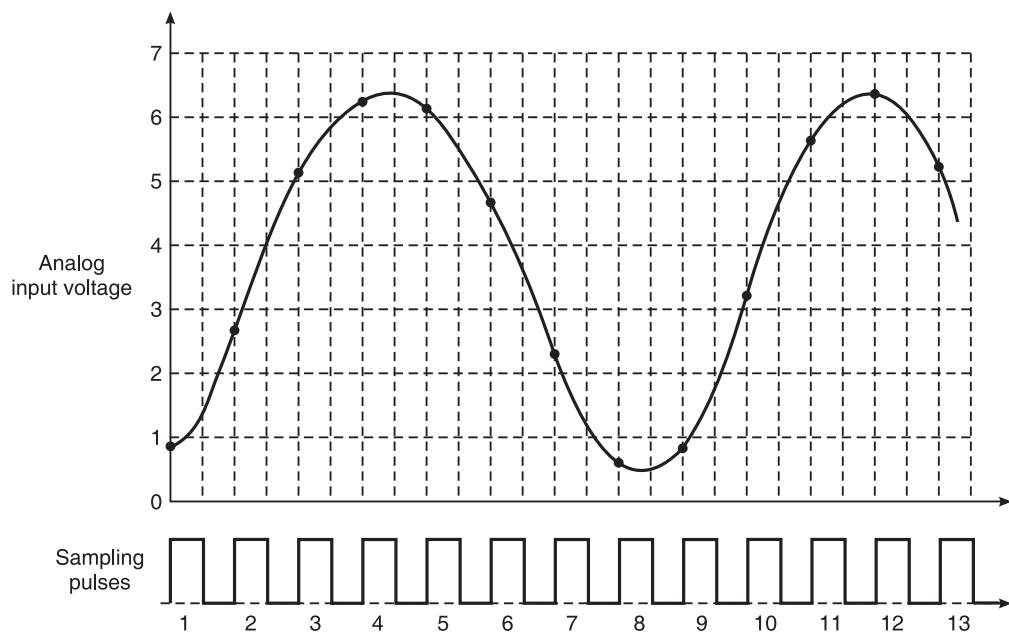


Figure 17.21 Example 17.16: Waveforms.

**Solution**

The resulting A/D output sequence is listed as follows and shown in Figure 17.22 in relation to the sampling pulses.

000, 010, 101, 110, 110, 100, 010, 000, 000, 011, 101, 110.

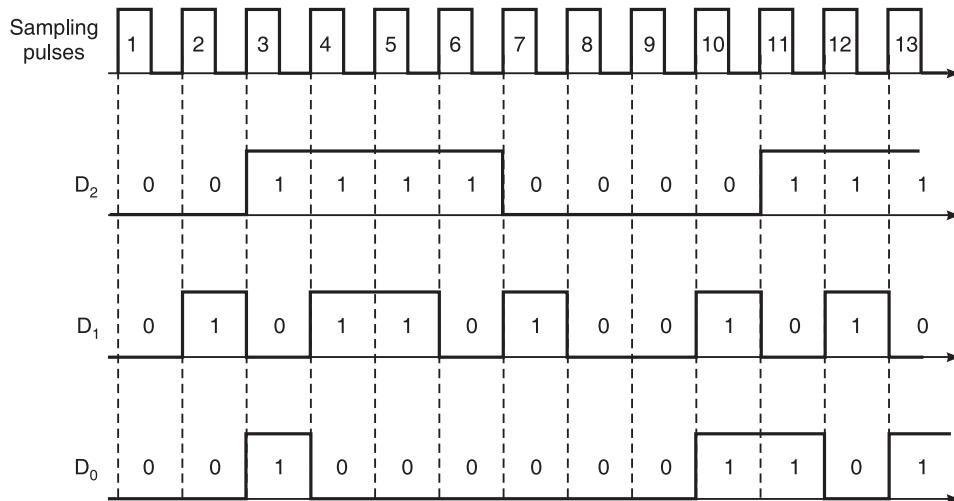


Figure 17.22 Example 17.16: A/D output sequence.

### 17.11 THE DUAL-SLOPE TYPE A/D CONVERTER

The dual-slope converter is one of the slowest converters, but is relatively inexpensive because it does not require precision components such as a DAC or VCO. Another advantage of the dual-slope ADC is its low sensitivity to noise, and to variations in its component values caused by temperature changes. Because of its large conversion time, the dual-slope ADC is not used in any data acquisition applications. The major applications of this type of converter are in digital voltmeters, multimeters, etc. where slow conversions are not a problem. Since it is not fast enough, its use is restricted to signals having low to medium frequencies.

A dual-slope ADC uses an operational amplifier to integrate the analog input. The output of the integrator is a ramp, whose slope is proportional to the input signal  $E_{in}$ , since the components  $R$  and  $C$  are fixed. If the ramp is allowed to continue for a fixed time, the voltage it reaches in that time, depends on the slope of the ramp and, therefore, on the value of  $E_{in}$ . The basic principle of the integrating ADC is that, the voltage reached by the ramp controls the length of time that the binary counter is allowed to count. Thus, a binary number proportional to the value of  $E_{in}$  is obtained. In the dual-slope ADC, two integrations are performed.

Figure 17.23 shows the functional block diagram of a dual-slope ADC. Assume that the counter is reset and the output of the integrator is zero. A conversion begins with the switch connected to the analog input. Assume that the input is a negative voltage and is constant for a period of time; so, the output of the integrator is a positive ramp. The ramp is allowed to continue for a fixed time and the voltage it reaches in that time is directly dependent on the analog input. The fixed time is controlled by sensing the time when the counter reaches a particular count. At that time, the counter is reset and the control circuitry causes the switch to be connected to a reference voltage  $E_{REF}$ , having a polarity opposite to that of the analog input; in this case a positive reference voltage. Therefore, the output of the integrator is a negative going ramp, beginning from the positive value it reached during the first integration. The AND gate is enabled and the counter starts counting. When the ramp reaches 0 V, the voltage comparator switches to LOW, inhibiting the clock pulses and the counter stops counting. The binary count is latched, thus, completing one conversion. The count it contains at that time is proportional to the time required for the negative ramp to reach zero, which is proportional to the positive voltage reached during the first integration, which in turn is proportional to the analog input.

The accuracy of the converter does not depend on the values of the integrator components or upon any changes in them. The accuracy does depend on  $E_{REF}$ ; so, the reference voltage should be very precise.

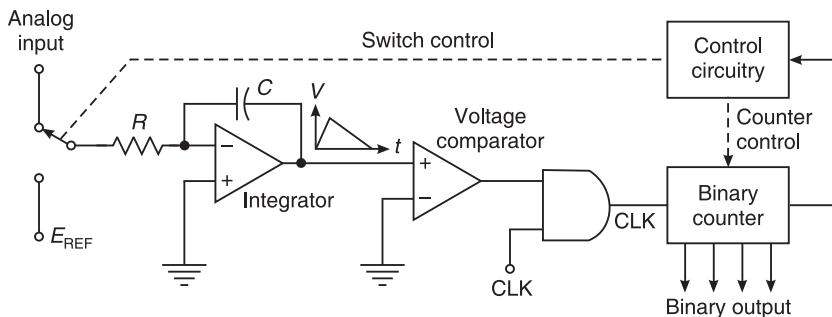
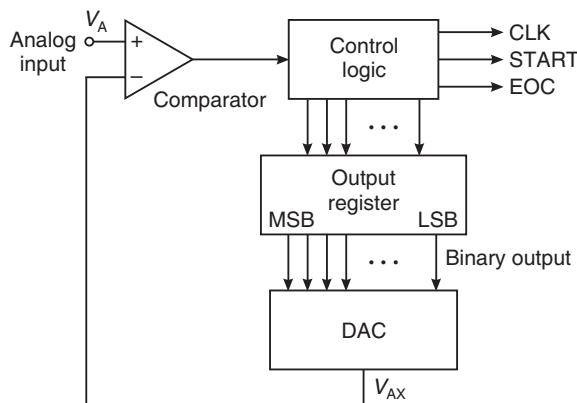


Figure 17.23 The dual-slope ADC.

## 17.12 THE SUCCESSIVE-APPROXIMATION TYPE ADC

The successive-approximation (SA) converter is one of the most widely used types of ADCs. It has a much shorter conversion time than the other types, with the exception of the flash type. It also has a fixed conversion time which is not dependent on the value of the analog input.

Figure 17.24 shows a basic block diagram of a 4-bit successive-approximation type ADC. It consists of a DAC, an output register, a comparator, and control circuitry or logic. The basic operation is as follows: The bits of the DAC are enabled one at a time, starting with the MSB. As each bit is enabled, the comparator produces an output that indicates whether the analog input voltage is greater or less than the output of the DAC  $V_{AX}$ . If the D/A output is greater than the analog input, the comparator output is LOW, causing the bit in the control register to reset. If the D/A output is greater than the analog input, the comparator output is HIGH, and the bit is retained in the control register.



**Figure 17.24** The successive-approximation type ADC.

The system enables the MSB first, then the next significant bit, and so on. After all the bits of the DAC have been tried, the conversion cycle is complete. The processing of each bit takes one clock cycle; so, the total conversion time for an  $N$ -bit SA-type ADC will be  $N$  clock cycles. That is,

$$t_c \text{ for SAC} = (N \times 1) \text{ clock cycles}$$

The conversion time will be the same regardless of the value of  $V_A$ . This is because the control logic has to process each bit to see whether a 1 is needed or not.

The method is best explained by an example. Let us assume that the output of the DAC ranges from 0 V to 15 V as its binary input ranges from 0000 to 1111, with 0000 producing 0 V, and 0001 producing 1 V, and so on. Suppose that the unknown analog input voltage  $V_A$  is 10.3 V. On the first clock pulse, the output register is loaded with 1000, which is converted by the DAC to 8 V. The voltage comparator determines that 8 V is less than the analog input (10.3 V); so, the control logic retains that bit. On the next clock pulse, the control circuitry causes the output register to be loaded with 1100. The output of the DAC is now 12 V, which the comparator determines as greater than the analog input. Therefore, the comparator output goes LOW. The control logic clears that bit; so, the output goes back to 1000. On the next clock pulse, the control circuitry causes the output register to be loaded with 1010. The output of the DAC is now 10 V, which the comparator determines as less than the analog input. Thus, on the next clock pulse, the control

logic causes the output register to be loaded with 1011. The output of the DAC is now 11 V, which the comparator determines as greater than the analog input; so, the control logic clears that bit. Now the output of the ADC is 1010 which is the nearest integer value to the input (10.3 V).

At this point, all of the register bits have been processed, the conversion is complete and the control logic activates its EOC output to signal that the digital equivalent of  $V_A$  is now in the output register.

**EXAMPLE 17.17** Compare the maximum conversion periods of an 8-bit digital ramp ADC and an 8-bit successive approximation ADC if both utilize a 1 MHz clock frequency.

**Solution**

For the digital-ramp converter, the maximum conversion time is

$$(2^N - 1) \times (\text{1 clock cycle}) = 255 \times 1 \mu\text{s} = 255 \mu\text{s}$$

For an 8-bit successive-approximation converter, the conversion time is always 8 clock periods, i.e.  $8 \times 1 \mu\text{s} = 8 \mu\text{s}$ .

Thus, the successive-approximation conversion is about 30 times faster than the digital-ramp conversion.

**EXAMPLE 17.18** An 8-bit SAC has a resolution of 30 mV. What will its digital output be for an analog input of 2.86 V?

**Solution**

Since,  $2.86 \text{ V}/30 \text{ mV} = 95.3$ , the step 95 would produce 2.85 V and step 96 would produce 2.88 V. The SAC always produces a final output, that is, at the step below the analog input. Therefore, for the case of  $V_A = 2.86 \text{ V}$ , the digital result would be  $95_{10} = 01011111_2$ .

### 17.12.1 A Specific A/D Converter

The ADC 0801 is an example of a successive-approximation type analog-to-digital converter. The pin diagram is shown in Figure 17.25. This device operates from a + 5 V supply, and has a resolution of 8 bits with a conversion time of 100  $\mu\text{s}$ . Also, it has guaranteed monotonicity and an on-chip clock generator. The data outputs are tri-stated so that it can be interfaced with a microprocessor bus system. The two analog inputs are  $V_{IN+}$  and  $V_{IN-}$ .

**$\overline{CS}$  (Chip Select):** This input has to be in active-LOW state, for  $\overline{RD}$  and  $\overline{WR}$  inputs to have any effect. With  $\overline{CS}$  HIGH, the digital outputs are in the Hi-Z state and no conversion takes place.

**$\overline{RD}$  (Output Enable):** This input is used to enable the digital output buffers. With  $\overline{CS} = \overline{RD} = \text{LOW}$ , the digital output pins have logic levels representing the results of the last A/D conversion.

**$\overline{WR}$  (Start Conversion):** A LOW pulse is applied to this input to signal the start of a new conversion.

**$\overline{INTR}$  (End of Conversion):** This output signal will go HIGH at the start of a conversion and return LOW to indicate the end of the conversion.

**$V_{ref}/2$ :** This is an optional input that can be used to reduce the internal reference voltage and thereby change the analog input range that the converter can handle.

**$CLK OUT$ :** A resistor is connected to this pin to use the internal clock. The clock signal appears on this pin.

**CLK IN:** It is used for the external clock input, or for the capacitor connection when the internal clock is used.

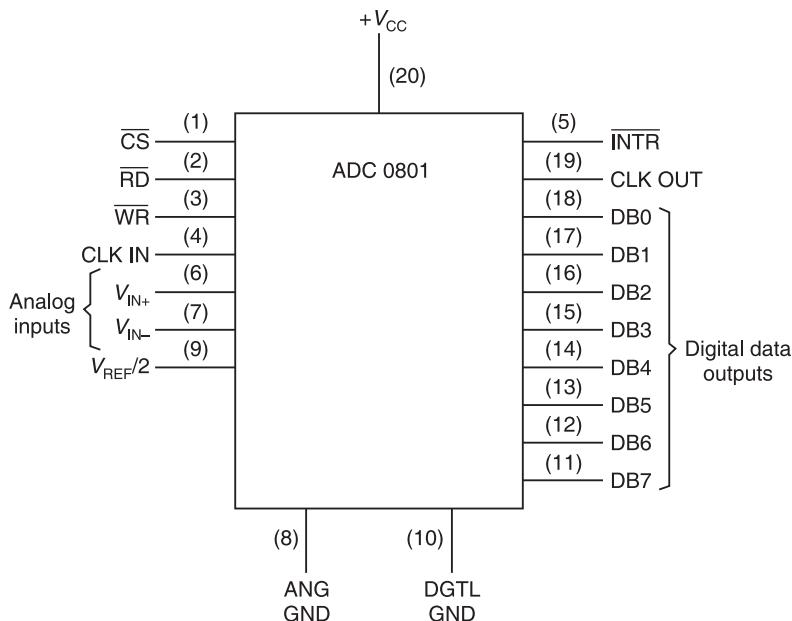


Figure 17.25 Pin configuration of the ADC 0801.

### 17.12.2 Voltage-to-Frequency ADC

The voltage-to-frequency ADC is simpler than other ADCs, because it does not use a DAC. Instead, it uses a linear voltage controlled oscillator (VCO) that produces an output frequency proportional to its input voltage. The analog voltage is applied to the VCO to generate an output frequency. This frequency is fed to a counter, to be counted for a fixed time interval. The final count is proportional to the value of the analog input.

To illustrate this, suppose the VCO generates a frequency of 5 kHz for each volt of input (i.e. 1 V produces 5 kHz, 1.5 V produces 7.5 kHz, 2.6 V produces 13 kHz). If the analog input is 4.65 V, then the VCO output will be a 23.25 kHz signal that clocks a counter for, say, 10 ms. After the 10 ms counting interval, the counter will hold the count of 232.

Although this is a simple method of conversion, it is difficult to achieve a high degree of accuracy because of the difficulty in designing VCOs with accuracies better than 0.1 per cent.

One of the main applications of this type of converter is in noisy industrial environments, where small analog signals have to be transmitted from transducer circuits to a control computer. The small analog signals can get drastically affected by noise, if they are transmitted directly to the control computer. A better approach is to feed the analog signal to a VCO, which generates a digital signal whose output frequency changes according to the analog input. The digital signal is transmitted to the computer and will be much less affected by noise. Circuitry in the control computer will count the digital pulses to produce a digital value, equivalent to the original analog input.

## SHORT QUESTIONS AND ANSWERS

**1.** What do you mean by analog quantity?

**A.** An analog quantity is one that can take on any value over a continuous range of values. It represents an exact value.

**2.** What do you mean by digital quantity?

**A.** A digital quantity is one which can take only discrete values. The value is expressed in a digital code such as a binary or a BCD number.

**3.** What is the need of A/D and D/A converters in controlling a process?

**A.** When a physical process is monitored or controlled by a digital system such as a digital computer, the physical variables are first converted into electrical signals using transducers, and then these electrical analog signals are converted into digital signals using ADCs. These digital signals are processed by a digital computer and the output of the digital computer is converted into analog signals using DACs. The output of the DAC is modified by an actuator and the output of the actuator is applied as the control variable.

**4.** What is the function of ADCs and DACs?

**A.** ADCs and DACs function as interfaces between a completely digital system such as a digital computer and the analog world.

**5.** What is D/A conversion?

**A.** Basically D/A conversion is the process of converting a value represented in digital code such as straight binary or BCD into a voltage or current which is proportional to the digital value.

**6.** Is the output of DAC a true analog quantity?

**A.** No. The output of DAC is not a true analog quantity: It is a pseudo analog quantity.

**7.** Define the following parameters of DACs:

- |                    |                           |
|--------------------|---------------------------|
| (a) Resolution     | (b) Accuracy              |
| (c) Monotonicity   | (d) Settling time         |
| (e) Offset voltage | (f) Percentage resolution |
| (g) Step size      | (h) Linearity error       |

**A.** (a) *Resolution:* The resolution of a DAC is defined as the smallest change that can occur in an analog output as a result of a change in the digital input. It is also defined as the reciprocal of the number of discrete steps in the full scale output of the DAC.

(b) *Accuracy:* The accuracy of a DAC is the nearness of the DAC's output to its expected (ideal) value.

(c) *Monotonicity:* The increase in the output of a DAC as the input increases is called monotonicity.

(d) *Settling time:* The settling time of a DAC is specified as the total time between the instant when the digital input changes and the time that the output enters a specified error band for the last time, usually  $\pm 1/2$  LSP around the final value after the change in digital input.

(e) *Offset voltage:* The offset voltage of a DAC is the small output voltage present when the binary input is zero.

(f) *Percentage resolution:* The percentage resolution of a DAC is the percentage of the full scale output.

$$\% \text{ resolution} = \frac{\text{Step size}}{\text{Full scale}} \times 100\% = \frac{1}{\text{Total number of steps}} \times 100\%$$

- (g) *Step size:* The step size or resolution is the size of the jumps in the staircase waveform of a DAC. It is the same as the proportionality factor in the DAC I/O relationship.
- (h) *Linearity error:* The linearity error is the maximum deviation of the analog output from the ideal output.
- 8.** How many different output voltages can an 8-bit DAC produce?  
**A.** An 8-bit DAC can produce 100 different output voltages.
- 9.** What is the advantage of smaller (finer) resolution?  
**A.** The advantage of smaller (finer) resolution is that the smaller the resolution, the more the output looks like an analog quantity.
- 10.** Distinguish between unipolar and bipolar DACs.  
**A.** Unipolar DACs are DACs whose output is a positive voltage or current. Bipolar DACs are DACs designed to produce both positive and negative voltages or currents.
- 11.** Which is the most popular DAC?  
**A.** The  $R-2R$  ladder type DAC is the most popular DAC.
- 12.** What is the advantage of the  $R-2R$  ladder DAC over the weighted resistor type DAC?  
**A.** The principal advantage of  $R-2R$  ladder type DAC over the weighted resistor type DAC is: In  $R-2R$  ladder DAC, resistors of only two values are required. Therefore standard resistors can be used. The main disadvantage of the weighted resistor type DAC is that a different valued precision resistor must be used for each bit position of the digital input.
- 13.** A certain 6-bit DAC uses binary weighted resistors. If the MSB resistor is  $40\text{ k}\Omega$ , what is the LSB resistor?  
**A.**  $1280\text{ k}\Omega$ .
- 14.** Why are current DACs generally faster than voltage DACs?  
**A.** Current DACs are generally faster than voltage DACs because the current can be switched in and out of the circuit faster than the voltages.
- 15.** Name two switched voltage type DACs?  
**A.** The  $R-2R$  ladder network DAC and the weighted resistor DAC are switched voltage type DACs.
- 16.** List the various types of DACs and ADCs. Name the most widely used DAC.  
**A.** The various types of DACs are: The  $R-2R$  ladder network type DAC, the weighted resistor type DAC, the switched-current-source type DAC and the switched-capacitor type DAC. The various types of ADCs are: the counter type ADC, the tracking type ADC, the flash-type ADC, the dual-slope type ADC, the successive approximation-type ADC and the voltage to frequency ADC.
- 17.** What is the advantage of switched capacitor technology?  
**A.** The principal advantage of switched capacitor technology is that small capacitors of the order of a few picofarads can be constructed in the ICs to perform the function of the much larger capacitors that are normally needed in low frequency analog circuits.
- 18.** Which is the simplest type ADC? What is its other name?  
**A.** The counter-type ADC is the simplest type of ADC. It is also called a digital ramp ADC.
- 19.** What is quantization?  
**A.** The process of approximation used in digitizing samples is called quantization.
- 20.** Give one advantage and one disadvantage of a counter-type ADC.  
**A.** The main advantage of counter-type ADC is that it is the simplest ADC. The principal disadvantage of counter-type ADC is that, the conversion time depends on the magnitude of the analog input.

- 21.** Why does the conversion time increase with the value of the analog input voltage in a counter type ADC?
- A. In a counter-type ADC, the conversion time increases with the value of the analog input voltage because the larger the input, the more will be the number of clock pulses that must pass to reach the proper count.
- 22.** Why is the tracking-type ADC faster than the counter type ADC?
- A. The counter-type ADC is slow because the counter resets itself after each conversion. The tracking-type ADC uses an up-down counter and is faster than the counter-type ADC because the counter is not reset after each sample, but tends to track the analog input, i.e. counts up or down from its last count to its new count.
- 23.** Why is the name tracking-type ADC?
- A. Since the count more or less keeps up with the changing analog input, the ADC is called the tracking-type ADC.
- 24.** What is the other name of the tracking-type ADC?
- A. The tracking-type ADC is also called the continuous conversion type ADC.
- 25.** Which is the fastest ADC and why?
- A. The flash-type ADC is the fastest type of ADC, because it uses no clock signal and there is no timing or sequencing period. The conversion takes place continuously.
- 26.** Does a flash-type ADC contain a DAC?
- A. No. The flash-type ADC does not contain a DAC, but a modified flash-type ADC contains a DAC.
- 27.** Which is the most expensive ADC?
- A. The flash-type ADC is the most expensive ADC.
- 28.** What is the main disadvantage of a flash-type ADC?
- A. The main disadvantage of a flash-type ADC is it is complex. An  $n$ -bit converter requires  $2^n - 1$  comparators,  $2^n$  resistors and a priority encoder.
- 29.** Give two advantages and one disadvantage of a dual-slope ADC.
- A. The advantages of dual-slope ADC are: (a) it is relatively inexpensive and (b) it is less sensitive to variations in its component values.
- 30.** What are the major applications of dual-slope type ADC?
- A. The major applications of dual-slope type ADC are in digital voltmeters, multimeters etc., where slow conversion is not a problem.
- 31.** Which ADC has a fixed conversion time?
- A. The successive approximation type ADC has a fixed conversion time.
- 32.** What is the main advantage and disadvantage of a successive approximation type ADC over a digital ramp ADC?
- A. The advantage of SA type ADC over a digital ramp ADC is that the conversion time of SA type ADC is fixed and low compared to the conversion time of digital ramp ADC, but the disadvantage is it is not as simple as that.
- 33.** Which is the most widely used ADC?
- A. The successive approximation type ADC is the most widely used type of ADC.
- 34.** Name the three types of ADCs that do not use a DAC.
- A. The three types of ADCs that do not use a DAC are: (a) voltage-to-frequency ADC, (b) The flash type ADC, and (c) the dual-slope type ADC.

**REVIEW QUESTION**

1. With the help of neat diagrams explain the working of the following DACs and ADCs.
 

(a) R-2R ladder network type DAC	(b) Weighted-resistor type DAC
(c) Current-switching type DAC	(d) Switching-capacitor type DAC
(e) Counter-type ADC	(f) Tracking-type ADC
(g) Flash-type ADC	(h) Dual-slope ADC
(i) Successive-approximation type ADC.	

**FILL IN THE BLANKS**

1. Digital processing of analog signals requires \_\_\_\_\_ and \_\_\_\_\_ converters.
2. \_\_\_\_\_ and \_\_\_\_\_ function as interfaces between a completely digital system, such as a digital computer and the analog world.
3. The output of DAC is not a \_\_\_\_\_ quantity. It is a \_\_\_\_\_ quantity.
4. The \_\_\_\_\_ of a DAC is equal to the step size.
5. The \_\_\_\_\_ of a DAC is the small voltage that appears at its output, when its binary input has all 0s.
6. The output of a \_\_\_\_\_ DAC increases when its binary input is incremented.
7. The \_\_\_\_\_ of a DAC is the same as the proportionality factor in the DAC I/O relationship.
8. The \_\_\_\_\_ is the maximum deviation of the DACs output from its expected value.
9. The \_\_\_\_\_ DAC is the most popular DAC.
10. \_\_\_\_\_ DACs are faster than \_\_\_\_\_ DACs.
11. The R-2R ladder network DAC and the weighted resistor DAC are \_\_\_\_\_ type DACs.
12. The process of approximation used in digitizing samples is called \_\_\_\_\_.
13. The \_\_\_\_\_ type ADC is the simplest and the \_\_\_\_\_ type the fastest.
14. The \_\_\_\_\_ type ADC is also called the continuous conversion type ADC.
15. The \_\_\_\_\_ type ADC is the most expensive ADC.
16. The \_\_\_\_\_ ADC is the slowest type, but it is cheap.
17. The \_\_\_\_\_ type ADC is the most widely used type of ADC.
18. \_\_\_\_\_ DACs are generally slower than \_\_\_\_\_ DACs.
19. The ADCs which do not contain a DAC are:
 

(a) _____ type DAC,	(b) _____ type DAC
(c) _____ type DAC.	
20. The \_\_\_\_\_ type ADC has fixed conversion time.
21. The \_\_\_\_\_ type ADCs are used in digital voltmeters and multimeters.
22. The counter-type ADC is also called a \_\_\_\_\_ ADC.

### OBJECTIVE TYPE QUESTIONS

- 1.** The most popular DAC is
 

(a) $R-2R$ ladder type	(b) weighted-resistor type
(c) switched-current source type	(d) switched-capacitor type
- 2.** The simplest type of ADC is
 

(a) counter-type	(b) flash-type
(c) successive-approximation type	(d) dual-slope type
- 3.** The fastest ADC is
 

(a) counter-type	(b) flash-type
(c) successive-approximation type	(d) dual-slope type
- 4.** The most expensive ADC is
 

(a) counter-type	(b) flash-type
(c) successive-approximation type	(d) dual-slope type
- 5.** The slowest ADC is
 

(a) counter-type	(b) flash-type
(c) successive-approximation type	(d) dual-slope type
- 6.** The most widely used type of ADC is
 

(a) counter-type	(b) flash-type
(c) successive-approximation type	(d) dual-slope type
- 7.** The ADC which has fixed conversion time is
 

(a) counter-type	(b) flash-type
(c) successive-approximation type	(d) dual-slope type
- 8.** A certain 4-bit DAC uses binary weighted resistors. If the MSB resistor is  $100\text{ k}\Omega$ , the LSB resistor will be
 

(a) $400\text{ k}\Omega$	(b) $25\text{ k}\Omega$	(c) $800\text{ k}\Omega$	(d) $12.5\text{ k}\Omega$
--------------------------	-------------------------	--------------------------	---------------------------
- 9.** The ADC used in digital voltmeters and multimeters is
 

(a) counter-type	(b) flash-type
(c) successive-approximation type	(d) dual-slope type
- 10.** In which type of ADC, the conversion time depends on the magnitude of the analog input?
 

(a) counter-type	(b) flash-type
(c) successive-approximation type	(d) dual-slope type

### PROBLEMS

- 17.1** How many bits are required for a DAC, so that its full-scale output is  $12.6\text{ V}$  and resolution  $200\text{ mV}$ ?
- 17.2** A 5-bit DAC produces an output of  $0.1\text{ V}$  for a digital input of 00001. What is the full-scale output? Find the output for an input of 10101.

**942 FUNDAMENTALS OF DIGITAL CIRCUITS**

- 17.3** The logic levels used in a 6-bit  $R$ - $2R$  ladder DAC are: 1 = 5 V and 0 = 0 V. Find the output voltage for inputs (a) 010110, and (b) 101011.
- 17.4** The logic levels used in an 8-bit  $R$ - $2R$  ladder DAC are: 0 = 0 V and 1 = +5 V. What is the binary input when the analog output is 4 V?
- 17.5** Design a 4-bit weighted-resistor DAC whose full-scale output voltage is -12 V. Logic levels are 1 = +5 V and 0 = 0 V. What is the output voltage when the input is 1011?
- 17.6** A 6-bit switched current source DAC has an output current of 20 mA for a digital input of 101100. What will be the output current for an input of 010110?
- 17.7** In the switched current-source DAC shown in Figure 17.13,  $R = 10 \text{ k}\Omega$  and  $E_{\text{REF}} = 15 \text{ V}$ . Find the current in each  $2R$  resistor when it is connected to  $E_{\text{REF}}$ .
- 17.8** An 8-bit switched current-source DAC of the design shown in Figure 17.13, has  $R = 5 \text{ k}\Omega$  and  $E_{\text{REF}} = 20 \text{ V}$ . Find the total current  $I_T$  delivered to the amplifier and the output voltage when the input is 01110100.
- 17.9** A 4-bit switched capacitor type DAC of the type shown in Figure 17.14 has  $E_{\text{REF}} = 10 \text{ V}$ . Find the output voltage for a digital input of 1011.
- 17.10** A 4-bit counter-type ADC has a full scale reading of 10 V. What is the quantization error of the ADC.
- 17.11** The maximum conversion time of a tracking-type ADC is 100 ns. At what frequency is it clocked?
- 17.12** A flash-type 5-bit ADC has a reference voltage of 20 V. How many voltage comparators does it have? How many resistors does it have? What is the increment between the voltages applied to the comparators?
- 17.13** The resolution of a 12-bit ADC is 10 mV. What is its full-scale range?
- 17.14** The frequency components of the analog input to an ADC range from 50 Hz to 50 kHz. What is the maximum total conversion time that the converter can have?
- 17.15** An 8-bit SA type ADC has a resolution of 15 mV. What will its digital output be for an analog input of 2.65 V?

# 18

## MEMORIES

### 18.1 THE ROLE OF MEMORY IN A COMPUTER SYSTEM

#### 18.1.1 Program and Data Memory

Basically, memory is a means for storing data or information in the form of binary words. It is made up of storage locations in which numeric or alphanumeric information or programs (sets of instructions that a computer executes to achieve a desired result) may be stored. Memory used to store data is called *data memory*, and memory used to store programs is called *program memory*. Small special-purpose computer systems may have little or no data memory, whereas a large portion of memory of a general-purpose computer system is usually reserved for data storage.

Computers which store programs in their memory are called *stored-program type computers*. Virtually, all modern computers are of the stored-program type. In these computers, programs are stored as a set of *machine language instructions*, in binary codes. Each memory location is identified by an *address*. The number of storage locations can vary from a few in some memories to millions in others.

Each storage location can accommodate one or more bits. Generally, the total number of bits a memory can store is its *capacity*. Sometimes the capacity is specified in terms of bytes. Memories are made up of storage elements (FFs or capacitors in semiconductor memories and magnetic domains in magnetic memories), each of which stores one bit of data. A storage element is called a *cell*.

Magnetic tapes and magnetic disks are popular mass storage devices that cost less per bit than the internal memory devices. A newer entry into the mass memory category is the magnetic

bubble memory (MBM), a semiconductor memory that uses magnetic principles to store millions of bits on one chip. The MBM is relatively slow and cannot be used as an internal memory.

### 18.1.2 Main and Peripheral Memory

Computer memories may also be classified as *main* or *peripheral*. The main memory is an internal part of the computer and is very fast. The peripheral memory also called the *auxiliary* memory is a typically add-on memory with very large storage capacity, but it is much slower than the main memory. It often serves as the data memory for storing very large quantities of data. The main memory that serves as the program memory is in constant communication with the CPU during program execution. The program to be currently executed and any data used by the program are stored in the main memory. Semiconductor memories are well-suited as the main memory because of their high speed of operations.

A semiconductor memory is typically constructed from semiconductor IC devices, whereas the peripheral memory consists of magnetic tapes or disks. In older computers, the main memory was constructed from tiny electromagnets called magnetic cores and the term *core memory* or simply *core* was used synonymously with the main memory. Mass storage refers to memory that is external to the main computer (magnetic tapes and disks) and has the capacity to store millions of bits of data without the need for electrical power. Mass memory is normally much slower than the main memory and is used to store information (programs, data, etc.) not currently being used by the computer. The required information is transferred to the main memory from the mass memory when the computer actually needs it.

## 18.2 MEMORY TYPES AND TERMINOLOGY

### 18.2.1 Memory Organization and Operation

All memory, regardless of its type or use, consists of locations for storing binary information or bits. Each location is identified by an *address*. A *word* is the fundamental group of bits used to represent one entity of information such as one numerical value. The word size—the number of bits in a word—varies among computer systems and may range from 4 to 64 or more bits. The word size is usually expressed as a certain number of bytes. For example, a 16-bit word is 2 bytes.

A memory location is thus a set of devices capable of storing one word. For example, each memory location in an 8-bit microcomputer (one that uses 8-bit words) might consist of eight latches. Each *latch* stores one bit of a word, and is referred to as a *cell*. The capacity or size of a memory is the total number of bits or bytes or words that it can store. For convenience, the size of a memory is expressed as a multiple of  $2^{10} = 1024$ , which is abbreviated as K. For example, a memory of size  $2^{11} = 2048$  is said to be 2 K. A memory of size  $2^{14}$  (16,384) is 16 K, and a memory of size  $2^{16}$  (65,536) is 64 K.

Every memory system requires several different types of input and output lines to perform the following functions.

1. Select the address in memory that is to be accessed for a read or write operation.
2. Select either a read or a write operation to be performed.
3. Supply the input data to be stored in memory during a write operation.

4. Hold the output data coming from memory during a read operation.
5. Enable (or disable) the memory, so that it will (or will not) respond to the address inputs and read/write command.

Figure 18.1a illustrates these basic functions in a simplified diagram of a  $32 \times 4$  memory that stores 32 4-bit words. Since the word size is 4-bits, there are four data input lines  $I_0$  to  $I_3$  and four data output lines  $O_0$  to  $O_3$ . During a write operation, the data to be stored in memory have to be applied to the data input lines. During a read operation, the word being read from memory appears at the data output lines.

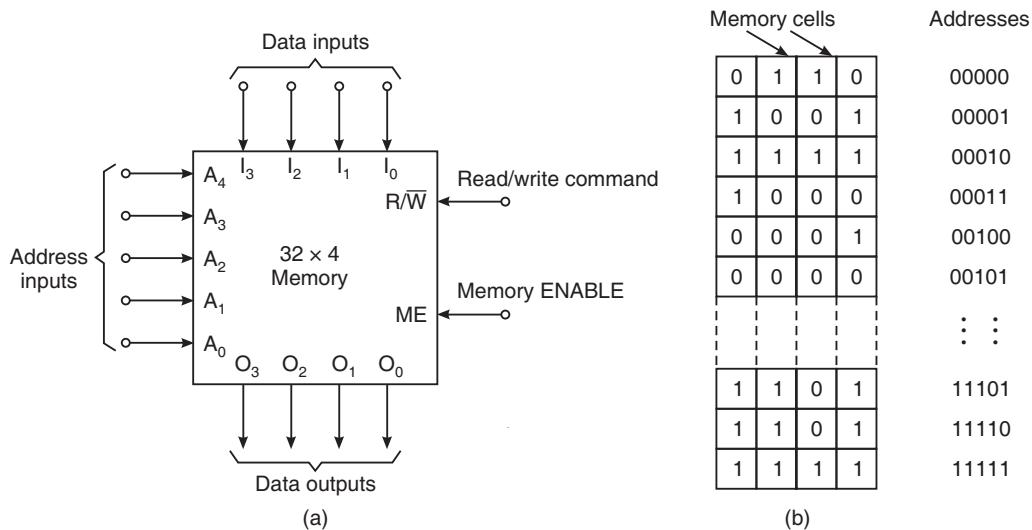


Figure 18.1 Diagram of a  $32 \times 4$  memory and arrangement of memory cells.

### Address inputs

Since this memory stores 32 words, it has 32 different storage locations and, therefore, 32 different binary addresses ranging from 00000 to 11111 (0 to 31 decimal). Thus, there are five address inputs  $A_0$  to  $A_4$ . To access one of the memory locations for a read or write operation, the 5-bit address code for that particular location is applied to the address inputs. In general,  $N$  address inputs are required for a memory that has a capacity of  $2^N$  words.

We can visualize the memory of Figure 18.1a as an arrangement of 32 registers with each register holding a 4-bit word as illustrated in Figure 18.1b. Each address location is shown containing four memory cells that hold 1s and 0s to make up the data word stored at that location.

### The R/W input

The read/write ( $R/\bar{W}$ ) input line determines the memory operation that would take place. Some memory systems use two separate inputs, one for read and one for write. When a single  $R/\bar{W}$  input is used, the read operation takes place for  $R/\bar{W} = 1$  and the write operation takes place for  $R/\bar{W} = 0$ .

### Memory ENABLE

Many memory systems have some means of completely disabling all or part of the memory so that it does not respond to the other inputs. This is represented in Figure 18.1a as the memory ENABLE

input, although it can have different names in the various memory systems. Here it is shown as an active-HIGH input that enables the memory to operate normally when it is kept HIGH. A LOW on this input disables the memory, preventing it to respond to address and R/W inputs. This type of input is useful when several memory modules are combined to form a larger memory.

**EXAMPLE 18.1** A certain memory has a capacity of  $8K \times 16$ .

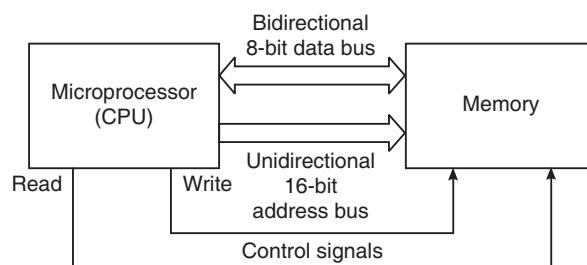
- (a) How many data input and data output lines does it have?
- (b) How many address lines does it have?
- (c) What is its capacity in bytes?

**Solution**

- (a) Since the word size is 16 bits, data input and data output will be 16 lines each.
- (b) The memory stores  $8K = 8 \times 1024 = 8192$  words. Thus, there are 8192 memory addresses.  
Since  $8192 = 2^{13}$ , it requires 13 address lines.
- (c) A byte is 8-bits. This memory therefore, has a capacity of 16K bytes.

### 18.2.2 Reading and Writing

The process of storing data in memory is called *writing* in memory. Retrieving data from memory is called *reading* memory. Figure 18.2 illustrates how reading from and writing into memory is accomplished in an 8-bit microprocessor system. The microprocessor serves as the central processing unit (CPU) for the computer. It contains an arithmetic/logic unit (ALU), numerous registers, and logic circuitry, that it uses to perform read and write operations as well as to execute programs. Note the control signals labelled *read* and *write*. The CPU activates these when a read or a write operation is to be performed. The wide two-headed arrow represents an 8-bit data bus consisting of eight lines on which data bits  $D_0$  through  $D_7$  are transmitted. It is called a *bidirectional data bus*, because words can be transmitted from the CPU to memory (when writing) or from memory to the CPU (when reading). The unidirectional address bus is the set of lines over which the CPU transmits the address bits corresponding to the memory address to be read or written into. In the example shown, the address bus is a 16-bit bus ( $A_0$  through  $A_{15}$ ) meaning that the CPU can access (read or write into) up to  $2^{16} = 65,536$  different memory addresses. The following is a typical sequence of events, during which a byte is read from one memory location and written into another.



**Figure 18.2** Reading and writing of data in a microcomputer system.

1. The CPU activates the *read* control and transmits the 16-bit address, say,  $000A_{16}$  to memory via the address bus.

2. As a result of step 1, the 8-bit word stored in address  $000A_{16}$ , say,  $45_{16}$  is placed on the data bus and transmitted to the CPU.
3. The CPU activates the *write* control and transmits the 16-bit address, say,  $000B_{16}$  to memory. It also transmits the 8-bit data word  $45_{16}$  to memory via the data lines.
4. As a result of step 3, the data word  $45_{16}$  is stored at address  $000B_{16}$  (the original contents of that address are lost).

### 18.2.3 RAMs, ROMs and PROMs

The type of memory we have discussed is called the read/write memory (RWM) because it can be accessed for both these kinds of operations. On the other hand, read only memory (ROM) cannot be written into. It is, therefore, used for permanent storage of programs or data in dedicated applications.

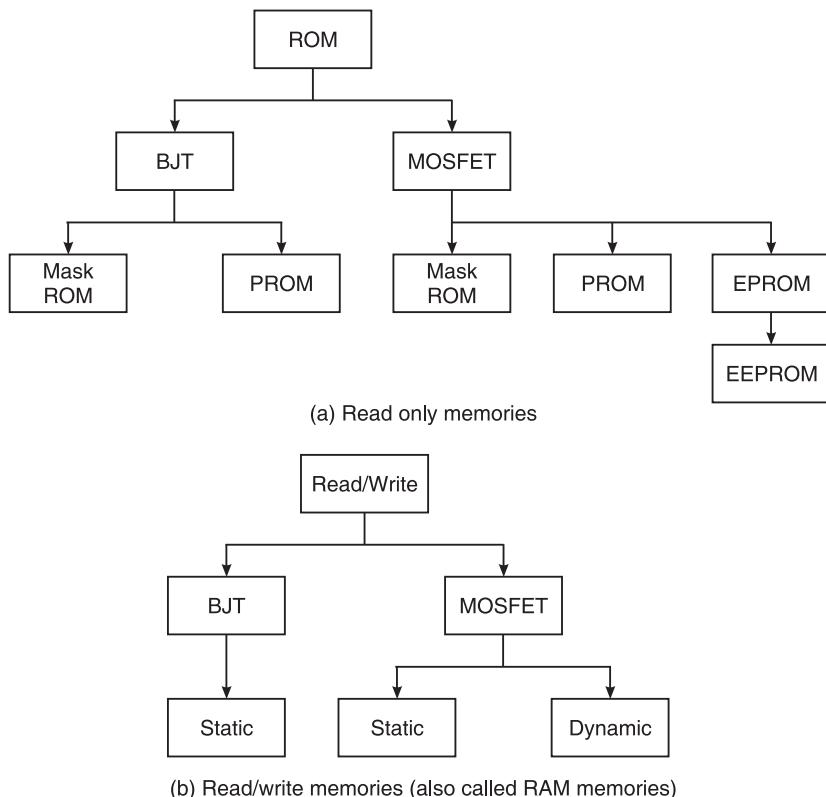
The data stored in RWMs constructed from semiconductor devices will be lost if power is removed. Such memory is said to be *volatile*. But ROM is non-volatile. Random access memory (RAM) is the memory whose memory locations can be accessed directly and immediately. By contrast, to access a memory location on a magnetic tape, it is necessary to wind or unwind the tape and go through a series of addresses before reaching the address desired. Therefore, a tape is called the *sequential access memory*. It is conventional to use the term RAM to mean read/write memory, in contrast to ROM. However, most ROMs are random access (to read only) in the sense we have described.

### 18.2.4 Constituents of Memories

We know that there are two principal types of transistors used in the manufacture of digital ICs. They are BJTs and MOSFETs. The IC memories are available in both of these technologies. However, not every type of memory we have discussed is available in bipolar technology.

Because MOSFETs are more easily manufactured than BJTs and occupy much less space in an IC chip, that technology has become more dominant in the production of large capacity integrated circuit memories. However, the BJT memory is considerably faster than the MOS memory and the BJT ICs are used in applications where high speed is more important than the large storage capacity. Memory circuits constructed with the I<sup>2</sup>L technology are also available. These feature higher speeds than those of the MOS memories and greater capacity per circuit than that of the BJT memories.

In Chapter 16, we discussed dynamic logic in which logic levels are preserved as charge (or absence of charge) on capacitances. Dynamic memory is constructed with this technology. Its principal advantages comprise a very large number of memory cells in a single circuit and very low power consumption. However, such circuitry must be periodically refreshed to replenish the stored charges. This requirement adds somewhat to the complexity of the systems that incorporate dynamic logic. Dynamic memory is available only in MOSFET circuits. To distinguish between the dynamic memory and the memory that utilizes conventional storage devices such as latches, the latter type is termed *static memory*. Static memory is available in both BJT and MOSFET technologies. To distinguish between static RAM and dynamic RAM, the terms SRAM and DRAM are sometimes used. Figure 18.3 summarizes the technologies used in the manufacture of IC ROMs and RAMs (read/write memories).



**Figure 18.3** Technologies used in the fabrication of IC memories.

### 18.2.5 Applications of ROMs

ROMs can be used in any application requiring non-volatile data storage, where the data rarely or never changes. We briefly describe below some of the most common application areas.

#### Microcomputer program storage (firmware)

At present, microcomputer firmware is the most widespread application of ROMs. Some personal and business microcomputers use ROMs to store their operating system programs and their language interpreters (e.g. BASIC), so that the computer can be used immediately after power is turned on. Products that include a microcomputer to control their operation use ROMs to store the control programs. Examples are, electronic games, electronic cash registers, electronic scales, and microcomputer-controlled automobile fuel injection systems. The microcomputer programs that are stored in ROMs are referred to as *firmware*, because they are not subject to change. Programs that are stored in RWMs are referred to as *software*, because they can be easily altered.

#### Bootstrap memory

Many microcomputers and most large computers do not have their operating system programs stored in ROMs. Instead, these programs are stored in external mass memory, usually the magnetic disk. How, then, do these computers know what to do when they are powered on? A relatively

small program, called the *bootstrap program*, is stored in a ROM (the term *bootstrap* comes from the idea of pulling oneself up by one's own boot straps). When the computer is powered on, it will execute the instructions that are in its bootstrap program. These instructions typically cause the CPU to initialize the system hardware. The bootstrap program then loads the operating system programs from mass storage (disk into its main internal memory). At that point, the computer begins executing the operating system program and is ready to respond to the user commands. This start-up process is often called 'booting up' the system.

### **Data tables**

ROMs are often used to store tables of data that do not change. Some examples are trigonometric tables (i.e. sine, cosine, etc.) and code conversion tables. Several standard ROM look-up tables are available with trigonometric functions.

### **Data converters**

The data converter circuit takes data expressed in one type of code and produces an output expressed in another type. Code conversion is needed, for example, when a computer is outputting data in straight binary code and it is required to convert it to BCD in order to display it on seven-segment LED readouts.

One of the easiest methods of code conversion uses a ROM programmed such that the application of a particular address (the old code) produces a data output that represents the equivalent one in the new code.

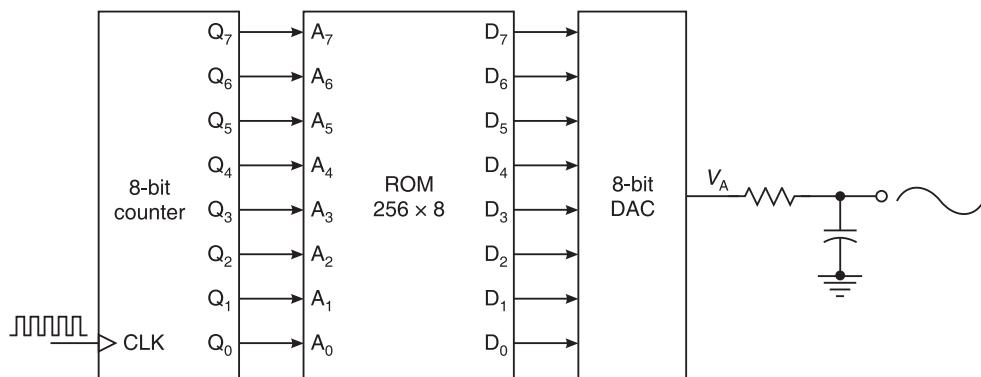
### **Character generators**

If you have ever looked closely at the alphanumeric characters (letters, numbers, etc.) printed on a computer's video display screen, you might have noticed that each is generally made up of a group of dots. Depending on the character being displayed, some dot positions are made bright while others are dark. Each character is made to fit into a pattern of dot positions, usually arranged as a  $5 \times 7$  or  $7 \times 9$  matrix. The pattern of dots for each character can be represented as a binary code (i.e. bright dot = 1, dark dot = 0).

A character generator ROM stores the dot pattern codes for each character at an address corresponding to the ASCII code for that character. Character generator ROMs are used extensively in any application that displays or prints out alphanumeric characters.

### **Function generator**

The function generator is a circuit that produces waveforms such as sine waves, saw tooth waves, triangular waves, and square waves. Figure 18.4 shows how a ROM look-up table and DAC are used to generate a sine wave output signal. The ROM stores 256 different 8-bit values, each one corresponding to a different waveform value, i.e. a different voltage point on the sine wave. The 8-bit counter is continuously pulsed by a clock signal to provide sequential address inputs to the ROM. As the counter cycles through the 256 different addresses, the ROM outputs the 256 data points to the DAC. The DAC output will be a waveform that steps through the 256 different analog voltage values corresponding to the data points. A low-pass filter smoothes out the steps in the DAC output to produce a smooth waveform. Circuits such as these are used in some commercial function generators. The same idea is used in some speech synthesizers where the digitized speech waveform values are stored in a ROM.



**Figure 18.4** Function generator using a ROM and DAC.

### 18.3 SEMICONDUCTOR RAMs

When the term RAM is used with semiconductor memories, it is usually taken to mean read/write memory (RWM) as opposed to ROM. The RAMs are used in computers for the temporary storage of programs and data. The contents of many RAM address locations are read from and written to as the computer executes a program. This requires fast read and write cycle times for the RAM so as not to slow down the computer operation. A major disadvantage of RAMs is that they are volatile and lose all stored information if power is interrupted or turned off. Some CMOS RAMs, however, use such small amounts of power in the standby mode (when no read or write operations take place) that they can be powered from batteries whenever the main power is interrupted. Of course, the main advantage of RAMs is that they can be written into and read from rapidly with equal ease.

Like the ROM, the RAM can also be thought of as consisting of a number of registers, each storing a single data word and having a unique address. The RAMs typically come with word capacities of 1K, 4K, 8K, 16K, 64K, 128K, 256K and 1024K, and word sizes of 1, 4, or 8-bits. The word capacity and word size can be expanded by combining several memory chips.

#### 18.3.1 Static RAMs (SRAMs)

The static RAM can store data as long as power is applied to the chip. Its memory cells are essentially flip-flops that will stay in a given state (store a bit) indefinitely, provided that power to the circuit is not interrupted. On the other hand, dynamic RAMs store data as charges on capacitors. With dynamic RAMs, the stored data will gradually disappear because of capacitor discharge, therefore, it is necessary to periodically refresh the data (i.e. recharge the capacitors).

Static RAMs (SRAMs) are available both in bipolar and MOS technologies, although the vast majority of applications use NMOS or CMOS RAMs. As stated earlier, the bipolars have the advantage of speed (though NMOS is gradually closing the gap), while the MOS devices have much greater capacities and lower power consumption. Figure 18.5 shows, for comparison, a typical bipolar static memory cell, a typical NMOS static memory cell, and a typical CMOS static memory cell. The bipolar cell contains two multi-emitter transistors and two resistors. The NMOS cell contains four *N*-channel MOSFETs. The CMOS cell requires two CMOS FETs.

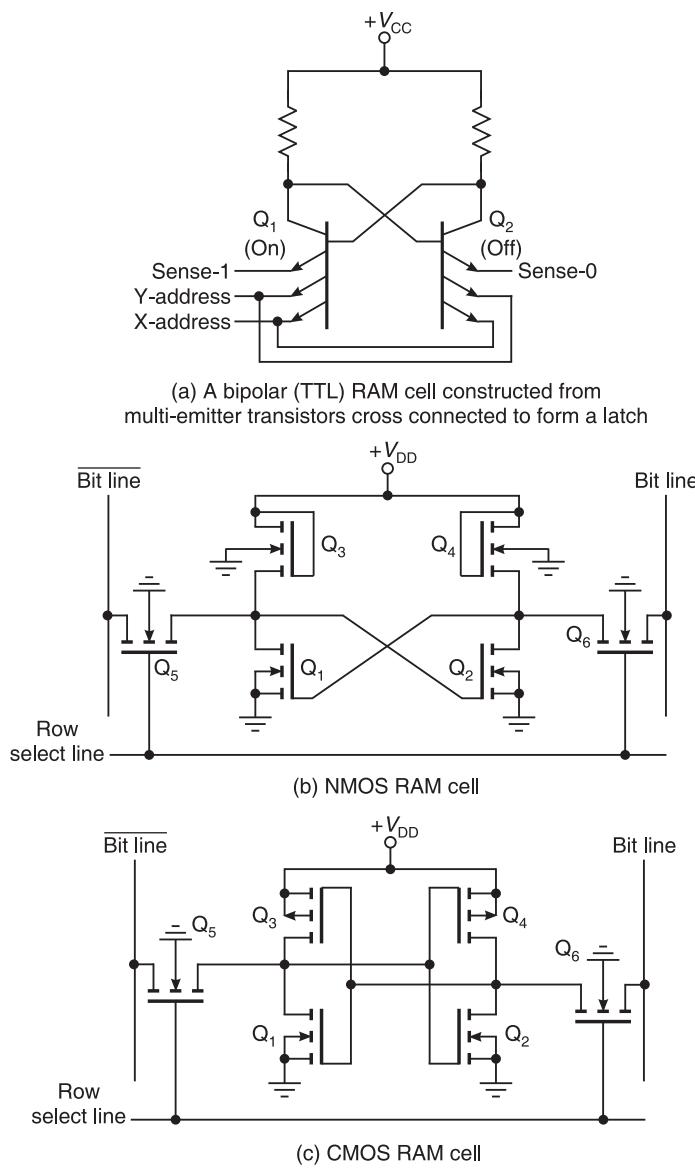


Figure 18.5 The bipolar, NMOS, and CMOS RAM cells.

The RAM ICs are most often used as the internal memory of a computer. The CPU continually performs read and write operations on this memory at a very fast rate constrained only by the limitations of the CPU. The memory chips that are interfaced to the CPU have to be, therefore, fast enough to respond to the CPU read and write commands.

### 18.3.2 ECL RAMs

The RAMs constructed using the ECL technology have the highest speed among those we have considered so far. The access time for bipolar TTL RAMs is of the order of 25–50 ns; for MOSFET

RAMs, it is of the order of 200–400 ns (although some new CMOS designs have speeds approaching those of the bipolar RAMs). The access time for ECL RAMs is of the order of 5–10 ns. The ECL RAMs consume considerable power and are not available in large sizes; they are therefore used where speed is the most important consideration. Examples include cache memory, where data from a slower memory can be stored for quick access by a CPU, and scratch pad memory used for storage of the intermediate results of computations. In these applications, data must be stored and retrieved very rapidly in order not to delay the high-speed computations performed by an ALU.

The ECL RAMs typically have open-emitter outputs to facilitate expansion and wire-ORing. They also have separate pins for input and output data, a feature that can be used to reduce delays between the read and write operations; new data to be written can be applied while the old data is still being read. As an illustration of the varieties available, the National Semiconductor line of ECL RAMs ranges from  $256 \times 1$  to  $1\text{K} \times 4$  and  $4\text{K} \times 1$ . Note that, these sizes, while small compared to MOSFET RAMs, are considerably greater than those of the TTL RAMs.

### 18.3.3 Dynamic RAMs (DRAMs)

Dynamic RAMs are fabricated using only MOS technology and they are noted for their high capacity, low power requirement and moderate speed. The need for *refreshing* is a disadvantage of DRAMs as it adds complexity to the memory system design. Up till recently, the system designers had to include additional circuitry to implement the memory refresh operation during the time intervals when the memory was not being accessed for a read or write operation. Now, there are two alternatives available to designers to help neutralize this disadvantage. For relatively small memories ( $< 64\text{K}$  words), the integrated RAM (iRAM) provides a solution. An iRAM is an IC that includes the refresh circuitry on the same chip as the memory cell array. The result is a chip that externally operates just like a static RAM chip—you apply the address and collect the data—but internally uses a high density DRAM structure. The designer is not concerned with the memory refresh operation; it is done internally and automatically.

For larger memory systems ( $> 64\text{K}$ ), a more cost-effective approach uses LSI chips called *dynamic memory controllers* which contain all of the necessary logic for refreshing the DRAM chips that make up the system. This greatly reduces the amount of extra circuitry in a dynamic RAM system.

For applications, where speed and reduced complexity are more critical than space and power considerations, static RAMs are still the best. They are generally faster than DRAMs and require no refresh operation. They are simpler to design, but cannot compete with higher capacity and lower power requirements of DRAMs.

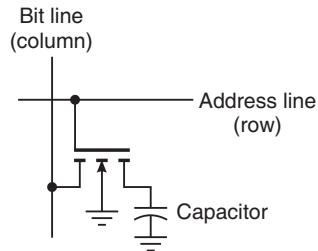
Because of their simple cell structure, the DRAMs typically have four times the density of SRAMs. This allows four times as much memory capacity to be placed on a single board, or alternatively, requires one-fourth as much board space for the same amount of memory. The cost per bit of DRAM storage is typically one-fifth to one-fourth that of SRAMs. A further cost saving is realized because the lower power requirement of a DRAM, typically one-sixth to one-half that of a SRAM, allows the use of smaller and less expensive power supplies.

The main applications of SRAMs are in areas where only small sizes of memory are needed (up to  $64\text{K}$ ) or where high speed is required. Many microprocessor controlled instruments and appliances have very small memory capacity requirements. Some instruments such as digital storage

oscilloscopes and logic analyzers require very high speed memory; for such applications, SRAMs are normally used.

The main internal memory of most personal computers is DRAM, because of its high capacity and low power consumption. These computers, however, sometimes use small amounts of SRAM as well, for functions requiring maximum speed such as video graphics and look-up tables.

Early DRAMs had four NMOS transistors per cell and relied on inherent gate capacitance of the transistors for storage of charge. The 4-transistor cell was complex, because transistors were needed to buffer and sense the tiny charges stored on the gate capacitance. A later improvement was a 3-transistor cell and the most current design is a single-transistor cell shown in Figure 18.6. The single-transistor cell is the ultimate in simplicity. The transistor serves as a transmission gate controlled by the address line. To read, the address line is made HIGH, turning on the transmission gate, and the capacitor voltage appears on the bit line. To write, the address line is again made HIGH, and the voltage on the bit lines charges or discharges the capacitor through the transmission gate. Read out is destructive, so every read operation must be followed by a write operation.



**Figure 18.6** The single-transistor dynamic memory cell.

#### 18.3.4 Address Multiplexing

In order to reduce the number of pins on high capacity DRAM chips, manufacturers utilize address multiplexing whereby each address input pin can accommodate two different address bits.

#### 18.3.5 DRAM Refreshing

DRAM cells have to be refreshed periodically as otherwise the stored data will be lost. In dynamic memories, refreshing can be accomplished by reading, since data must be automatically written back into the cells that are read. The maximum time between refresh cycles is typically 2, 4, or 8 ms in modern RAMs. Although it would be possible to read every cell in succession and thus refresh the entire memory during each refresh cycle, most dynamic memories are refreshed one entire row at a time in order to reduce the number of read operations that must take place to refresh the complete memory. In burst mode refreshing, each row of cells is refreshed in succession, with all normal memory operations suspended until all rows have been refreshed. Alternatively, row refreshing can be interspersed with other memory operations. In either case, refresh control circuitry is necessary to synchronize refresh cycles and to ensure that every row is refreshed within the specified time. Most manufacturers of dynamic memory ICs have developed special ICs to handle the refresh operation as well as the address multiplexing needed by the DRAM systems. These ICs are called *dynamic RAM controllers*. Some DRAMs have built-in refresh control circuitry and are said to be pseudo-dynamic (or semi-dynamic), because a user is not required to provide external hardware to accomplish the refresh task.

### 18.4 MEMORY EXPANSION

In most IC memory applications, the required memory capacity or word size cannot be satisfied by one memory chip. Instead, several memory chips are combined to provide the desired capacity

and the word size. The process of increasing the word size or capacity by combining a number of IC chips is called *memory expansion*. In a *bit-organized* memory, each IC stores 1-bit of each word. For example, in a bit-organized  $128K \times 8$  memory, the LSB of each word is stored in one  $128K \times 1$  circuit, the next LSB in another  $128K \times 1$  circuit, and so forth. A memory in which every bit of a word is stored in each circuit, is said to be *word organized*. For example, a word-organized  $128K \times 8$  memory might consist of sixteen  $8K \times 8$  circuits. In each of the above examples, memory expansion is required. In the bit-organized memory, we must interconnect eight  $128K \times 1$  circuits and in the word-organized memory, we must interconnect sixteen  $8K \times 8$  circuits. Some organizations require an expansion in word size as well as expansion in word capacity. For example, if we wish to construct a  $128K \times 8$  memory using  $16K \times 4$  circuits, we would need two of the latter for each  $16K$  of 8-bit words or a total of  $2 \times (128K/16K) = 16$  circuits.

Figure 18.7 shows how two  $16 \times 4$  memory RAM chips with common I/O lines are combined to produce a  $16 \times 8$  memory. Since each chip can store 16 4-bit words and since 16 8-bit words are to be stored, each chip is used to store half of each word. In other words, RAM 0 stores the four higher order bits of each of the 16 words, and RAM 1 stores the four lower order bits of each of the 16 words. A full 8-bit word is available at the RAM outputs connected to the data bus.

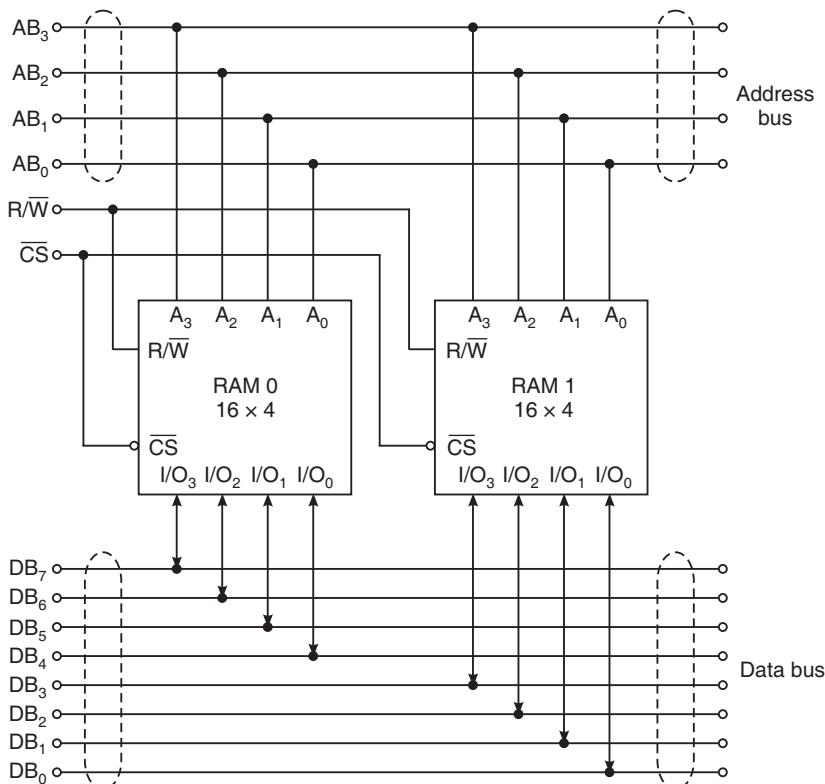


Figure 18.7 Combining two  $16 \times 4$  RAMs for a  $16 \times 8$  module.

Any one of the 16 words is selected by applying the appropriate address code to the four line address bus ( $AB_3, AB_2, AB_1$  and  $AB_0$ ). The address lines typically originate at the CPU. Note that,

each address bus line is connected to the corresponding address input of each chip. This means that once an address code is placed on the address bus, the same address code is applied to both chips such that the same location on each chip is accessed at the same time.

Once the address is selected, we can read or write at this address under the control of the common R/W and CS line. To read, R/W must be HIGH and CS must be LOW. This causes the RAM I/O lines to act as outputs. RAM 0 places its selected 4-bit word on the upper four data bus lines and RAM 1 places its selected 4-bit word on the lower four data bus lines. The data bus then contains the full selected 8-bit word which can now be transmitted to some other device (usually a register in the CPU).

To write, R/W = 0 and CS = 0, causes the RAM I/O lines to act as inputs. The 8-bit word to be written is placed on the data bus (usually by the CPU). The higher four bits will be written into the selected location of RAM 0 and the lower four bits will be written into RAM-1.

In essence, the combination of two RAM chips acts like a single  $16 \times 8$  memory chip. We would refer to this combination as a  $16 \times 8$  memory module.

The method illustrated here for increasing the word size applies to all of the memory devices we have discussed, including ROMs, PROMs, and static RAMs. If the memory circuits are dynamic, then the RAS and CAS inputs are also paralleled.

**EXAMPLE 18.2** The 2125A is a static RAM IC that has a circuitry of  $1K \times 1$ , one active-LOW chip select, and separate data input and output. Show how to combine several 2125A ICs to form a  $1K \times 8$  module.

#### **Solution**

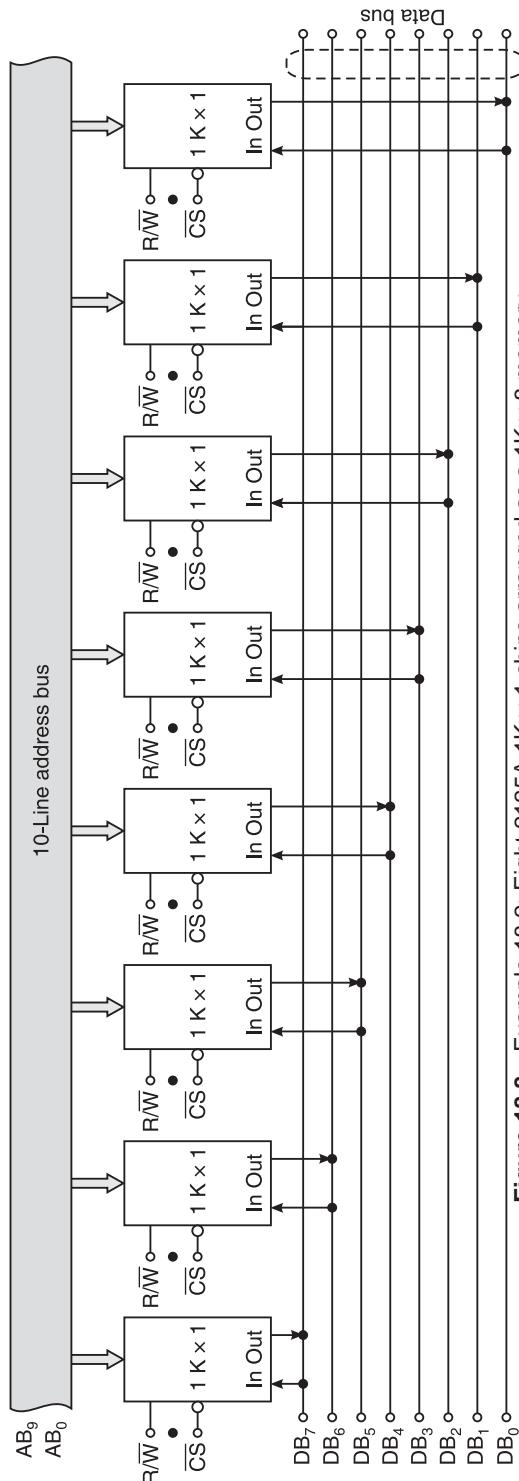
The arrangement is shown in Figure 18.8, where eight 2125A chips are used for a  $1K \times 8$  module. Each chip stores one of the bits of each of the 1024 8-bit words. Note that, all of the R/W and CS inputs are wired together and the 10-line address bus is connected to the address inputs of each chip. Also, note that since the 2125A has separate data in and data out pins, both of these pins of each chip are tied to the same data bus line.

Figure 18.9 shows how two  $16 \times 4$ -bit chips can be combined to store 32 4-bit words. Each RAM is used to store 16 4-bit words. The data I/O pins of each RAM are connected to a common 4-line data bus. Only one RAM chip can be selected (enabled) at one time such that there are no bus contention problems. This is ensured by driving the respective CS inputs from different logic signals.

Since the total capacity of this memory module is  $32 \times 4$ , there are 32 different addresses. This requires five address bus lines. The upper address line AB<sub>4</sub> is used to select one RAM or the other (via the CS inputs) which is to be read from or written into. The other four address lines AB<sub>0</sub> to AB<sub>3</sub> are used to select one memory location out of 16 from the selected RAM chip.

To illustrate, when AB<sub>4</sub> = 0, the CS of RAM-0 enables this chip for read or write. Then, any address location in RAM-0 can be accessed by AB<sub>3</sub> through AB<sub>0</sub>. The latter four address lines can range from 0000 to 1111 to select the desired location. Thus, the range of addresses representing locations in RAM-0 is AB<sub>4</sub> AB<sub>3</sub> AB<sub>2</sub> AB<sub>1</sub> AB<sub>0</sub> = 00000 to 01111.

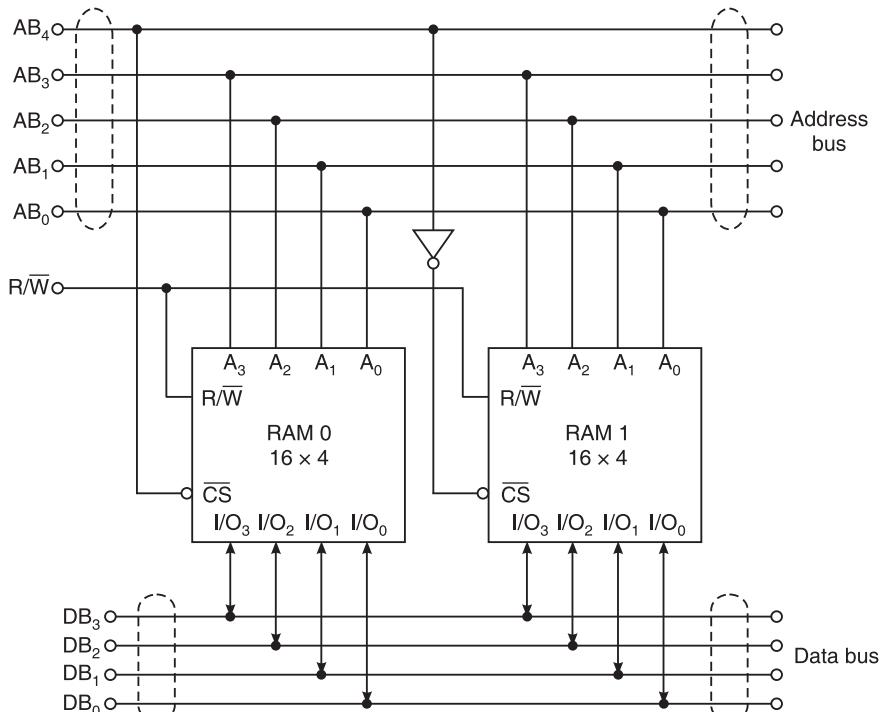
Note that, when AB<sub>4</sub> = 0, the CS of RAM 1 is HIGH, therefore, its I/O lines are disabled (high impedance) and cannot communicate (give or take data) with the data bus.



**Figure 18.8** Example 18.2: Eight 2125A  $1K \times 1$  chips arranged as a  $1K \times 8$  memory.

When  $AB_4 = 1$ , the roles of RAM-0 and RAM-1 are reversed. The RAM-1 is now enabled and the lines  $AB_3$  to  $AB_0$  select one of its locations.

Thus, the range of addresses selected in RAM-1 is  $AB_4 AB_3 AB_2 AB_1 AB_0 = 10000$  to 11111.



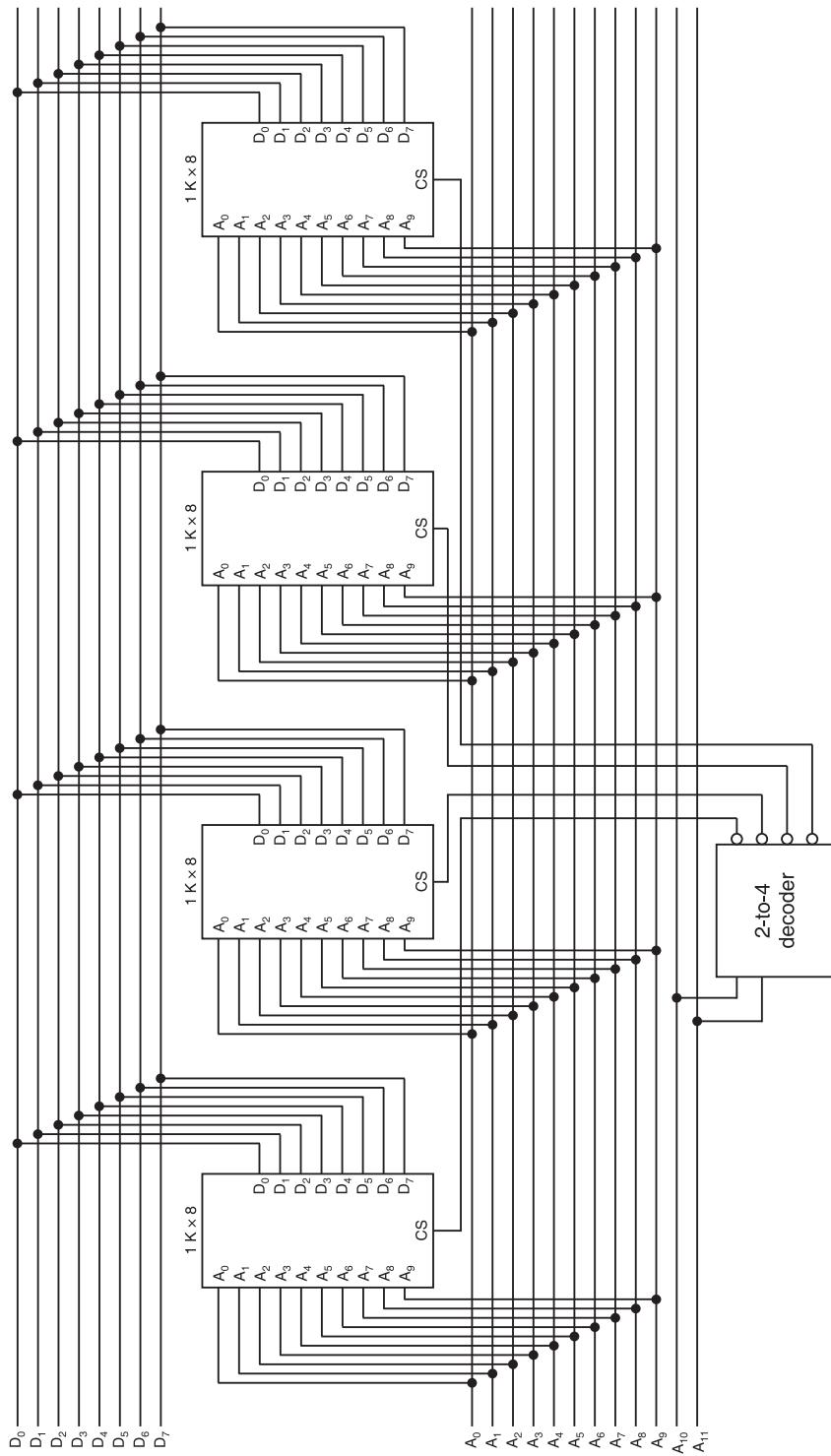
**Figure 18.9** Combining two  $16 \times 4$  chips for a  $32 \times 4$  memory.

**EXAMPLE 18.3** It is desired to combine several  $1K \times 8$  PROMs to produce a total capacity of  $4K \times 8$ . How many PROM chips are required? Show the arrangement.

#### Solution

Four PROM chips are required, each chip storing 1K of the 4K words. Since  $4K = 4 \times 1024 = 4096 = 2^{12}$ , 12 address lines are required. The arrangement is shown in Figure 18.10.

The decoder circuit is required for generating the  $\overline{CS}$  input signals. The two highest order lines  $A_{11}$  and  $A_{10}$  are used to select one of the PROM chips, the other 10 address bus lines go to each PROM to select the desired location within the selected PROM. The PROM selection is accomplished by feeding  $A_{11}$  and  $A_{10}$  into the decoder. The four possible combinations are decoded to generate active-LOW signals which are applied to  $\overline{CS}$  inputs. For example, when  $A_{11} = A_{10} = 0$ , the 0 output of the decoder goes LOW (all others are HIGH) and enables PROM-0. This causes the PROM-0 outputs to generate the data word internally stored at the address determined by  $A_9$  through  $A_0$ . All other PROMs are disabled.



**Figure 18.10** Example 18.3: Construction of 4K × 8 memory from 1K × 8 circuits.

While  $A_{11} = A_{10} = 0$ , the values of  $A_9$  through  $A_0$  can range from all 0s to all 1s. Thus, PROM-0 will respond to the following range of 12-bit addresses.

$$A_{11}-A_0 = 0000\ 0000\ 0000 \text{ to } 0011\ 1111\ 1111.$$

Similarly, when  $A_{11} = 1$  and  $A_{10} = 0$ , the decoder selects PROM-2 which then responds by putting the data word it has internally stored at the address  $A_9$  through  $A_0$ . Thus, PROM -2 responds to the following range of addresses.

$$A_{11}-A_0 = 1000\ 0000\ 0000 \text{ to } 1011\ 1111\ 1111.$$

#### 18.4.1 Combining DRAMs

DRAM ICs usually come with word sizes of 1 or 4 bits. In order to use these ICs in computer systems requiring word sizes of 8 or 16 bits, it is necessary to combine several of them in a manner similar to that for static RAMs and ROMs.

### 18.5 NON-VOLATILE RAMs

Though semiconductor RAM devices have the definite advantage of high-speed operation, they are volatile. The ROM, of course, is non-volatile, but it cannot be used as R/W memory. In some applications, the volatility of RAM can mean the loss of important or crucial data in the event of a power failure. There are two solutions to this problem. The first is to use memory that can be powered from back-up batteries whenever power failure occurs. This requires memory that will not rapidly drain the back-up batteries. The CMOS has the lowest power consumption of all semiconductor RAMs, and in many cases, can be powered from back-up batteries. Of course, when powered by batteries, the CMOS RAM chips need to be kept in their low power standby mode for the lowest power drain. Some CMOS SRAMs include a small lithium back-up battery right on the chip. Another solution is to use a device called a non-volatile RAM (NVRAM). A NVRAM contains a static RAM array and an EEPROM array on the same chip. It combines the high speed R/W operation of the static RAM with the non-volatile storage capability of the EPROM. Each cell in the static RAM has a corresponding cell in the EEPROM, and data can be transferred between the corresponding cells in both directions. During normal operation, data are written into and read from the static RAM cells as with any conventional SRAM IC. When power failure occurs or the power is turned off, the following sequence of events takes place.

1. An external voltage sensing circuit detects a drop in the ac source voltage, and sends a signal to the NVRAM's STORE input. Alternatively, the CPU may send the STORE signal after it receives a power failure interrupt signal.
2. This causes the NVRAM to transfer the contents of all the static RAM cells to their corresponding EEPROM cells in parallel, such that the complete transfer takes place in only a few ms. Because of its large output capacitors, the 5 V dc supply will keep the NVRAM powered up long enough for the transfer to be completed. The EEPROM now holds a copy of the RAM data at the instant of the power failure.
3. When power is restored, the NVRAM will automatically transfer the data from all EEPROM cells back into the RAM. The RAM now holds the same data which it had when the power interruption occurred.

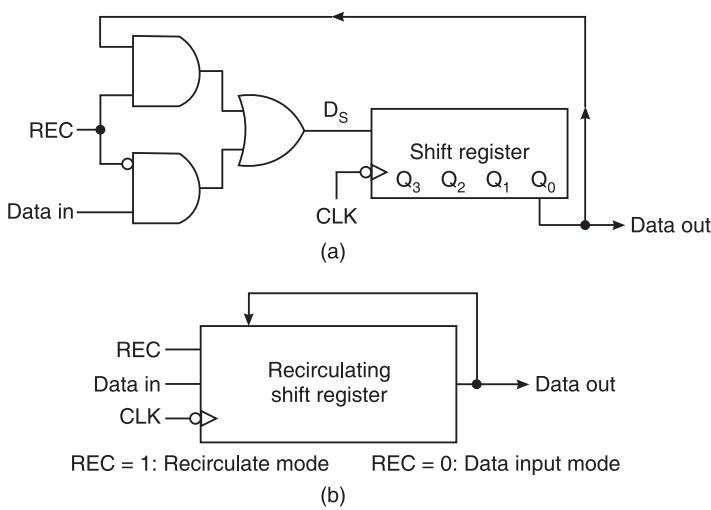
The NVRAM has the advantage of not needing a battery back-up. However, an NVRAM is more complex than a normal memory chip because it contains both RAM and EEPROM cells as well as the circuitry needed to transfer data between the two. For this reason, the NVRAMs are not available in very high capacities. When a high capacity non-volatile memory is required, it is best to use CMOS RAM with battery back-up.

## 18.6 SEQUENTIAL MEMORIES

The semiconductor memories that we have discussed so far are random access memories. The high speed operation of random access devices makes them suitable for use as the internal memory of the computer. Sequential access semiconductor memories utilize shift registers to store data that can be accessed in a sequential manner. Although they are not useful as internal computer memory because of their relatively slow speed, shift register memories find application in areas where sequential repetitive data are required. A primary example is the storage and sequential transmission of the ASCII-coded data for the characters on a video display. This data have to be supplied to the video display circuits periodically in order to refresh the displayed image on the screen. By using shift registers, the stored data can be recirculated to refresh the screen image periodically. Shift register memories are also used in digital storage oscilloscopes and logic analyzers.

### 18.6.1 Recirculating Shift Registers

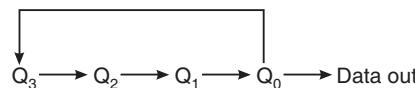
Figure 18.11a shows the block diagram of a recirculating shift register. There can be any number of FFs in a shift register, though four are shown in this illustration. Data enter the shift register from the serial input  $D_S$ , which shifts into  $Q_3$ ,  $Q_3$  shifts into  $Q_2$ ,  $Q_2$  into  $Q_1$ , and  $Q_1$  into  $Q_0$ . The  $Q_0$  output is recirculated back to the serial input through some controller. This logic provides two modes of operation that are controlled by the regulation input REC. The level at REC determines the source of the data that will reach the serial input.



**Figure 18.11** Block diagram of a recirculating shift register.

### Recirculate mode (REC = 1)

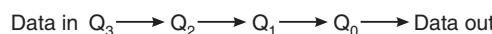
In this mode, the upper AND gate is enabled and  $Q_0$  output (Data out) is applied to  $D_S$ . As clock pulses are applied, the data in the shift register will recirculate as shown below.



While recirculating in the register, the data also appears at ‘Data out’, one bit at a time. In this mode, ‘Data in’ is inhibited and has no effect on the register data.

### Data input mode (REC = 0)

In this mode, the lower AND gate is enabled and the ‘Data in’ signal is applied to  $D_S$ . As clock pulses are applied, the data will shift as shown below.



There is no recirculation of data, since the upper AND gate is inhibited by REC = 0. This mode is used to enter new data at ‘Data in’ for storage in the register.

Figure 18.11b is a simplified symbol that we will use for the circulating shift register. The control logic is understood to be built into the symbol.

### 18.6.2 First In First Out (FIFO) Memories

The FIFO is also a sequential access memory formed by an arrangement of shift registers. There is an important difference between a conventional register and a FIFO memory register. In a conventional register, a data bit moves through the register only as new data bits are entered; but in a FIFO register, a data bit immediately goes through the register to the rightmost bit location that is empty. However, when a data bit is shifted out by a shift pulse, the data bits remaining in the register automatically move to the next position towards the output. This is illustrated in Tables 18.1 and 18.2.

**Table 18.1** Conventional shift register

Input	×	×	×	×	Output
0	0	×	×	×	→
1	1	0	×	×	→
1	1	1	0	×	→
0	0	1	1	0	→

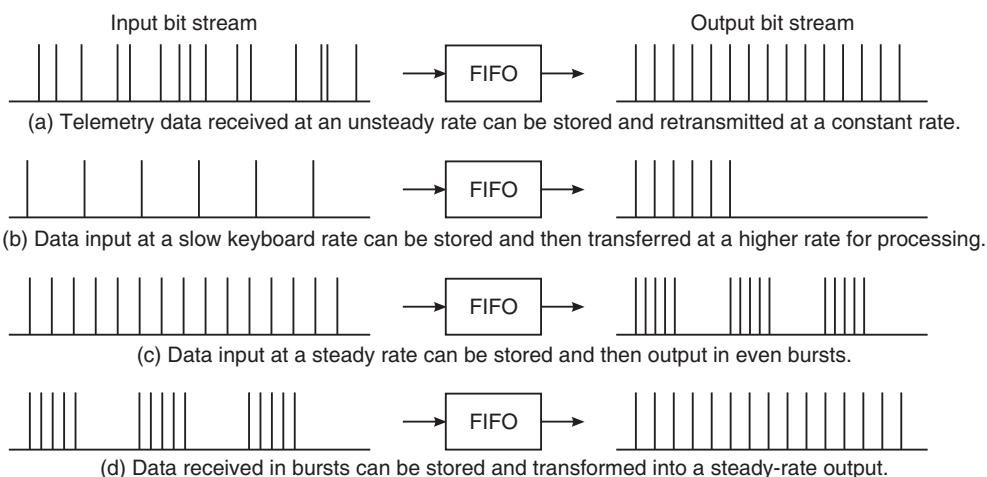
**Table 18.2** FIFO shift register

Input	-	-	-	-	Output
0	-	-	-	0	→
1	-	-	1	0	→
1	-	1	1	0	→
0	0	1	1	0	→

The FIFO memory is similar to the recirculating shift register memory in that, the order in which the data words are entered at ‘Data in’ is the same as the order in which they are read out at ‘Data out’. In other words, the first word that is written in is the first word that is read out; hence the name FIFO. There are two important differences, however, between a FIFO and the recirculating shift register memory. First, in a FIFO, the output data are not recirculated; once the output data are shifted out they are lost. Second, in a FIFO, the operation of shifting data into the memory is completely independent from the operation of shifting data out of the memory. In fact, the rate at which data are shifted in, is usually much different from the rate at which they are shifted out.

### Applications

One important application area of the FIFO is the case in which two systems of different data rates must communicate. Data can be entered into a FIFO at one rate and be put out at another rate. An example of this is the data transfer from a computer to a printer. The computer can send data to the printer at a much more rapid rate than the printer can accept it and print it out. A FIFO can act as a data-rate buffer between the computer and the printer by accepting data from the computer at a faster rate and storing it; the data are then shifted out to the printer at a slow rate. The FIFO can also be used as a data-rate buffer for the transmission of data from a relatively slow device like a keyboard to a much faster device like a computer. In this case, the FIFO accepts data from the keyboard at a slow rate and stores them. The data are then shifted out to the computer at a faster rate. In this way, the computer can be performing other tasks while the FIFO is slowly being filled with data. Other applications are: (a) data input at an unsteady rate can be stored and retransmitted at a constant rate by using a FIFO; (b) data output at a steady rate can be stored and then outputted in even bursts; (c) data received in bursts can be stored and retransmitted into a steady-rate output. Figure 18.12 shows the use of the FIFO as data-rate buffers.



**Figure 18.12** The FIFO used in data-rate buffering applications.

## 18.7 MAGNETIC MEMORIES

So far we have discussed semiconductor memory devices whose cells store data in the form of electrical charge or voltage. We will discuss now some storage devices whose basic storage

mechanism is magnetic rather than electronic. The characteristic common to all magnetic storage devices is their non-volatility. Magnetic core, magnetic tape and disk, hard drives, floppy disk, etc. are some of the magnetic memory devices.

### 18.7.1 Magnetic Core Memory

Magnetic core memory is a non-volatile random access read/write memory that was the predecessor to semiconductor memory. Almost all early computers used magnetic core as their primary internal memory before the wide availability of semiconductor memory. The basic magnetic core memory cell is a small dough-nut shaped core made of a ferromagnetic material. These cores are called ferrite cores and typically have a diameter of 0.05 inch. A small wire is threaded through the centre of the core. When a current pulse is passed through this wire, a magnetic flux is set up in the core in a direction that depends on the direction of the current. Because of the core's magnetic retentivity, it stays magnetized even after the current pulse is terminated. In other words, it is non-volatile. The two directions of magnetization are used to represent 1 and 0, respectively.

Stored data are read from a core by magnetizing it in the 0 direction (i.e. writing a 0) and using a second threaded wire as a sense wire. The size of the voltage induced in the sense wire will be greater if the core is initially in 1 state rather than in 0 state. The reading of a core is said to be destructive, since it always leaves it in 0 state. Thus, the data must be rewritten after a read operation.

Magnetic core memory systems have access times from 100 ns to 500 ns and can still be found in some old minicomputers and mainframes. But for all practical purposes, their large physical size and complex interface circuitry have made them virtually obsolete.

### 18.7.2 Magnetic Disk Memory

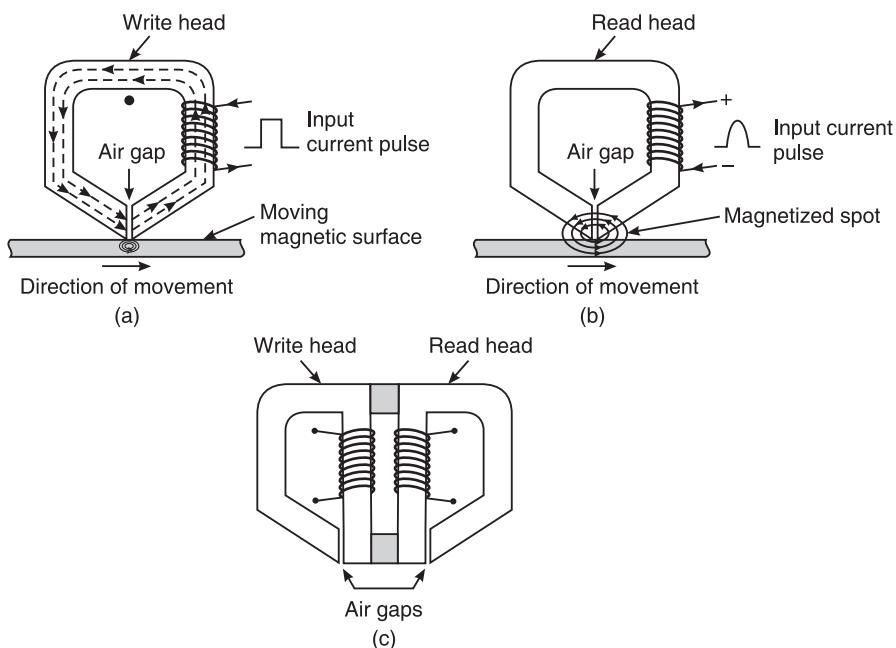
A magnetic disk is a flat, circular plate that is coated with a magnetic material. Binary data is stored on the disk by magnetizing tiny regiments of the surface, and data is read from the disk by sensing that magnetization. Magnetic disk memories are used primarily for peripheral or auxiliary memory and for mass storage, because their access times are slower than those of the semiconductor memories. They are less costly per bit of storage capacity than the semiconductor memories and they have become the dominant type of peripheral memory in computer systems of all sizes. The disks themselves range in size from a 2-inch diameter 'floppy' disk used with microcomputer systems to a 14-inch diameter hard disk used in larger systems. Floppy disks made of plastic are inexpensive and readily portable. Hard disks made of aluminium have much greater storage capacity than that of floppies and are used in complex and more expensive systems where higher speeds and larger capacities are required.

#### Read and write mechanisms

The basic principle of writing and reading a disk is the same whether the disk is small or large, hard or floppy. These devices use a magnetic surface moving past a read/write head to store and retrieve data.

A simplified diagram of the magnetic surface read/write operation is shown in Figure 18.13. A data bit (1 or 0) is written on the magnetic surface by magnetizing a small segment of the surface as it moves by the write head. The direction of the magnetic flux lines is controlled by the direction of the current pulse in the winding as shown in Figure 18.13a. At the air gap in the write head, the magnetic flux takes a path through the surface of the storage device. This magnetizes a tiny spot

on the surface in the direction of the field. A magnetized spot of one polarity represents a binary 1, and one of the opposite polarity represents a binary 0. Once a spot on the surface is magnetized, it remains there until written over with an opposite magnetic field. Thus, magnetic storage is non-volatile. To write a sequence of 1s and 0s, the disk is rotated and the winding is driven by a sequence of current pulses, each pulse having one direction or the other. Thus, 0s and 1s are stored in a sequence around a circular path on the disk. To read data on a disk, a read head similar in construction to the write head is used. When the magnetic surface passes a read head, the magnetized spots produce magnetic fields in the read head which induce voltage pulses in the winding. The polarity of these pulses depends on the direction of the magnetization spots and indicates whether the stored bit is a 1 or a 0. This is illustrated in Figure 18.13b. Very often, the read and write heads are combined into a single unit as shown in Figure 18.13c.



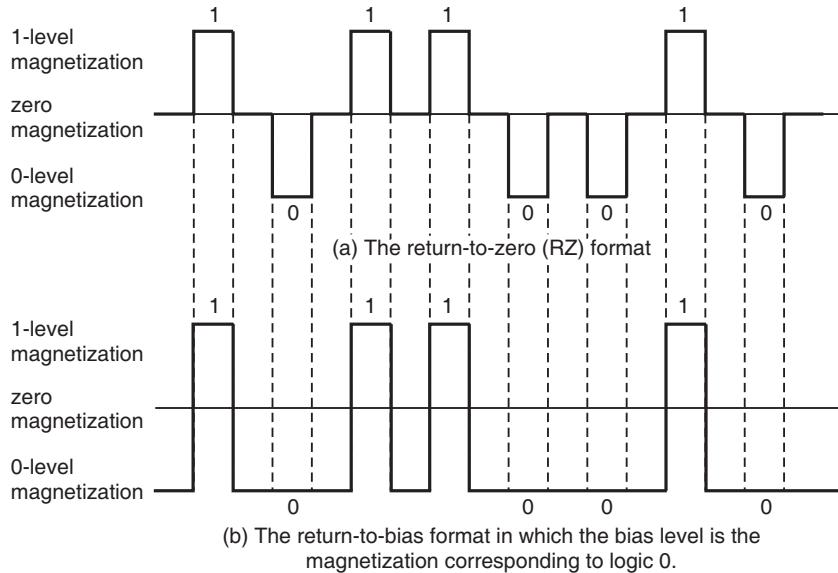
**Figure 18.13** Read/write function on a magnetic surface.

### 18.7.3 Magnetic Recording Formats

Our description of bit storage as a sequence of discrete dots having one magnetic polarity or the other implied that the regions between the dots were unmagnetized. That may or may not be the case, depending on the format used to record data. Several ways in which digital data can be represented for purposes of magnetic surface recording are return-to-zero (RZ), non-return-to-zero (NRZ), bi-phase, Manchester, and the Kansas city standard. These waveform representations are separated into bit times—the intervals during which the level or frequency of the waveform indicates a 1 or 0 bit. These bit times are definable by their relation to a basic system timing signal or clock.

In return-to-zero (RZ) recording, there is, in fact zero magnetization between every bit. Figure 18.14a illustrates the RZ format. As per convention, we regard one magnetic polarity as

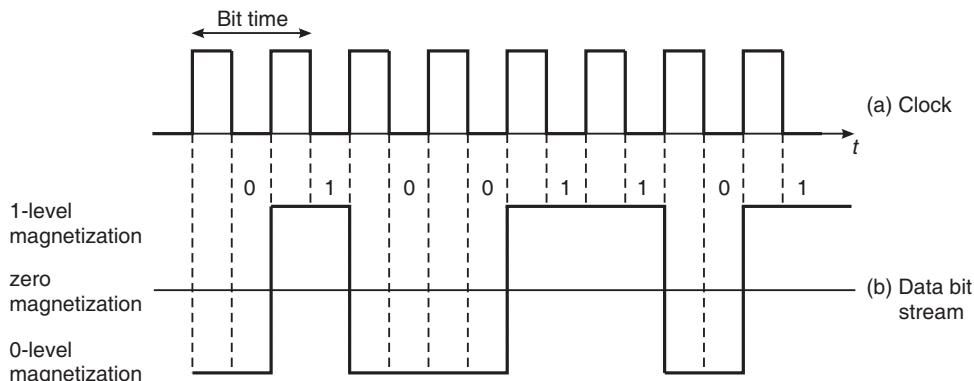
positive (corresponding to logic 1) and the other as negative (logic 0). The important point to note is that magnetization returns to zero between every bit, including adjacent 1s and adjacent 0s.



**Figure 18.14** Return-to-zero and return-to-bias formats for recording data.

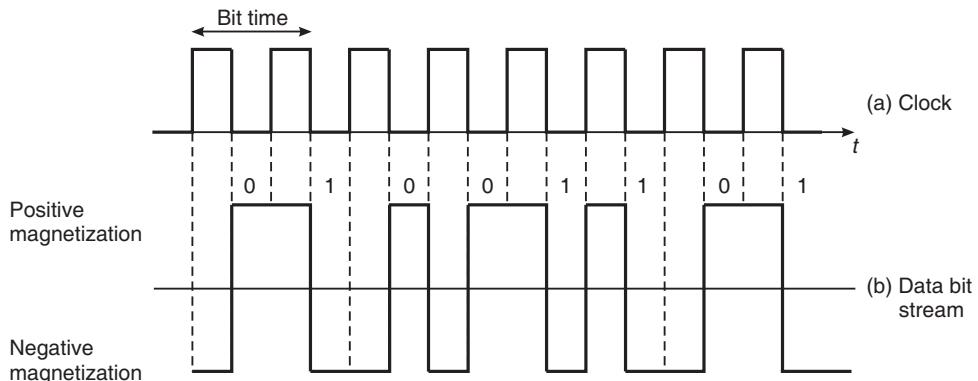
A generalization of RZ is called the return-to-reference (or bias), wherein the level to which magnetization returns between bits can be any level between 1 and 0. Figure 18.14b shows an example in which magnetization ‘returns’ to the polarity representing logic 0 between every bit. Although there is no return involved between adjacent 0s, this format has the advantage that the total change in magnetization between a 0 and a 1 is large, making it easier to detect such cases. In this format as well as the others that we will discuss, reading and writing are synchronized by a clock signal, usually recorded on the disk itself. A clock is necessary to ascertain the precise time interval, called the *bit time* during which successive bits occur.

Figure 18.15 illustrates a non-return-to-zero (NRZ) waveform. In this case, a 1 or a 0 level remains during the entire bit time. If two or more 1s occur in succession, the waveform does not return to the 0 level until a 0 occurs.



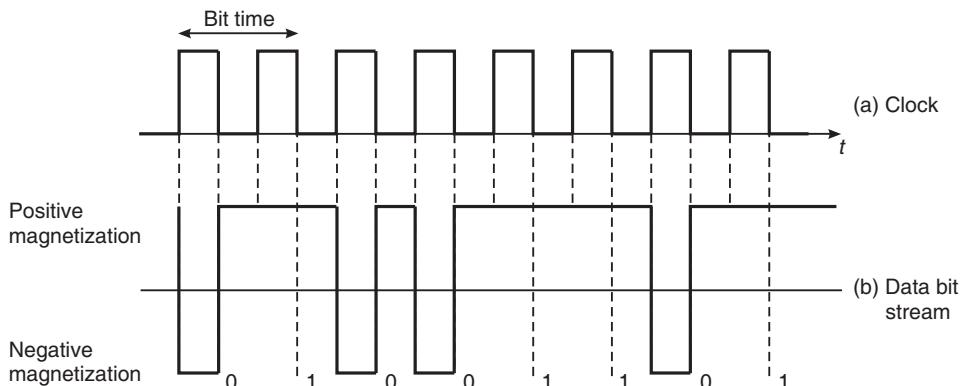
**Figure 18.15** An example of data (01001101) recorded in an NRZ format.

Figure 18.16 is an illustration of a bi-phase waveform. In this type, a 1 is a HIGH level for the first half of a bit time and a LOW level for the second half, so a HIGH-to-LOW transition occurring in the middle of a bit time is interpreted as a 1. A 0 is represented by a LOW level during the first half of a bit time followed by a HIGH level during the second half, so a LOW-to-HIGH transition in the middle of a bit time is interpreted as a 0.



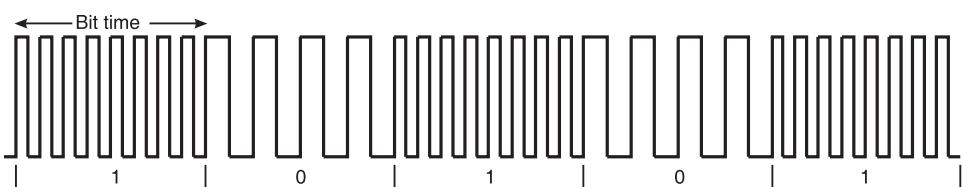
**Figure 18.16** An example of data (01001101) recorded in a bi-phase format.

Manchester is another type of phase encoding in which a HIGH-to-LOW transition at the start of a bit time represents a 0 and no transition represents a 1. Figure 18.17 illustrates a Manchester waveform.



**Figure 18.17** An example of data (01001101) recorded in Manchester format.

The Kansas city method uses two different frequencies to represent 1s and 0s. The standard 300 bits/second version uses four cycles of 1200 Hz signal to represent a 0 and eight cycles of 2400 Hz signal to represent a 1. This is illustrated in Figure 18.18.

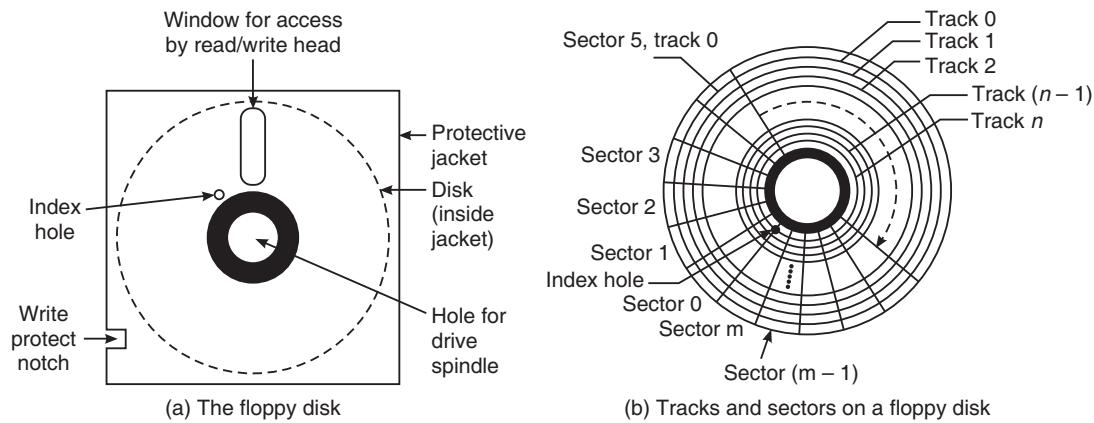


**Figure 18.18** An example of data (10101) recorded in Kansas city format.

#### 18.7.4 Floppy Disks

Floppy disks (often called diskettes) are thin flexible mylar disks and are available in 2-inch, 3½-inch, 5¼-inch, and 8-inch sizes. They are permanently enclosed in a square jacket, as shown in Figure 18.19a. A small index hole in the floppy is used for referencing the beginning point of the stored data. When the floppy is installed in a drive, it spins during read/write operations, inside the jacket. A window in the jacket exposes the surface of the floppy for access by the read/write head. Unlike the larger, hard-disk systems, the read/write head makes physical contact with the floppy. A write-protect notch in the jacket can be covered with a piece of tape to protect stored data, i.e. to prevent writing new data to the floppy.

Data on a floppy disk are stored in concentric circles, called *tracks* as illustrated in Figure 18.19b. The disk is further divided into *sectors*. Each track and each sector is numbered, therefore, the portion of a particular track within a particular sector defines one *block* of data having a specific track and sector address. The address is stored at the beginning of each such block; therefore the read/write head as it moves across the spinning disk, can identify and access any sector on any track. Typical access times are 200–500 ms.



**Figure 18.19** The floppy disk.

The index hole marks the beginning of sector 0 and track 0. A soft-sectored disk is one for which all the remaining sectors must be defined and identified by the disk controller and computer system before the disk can be used. This procedure is called *formatting* the disk. The disk is said to be soft-sectored because formatting is performed under the control of a program (software) and because the formatting can be changed at will. A hard-sectored disk has all sectors defined at the time of manufacture. It typically has index holes marking the beginning of each sector. The number of tracks and the number of sectors on a floppy disk vary widely with the system where it is used, the size of the disk, and the recording density for which it is designed. Disks are available in *single density* and *double density* and for single-sided and double-sided recording. Each sector on each track typically contains 128 or 256 bytes of data in addition to the bits reserved for addresses and various other identification and error-checking functions.

Floppy disk capacities typically range from 100 K to a few MB. Floppies have access times about 10 times larger and data rates about 10 times slower than those of hard disks. The average

access time of floppy drives is 100 to 500 ms, and the average data transfer rates range from 250 K/s to 1 Mbit/s. When inserted into the disk drive unit, a floppy disk is rotated at a fixed speed of 300 to 360 rpm, which is much slower than the speed of the hard disks.

Although floppy disk systems are slower and have less capacity than hard disk systems, they do have the advantages of low cost and portability. They can be easily transported from one computer to another, and can be sent through the mail.

**EXAMPLE 18.4** A single-sided double-density 8-inch floppy disk has tracks numbered 0 to 76 and sectors 0 to 25. What is the total storage capacity of the disk?

**Solution**

$$(77 \text{ tracks}) \times \left( 26 \frac{\text{Sectors}}{\text{track}} \right) \times \left( 256 \frac{\text{bytes}}{\text{sector}} \right) = 512,512 \text{ bytes}$$

**EXAMPLE 18.5** The total storage capacity of a floppy disk having 80 tracks and storing 128 bytes/sector is 163,840 bytes. How many sectors does the disk have?

**Solution**

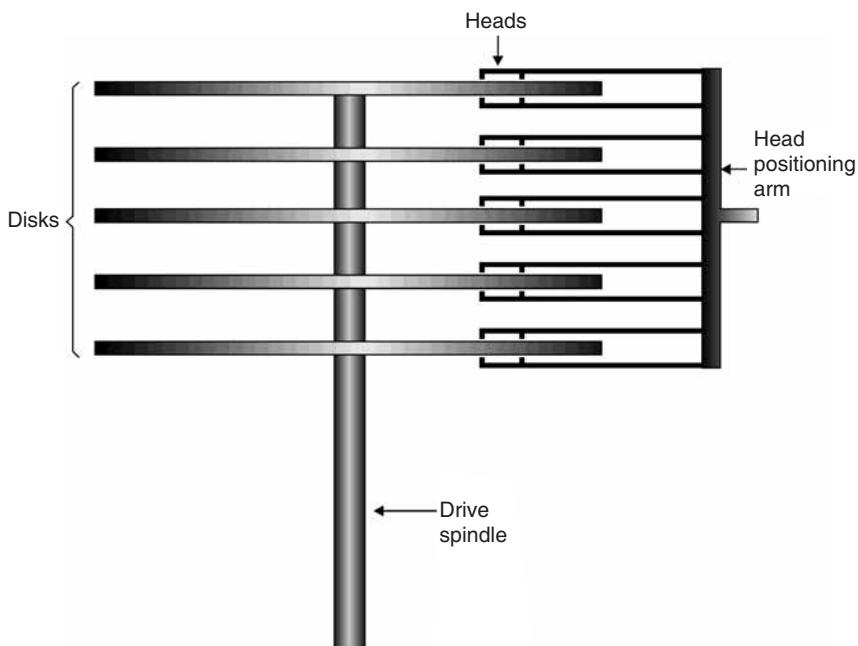
$$163,840 = (80) \times (\text{number of sectors}) \times (128). \text{ Therefore, the number of sectors} = 16.$$

### 18.7.5 Hard Disk Systems

In a hard disk system, the data are recorded in concentric tracks and sectors on an aluminium disk that is covered with a magnetic coating on both sides. Hard disks come in sizes from 3½ inch to 20 inch and are rotated at speeds from 1000 to 3600 rpm. A single 14-inch hard disk may have as many as 500 tracks and store 35 or more megabytes. Hard disks are stacked as illustrated in Figure 18.20 and rotated by a common drive spindle. Note that, multiple read/write heads may also be used, thus reducing the access times. A hard disk spins about 10 times faster than a floppy disk (about 3600 rpm versus 360 rpm) which further reduces the access time. The access time in a hard disk system is of the order of 20 ms versus 200 ms or more in a floppy disk system. Hard disk data transfer rates are 1 M to 10 M bits/s. This is the rate at which data can be read from the disk once the head is in position.

Multi-disk drives with capacities in the range of thousands of MBs are available. Most hard disks are fixed drives in which the disks are permanently mounted on the drive mechanism and cannot be removed in normal usage. Some hard disk systems use removable disks called *disk cartridges* or *disk packs*.

The high speed of rotation of hard disks produces a cushion of air a few micro inches above the disk surface. The read/write head floats on this cushion, such that it never touches the disk surface. It is thus called the *flying head*. This prevents wearing down of the disk surface and the read/write head. Since the clearance is very small, smoke or dust particles pose a serious threat of contamination. For this reason, most hard disk drives blow filtered air across the disks to keep them relatively clean. The disks are sealed in protective cases and cannot ordinarily be handled by a user.



**Figure 18.20** A hard disk unit consisting of five stacked disks.

#### 18.7.6 Magnetic Tape Memory

The primary role of the magnetic tape is mass storage and back-up—the duplication and preservation of data stored in other media like hard disks. Back-up is important in many systems because, critical records, computer programs and/or scientific data are susceptible to loss through power failures, mechanical malfunctions or human error. Magnetic tape storage is non-volatile and has immense capacity at relatively low cost per bit. A single tape may be of very long length and can store thousands of megabyte. The tape is wound on reels that must be unwound to read or write at a specific location. The principal drawback of the tape memory is the long access time required to find a specific block of data. This drawback is not a disadvantage when the tape is used as disk back-up. The data is not read from the tape very often, and when read, the complete tape will be read from the beginning. The technology of magnetic storage and read/write mechanism is quite similar to that of magnetic disks. The adjacent regions on the surface of the tape are magnetized with one polarity or the other to represent 1s and 0s, and read/write heads are used to sense or alter the magnetization. The recording formats discussed in connection with the magnetic disk memory can also be used for magnetic tape storage, the NRZ format being the most common.

Tapes used in large systems, typically, have nine parallel tracks and a read/write head for each track. Seven of the tracks are used for storing data in ASCII code and the other two for polarity and timing. Storage density on a single track can range from 200 to 1600 bits per inch.

Audio cassette tapes are sometimes used for auxiliary storage in small personal computer systems, although floppy disks have already displaced the tape in that role. A frequency modulation (FM) format is used for recording on the cassette tape, whereby 0s and 1s are represented by pulse

trains having different frequencies. In the Kansas city standard FM, a 0 is represented by four cycles of a 1200 Hz signal and a 1 by eight cycles of a 2400 Hz signal.

### 18.7.7 Magnetic Bubble Memory

Magnetic bubble memory (MBM) is a semiconductor memory that stores binary data in the form of tiny magnetic domains (bubbles) on a thin film of magnetic material. It is constructed from a magnetic material (yttrium-iron garnet) that has the property whereby small cylindrically shaped domains called *bubbles* can be created by subjecting the material to a strong externally applied magnetic field. The presence of a magnetic bubble in a specific position represents a stored 1, and its absence represents a stored 0. Continuously changing magnetic fields are used to move the bubbles around in loops inside the magnetic material much like recirculating shift registers. The data circulates past a pick-up point where it is available to the outside world at the rate of typically 50,000 bits per second. Clearly, the MBMs are sequential access devices.

The MBMs are better suited for the external mass storage function that has been previously dominated by magnetic tape and disk storage. At present, the MBMs are about 100 times faster than the floppy disks but somewhat more costly because of the required support circuitry. As and when their cost comes down, we might see more and more MBM systems being used in place of the slower, less reliable floppy disk systems.

The MBMs are compact and dissipate very little power (typically  $1 \mu\text{W}/\text{bit}$ ). They are capable of storing large amounts of data. They are also non-volatile. When power goes off, the bubbles simply remain in their fixed positions. When power is restored, the bubbles start circulating again around their loops. Unlike other non-volatile semiconductor memories such as ROMs, PROMs, EPROMs and EEPROMs, the MBMs can be written into and read from with equal ease. Because of these features, the MBMs are competitive with semiconductor memories in many applications. The main disadvantage is that it takes much longer to get data into and out of an MBM compared to semiconductor memories, because MBMs are serially accessed rather than randomly accessed. The typical access times are in the range of 1 to 20 ms. Therefore, an MBM is not suitable for use as the main internal memory. Compared with non-volatile tape and disk memories, an MBM system has no moving parts and is, therefore, quieter and more reliable.

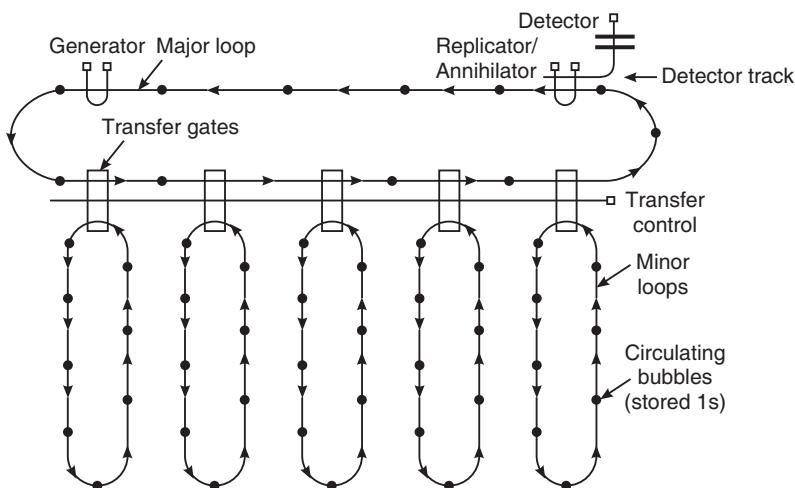
#### Major/Minor loop architecture

Loops in an MBM are in the form of continuous, elongated paths of Chevron patterns. The major/minor loop arrangement consists basically of one major loop and many minor loops as shown in Figure 18.21.

The minor loops are essentially the memory cell arrays that store the data bits. The major loop is primarily a path to get data from the minor loops to the output during read and to get data from the input to the minor loops during write.

Five control functions are used in the read/write cycles of an MBM. These are generation, transfer, replication, annihilation, and detection.

**The read cycle.** Data are read from an MBM in blocks called *pages*. Basically a ‘page’ consists of a number of bits equal to the number of minor loops. For example, a typical MBM may have 256 minor loops, each of which is capable of storing 600 bits. In this case, the page size would be 256 bits.



**Figure 18.21** Circulation of magnetic bubbles.

Each bit in a given page occupies the same relative location in each minor loop. During read, all of the bits in a page of data are shifted to the transfer gates and on to the major loop at the same time. Then, they are serially shifted around the major loop to the replicator/annihilator where each bubble is ‘stretched’ by the replicator until it ‘splits’ into two bubbles. One of the replicated bubbles (or no bubble as the case may be) is transferred to the detector where the presence or absence of the bubble is sensed and translated to the appropriate logic level to represent a 1 or a 0. The other replicated bubble continues along the major loop and is transferred back on to the appropriate minor loop for storage. The process continues until each bit in the page is read. The replication/detection process results in a non-destructive read-out.

**The write cycle.** Before a new page of data can be written into an address, the data currently stored at that address must be annihilated. This is done by a destructive read-out in which the bubbles are not replicated. Now, the new page of data is produced by the generator one bit at a time. Each bit is injected on to the major loop until the entire page of data has been entered. It is then serially moved into position and transferred on to the minor loops for storage.

## 18.8 OPTICAL DISK MEMORY

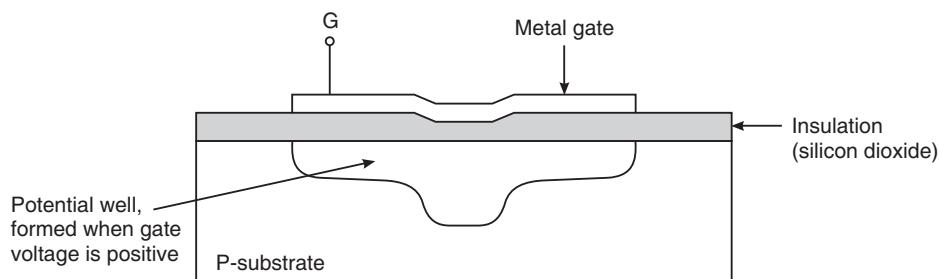
The optical disk memory is the latest mass memory technology that has a promising future. Its operation is based on the reflection or scattering of an extremely narrow laser beam off a disk that has microscopic pits or bubbles representing logic 1s ‘burned’ on its surface. One strong point in favour of the optical disk memory is its high capacity—4½ inch disk with 650 MB capacity is readily available. Other advantages are its relatively low cost and its immunity to dust. Its access times and data transfer rates are comparable to the newest hard disk systems.

The optical disk systems are available in three basic types based on their writability. Disks that can only be read from are called optical ROM (OROM) or compact disc ROM (CD-ROM). They are used to store large fixed databases such as dictionaries or encyclopaedias. An optical disk that can be written to once is called a write-once read-many (WORM) disk. The CD-ROMs

and WORMs cannot be erased. The read/write optical disk can be written and rewritten as often as desired, and therefore, operates like a magnetic hard disk. It uses a different disk surface from other types. Its surface is coated with a magnetic material that can have its magnetic properties changed by a laser beam. The optical disk storage has the potential to emerge as the principal mass storage device.

### 18.9 CHARGE-COUPLED DEVICES

The charge-coupled device (CCD) memory is a type of dynamic memory, in which packets of charges (electrons) are continuously transferred from one MOS device to another. The structure of a single MOS device is quite simple and is shown in Figure 18.22. When a high voltage is applied to the metal gate, holes are repelled from a region beneath the gate in the P-type substrate. This region called a *potential well* is then capable of accepting a packet of negative charges. Data in the form of charge is transferred from one device to an adjacent one by clocking their gates.



**Figure 18.22** Structure of a single MOS device in a CCD memory.

The CCD memory is inherently serial. Practical memories are constructed in the form of shift registers, each shift register being a line of CCDs. By controlling the timing of the clock signals applied to the shift registers, data can be accessed one bit at a time from a single register or several bits at a time from multiple registers. The principal advantage of the CCD memory is that, its single cell structure makes it possible to construct large capacity memories at low cost. On the other hand, like other dynamic memories, it must be periodically refreshed and driven by rather complex, multi-phase clock signals. Since data are stored serially, the average access time is long compared with the semiconductor RAM memory.

#### SHORT QUESTIONS AND ANSWERS

1. What is memory?  
A. Basically memory is a means for storing data or information in the form of binary words. It is made up of storage locations in which numeric or alpha numeric information or programs may be stored.
2. What is a data memory?  
A. Memory used to store data is called data memory.

- 3. What is program memory?**
  - A. Memory used to store programs is called program memory.
- 4. What are stored program type computers?**
  - A. Computers which store programs in their memory are called stored program type computers.
- 5. How are programs stored in a computer?**
  - A. In a computer, programs are stored as a set of machine language instructions in binary codes.
- 6. How is each memory location identified?**
  - A. Each memory location is identified by an address.
- 7. What is a storage element called?**
  - A. A storage element is called a cell.
- 8. What is memory capacity?**
  - A. The total number of bits a memory can store is its capacity.
- 9. What is main memory?**
  - A. The main memory is an internal part of the computer and is very fast. The program to be currently executed and any data used by the program are stored in the main memory. Semiconductor memory is well suited as the main memory.
- 10. What is peripheral memory?**
  - A. The peripheral memory also called the auxiliary memory is add on memory with very large storage capacity. It consists of magnetic tapes or disks. It is much slower than the main memory.
- 11. What is mass memory?**
  - A. Peripheral memory made up of magnetic tapes and disks which have the capacity to store millions of bits of data without the need for electrical power is called mass memory.
- 12. What are stored program computers?**
  - A. Stored program computers are those in which programs are stored as a set of machine language instructions—binary codes—in memory.
- 13. What do you mean by a word?**
  - A. A word is the fundamental group of bits used to represent one entity of information.
- 14. What do you mean by word size?**
  - A. The word size is the number of bits in a word.
- 15. What do you mean by writing?**
  - A. The process of storing data in memory is called writing in memory.
- 16. What do you mean by reading memory?**
  - A. Retrieving data from memory is called reading memory.
- 17. What is volatile memory?**
  - A. The memory whose contents will be lost once the power is off is called volatile memory.
- 18. What is non-volatile memory?**
  - A. Non-volatile memory is the memory which does not loose its contents when power is off.
- 19. What is random access memory?**
  - A. Random access memory is a memory in which memory locations can be accessed directly and immediately without going through a series of addresses. The access time is the same for every memory location.
- 20. What is software?**
  - A. Programs stored in RWMs are referred to as software because they can be easily altered.

## **974 FUNDAMENTALS OF DIGITAL CIRCUITS**

- 21.** What is firmware?
  - A.** Programs which are not subject to change are called firmware. Normally these programs are stored in ROM.
- 22.** Distinguish between dynamic and static memory.
  - A.** Memory that utilizes conventional storage devices (latches) is called static memory, whereas memory in which data is stored as charges on capacitors which must be periodically refreshed is called dynamic memory. Dynamic memory is available only in MOSFET circuits, whereas static memory is available in both BJT and MOSFET technologies. The dynamic memory is more complex than static memory.
- 23.** What is memory access time?
  - A.** Memory access time is the time required for valid data to appear on outputs after activation of appropriate inputs.
- 24.** What is sequential access memory?
  - A.** Sequential access memory is one in which the memory locations are accessed serially. The access time is not constant and depends on the particular memory location.
- 25.** What are the applications of ROMs?
  - A.** ROMs have innumerable applications. Some of the most common applications are:

(a) Microcomputer program storage	(b) Boot strap memory
(c) Data tables	(d) Data converters
(e) Character generators	(f) Functions generators.
- 26.** Distinguish between SRAMs and DRAMs.
  - A.** SRAMs are available both in bipolar and MOS technologies whereas DRAMs are available only in MOS technology. The SRAMs use latches for storage whereas DRAMs use capacitors for storage. DRAMs require refresh. DRAMs are preferred for high capacity, low power requirement and moderate speed.
- 27.** Where are ECL RAMs used?
  - A.** ECL RAMs are used as cache memory and scratch pad memory which are very fast.
- 28.** What are dynamic RAM controllers?
  - A.** Dynamic RAM controllers are special ICs which handle the refresh operation as well as the address multiplexing needed by the DRAM systems.
- 29.** What is FIFO memory?
  - A.** FIFO memory is a sequential access memory formed by an arrangement of shift registers. It is similar to the recirculating shift register memory. It is called FIFO memory because the first word that is written in is the first word that is read out.
- 30.** What are the differences between a FIFO and recirculating shift registers memories?
  - A.** (a) In FIFO, the output data are not recirculated. Once the output data are shifted out they are lost.  
(b) In FIFO, the operation of shifting data into the memory is completely independent from the operation of shifting data out of the memory. In fact, the rate at which data are shifted in is usually much different from the rate at which they are shifted out.
- 31.** Why is the name MBM?
  - A.** The name MBM is so because in this memory data are stored as tiny magnetic domains called bubbles.

- 32.** Which is the newest mass memory technology? What are its chief advantages?
- A. The optical disk memory is the newest mass memory technology. The optical disk systems are available in three basic types based on their writability—ORAM or CDROM, WORM, and Read/Write. Its advantages are:
- (a) its high capacity
  - (b) its relatively low cost and
  - (c) its immunity to dust.
- 33.** What is CCD memory ? What is its principal advantage?
- A. CCD memory is a type of dynamic memory in which packets of charges (electrons) are continuously transferred from one mass device to another. It is inherently a serial memory. The principal advantage of the CCD memory is that its single cell structure makes it possible to construct large capacity memories at low cost. The disadvantages are:
- (a) It is a serial memory. So the average access time is large.
  - (b) Like other dynamic memories it must be refreshed periodically.
- 34.** How are ROMs and RAMs classified?
- A. Depending on the device used (a) ROMS are either BJT ROMs or MOSFET ROMs. BJT ROMs may be MROMs or PROMs whereas MOSFET ROMs may be MROMs, PROMs, EPROMs and EEPROMs. (b) RAMs are also either BJT RAMs or MOSFET RAMs. BJT RAMs are only static RAMs, whereas MOSFET RAMs may be static RAMs or dynamic RAMs.
- 35.** What is the principal difference between the BJT memory and the MOSFET memory in terms of their speed and sizes?
- A. MOSFET memory is small in size compared to BJT memory, but its speed is less compared to the BJT memory.
- 36.** What are the main advantages and disadvantages of floppy disk storage compared to hard disk storage?
- A. Although floppy disk systems are slower and have less capacity than hard disk systems, they do have the advantages of low cost and portability. They can be easily transported from one computer to another.
- 37.** What is the major application of tape memory?
- A. The major application of tape memory is mass storage and backup. Magnetic tape storage is non-volatile and has immense capacity at relatively low cost per bit.
- 38.** Where are SRAMs preferred?
- A. For applications where speed and reduced complexity are more critical than space and power consumption, SRAMs are preferred. Many microprocessor-controlled instruments and appliances have very small memory capacity requirements. Some instruments such as digital storage oscilloscopes and logic analyzers require very high speed memory. For such applications SRAMs are normally used.
- 39.** Which memory technology needs the least power?
- A. The CMOS memory technology needs the least power?
- 40.** What is memory expansion? Why is it required?
- A. The process of increasing the word size or capacity by combining a number of IC chips is called memory expansion. It is required because, in most IC memory applications, the required memory capacity or word size cannot be satisfied by one memory chip. So several memory chips are combined to provide the desired capacity and the word size.

41. What is bit-organized memory?  
A. Bit-organized memory is one in which each IC stores one bit of each word. A  $128\text{ K} \times 8$  bit-organized memory requires eight numbers of  $128\text{ K} \times 1$  memory chips.

42. What is word organized memory?  
A. A word organized memory is one in which every bit of a word is stored in each circuit. A word organized  $128\text{ K} \times 8$  memory might consist of sixteen  $8\text{ K} \times 8$  circuits.

43. A certain memory has a capacity of  $32\text{ K} \times 16$ . How many bits are there in each word? How many words are being stored and how many memory cells does this memory contain?  
A. A  $32\text{ K} \times 16$  memory has 16 bits in each word.  $32 \times 1024$  words are being stored. It contains  $32 \times 1024 \times 16$  memory cells.

44. How many  $32 \times 4$  RAMs are needed to form a  $32 \times 8$  memory? Is this an example of word capacity expansion or word length expansion?  
A. Two  $32 \times 4$  RAMs are needed to form a  $32 \times 8$  memory. It is an example of word length expansion.

45. How many  $16\text{ K} \times 4$  memory circuits are required to construct each of the following memories?  
(a)  $256\text{ K} \times 8$       (b)  $128\text{ K} \times 16$       (c)  $1\text{ M} \times 4$   
A. (a) 32 circuits      (b) 32 circuits      (c) 64 circuits.

## **REVIEW QUESTIONS**

- Explain the meaning of the following types of memories:  
(a) Memory (b) RAM  
(c) ROM (d) PROM  
(e) EPROM (f) EEPROM  
(g) Mask/ROM (h) Volatile memory  
(i) Non-volatile memory (j) Static memory  
(k) Dynamic memory (l) Main memory  
(m) Peripheral memory
  - Define the following terms:  
(a) Memory cell (b) Address  
(c) Byte (d) Access time  
e) Memory word
  - Define the following terms:  
(a) Firmware (b) Software (c) Hardware
  - A  $1M \times 8$  memory is to be constructed by using  $64 K \times 8$  circuits, each of which has an active-low chip select input:  
(a) How many  $64 K \times 8$  circuits are required?  
(b) What type of decoder is necessary to select individual  $64 K \times 8$  circuits?  
(c) What are the address inputs to the decoder?
  - How are 1s and 0s represented in an MBM? What is the disadvantage of MBMs vis-a-vis semiconductor memories? What is a magnetic bubble? How is it created?

## **FILL IN THE BLANKS**

1. Computers which store programs in their memory are called \_\_\_\_\_ computers.
  2. Each memory location is identified by \_\_\_\_\_.
  3. The fundamental group of bits used to represent one entity of information is called a \_\_\_\_\_.
  4. A storage element is called a \_\_\_\_\_.
  5. \_\_\_\_\_ memories are well suited as the main memory.
  6. Peripheral memory is also called \_\_\_\_\_ memory.
  7. The process of storing data in memory is called \_\_\_\_\_.
  8. Retrieving data from memory is called \_\_\_\_\_.
  9. Programs stored in RWMs are called \_\_\_\_\_.
  10. Programs stored in ROMs are called \_\_\_\_\_.
  11. Memory that utilizes conventional storage devices is called \_\_\_\_\_.
  12. The time required for valid data to appear on outputs after activation of appropriate inputs is called \_\_\_\_\_.
  13. \_\_\_\_\_ RAMs are available only in MOS technology whereas \_\_\_\_\_ RAMs are available both in bipolar and MOS technologies.
  14. \_\_\_\_\_ are used in cache memory and scratch pad memory.
  15. The \_\_\_\_\_ is the newest mass memory technology.
  16. CCD memory is inherently a \_\_\_\_\_ memory.
  17. CCD memory is a type of \_\_\_\_\_ memory.
  18. BJT memory is \_\_\_\_\_ than MOSFET memory.
  19. MOSFET memory occupies \_\_\_\_\_ space than BJT memory.
  20. \_\_\_\_\_ memory technology needs the least space.
  21. Memory in which each IC stores one bit of each word is called \_\_\_\_\_ memory.
  22. Memory in which every bit of a word is stored in each circuit is called \_\_\_\_\_ memory.
  23. The major application of tape memory is \_\_\_\_\_.
  24. SRAMs are preferred for applications where \_\_\_\_\_ and \_\_\_\_\_ are more critical than \_\_\_\_\_ and \_\_\_\_\_.
  25. Combining a number of IC chips to increase the word size or capacity is called \_\_\_\_\_.

## **OBJECTIVE TYPE QUESTIONS**

1. The memory technology which needs the least power is  
(a) ECL                    (b) MOS                    (c) CMOS                    (d) none of these
  2. To construct a  $512\text{ K} \times 8$  memory, the number of  $32\text{ K} \times 4$  memory circuits required is  
(a) 16                    (b) 32                    (c) 8                            (d) 64
  3. To construct a  $128\text{ K} \times 16$  memory, the number of  $16\text{ K} \times 8$  memory circuits required is  
(a) 16                    (b) 32                    (c) 8                            (d) 64

**978** FUNDAMENTALS OF DIGITAL CIRCUITS



## PROBLEMS

1. Draw a logic diagram showing how to interconnect  $2\text{ K} \times 8$  memory circuits to obtain a  $4\text{ K} \times 8$  memory? Each circuit has an active-LOW chip select input and common data-in/data-out pins.
  2. Draw a logic diagram showing how to interconnect  $2\text{ K} \times 4$  memory circuits to obtain a  $2\text{ K} \times 8$  memory. Each circuit has an active-LOW chip select input and common data-in/data-out pins.
  3. Draw a logic diagram showing how to interconnect  $128 \times 4$  static RAMs to construct a  $256 \times 8$  memory. Each circuit has an active-LOW chip select input and common data-in/ data-out pins.

## VHDL PROGRAMS

## 1. VHDL PROGRAM FOR RAM

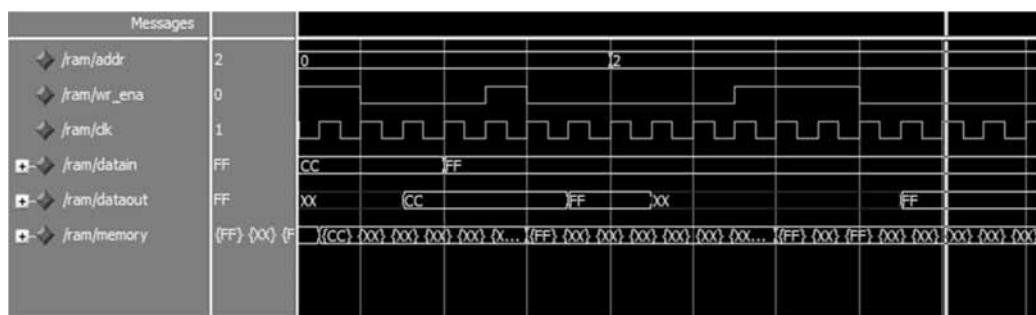
```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ram is
    generic ( bits : integer := 8; words : integer := 16);
    port (addr : in integer range 0 to (words-1); wr_ena, clk : in std_logic;
          datain : in std_logic_vector(bits-1 downto 0);
          dataout : out std_logic_vector(bits-1 downto 0));
end ram;
architecture Behavioral of ram is
    type vectorarray is array(0 to words-1) of
std_logic_vector(bits-1 downto 0);
    signal memory : vectorarray;
begin
    process(clk, wr_ena)
    begin

        if (clk = '1' and clk'event) then
            if(wr_ena = '1') then
                memory(addr) <= datain;
            else
                dataout <= memory(addr);
            end if;
        end if;
    end process;
end Behavioral;

```

## SIMULATION OUTPUT:



## 2. VHDL PROGRAM FOR ROM

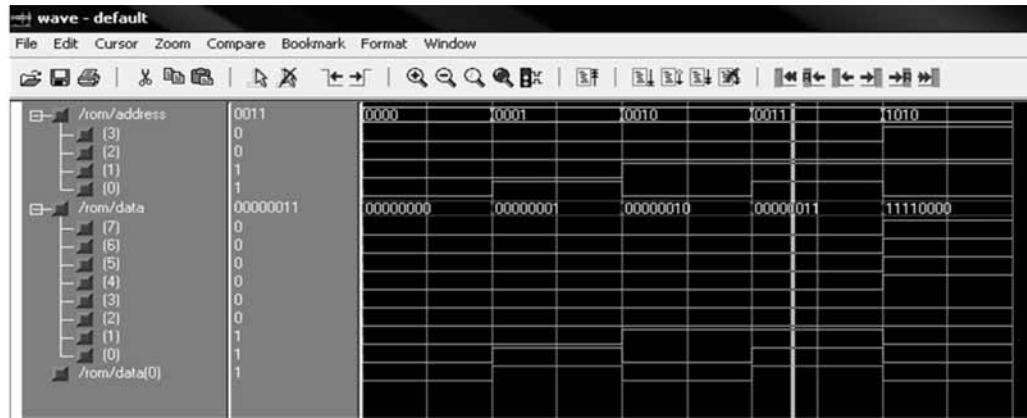
```

library ieee;
use ieee.std_logic_1164.all;
entity ROM is
    port ( address : in std_logic_vector(3 downto 0);
           data : out std_logic_vector(7 downto 0) );
end entity ROM;
architecture Behavioral of ROM is
    type mem is array ( 0 to 2**4 - 1 ) of std_logic_vector(7 downto
0);
    constant my_Rom : mem := (
        0 => "00000000",
        1 => "00000001",
        2 => "00000010",
        3 => "00000011",
        4 => "00000100",
        5 => "11110000",
        6 => "11110000",
        7 => "11110000",
        8 => "11110000",
        9 => "11110000",
        10 => "11110000",
        11 => "11110000",
        12 => "11110000",
        13 => "11110000",
        14 => "11110000",
        15 => "11110000");
begin
    process (address)
    begin
        case address is
            when "0000" => data <= my_rom(0);
            when "0001" => data <= my_rom(1);
            when "0010" => data <= my_rom(2);
            when "0011" => data <= my_rom(3);
            when "0100" => data <= my_rom(4);
            when "0101" => data <= my_rom(5);
            when "0110" => data <= my_rom(6);
            when "0111" => data <= my_rom(7);
            when "1000" => data <= my_rom(8);
            when "1001" => data <= my_rom(9);
            when "1010" => data <= my_rom(10);
            when "1011" => data <= my_rom(11);
        end case;
    end process;
end architecture;

```

```
when "1100" => data <= my_rom(12);
when "1101" => data <= my_rom(13);
when "1110" => data <= my_rom(14);
when "1111" => data <= my_rom(15);
when others => data <= "00000000";
end case;
end process;
end Behavioral;
```

### SIMULATION OUTPUT:



# 19

## TIMING CIRCUITS AND DISPLAY DEVICES

### 19.1 SCHMITT TRIGGER

A Schmitt trigger is not classified as a flip-flop, but it does exhibit a type of memory characteristic that makes it useful in certain special situations.

A Schmitt trigger inverter accepts slow changing signals and produces an output that has oscillation-free transitions. Figures 19.1a and f show a Schmitt trigger inverter and its response to a slow changing input. From the waveform of Figure 19.1f it can be noticed that the output does not change from HIGH to LOW until the input exceeds the positive-going threshold voltage  $V_{T+}$  (also called UTL) and then remains LOW even if the input falls below  $V_{T+}$  and changes from LOW to HIGH only when the input falls below the negative-going threshold voltage  $V_{T-}$  (also called LTL). Logic designers use ICs with Schmitt trigger inputs to convert slow-changing signals to clean, fast-changing signals that can drive standard IC inputs.

There are several ICs available with Schmitt trigger inputs. The 7414, 74LS14, and 74HC14 are hex inverter ICs with Schmitt trigger inputs. The 7413, 74LS13, and 74HC13 are dual 4-input NANDs with Schmitt trigger inputs. The 74132 contains four 2-input NAND gates with built-in Schmitt triggers.

The logic symbol for these devices shown in Figures 19.1a, b, and c contain a box-like symbol called the *hysteresis loop*, which represents the transfer characteristic (output voltage vs input voltage) of a device having hysteresis. The lower and upper trigger levels of TTL Schmitt triggers are fixed by design, therefore, hysteresis is not adjustable. Typical values are,  $V_{T+} = 1.7$  V and  $V_{T-} = 0.9$  V, giving a hysteresis of 0.8 V.

Figure 19.1d shows a Schmitt trigger inverter using an op-amp. Figure 19.1e shows the hysteresis loop.

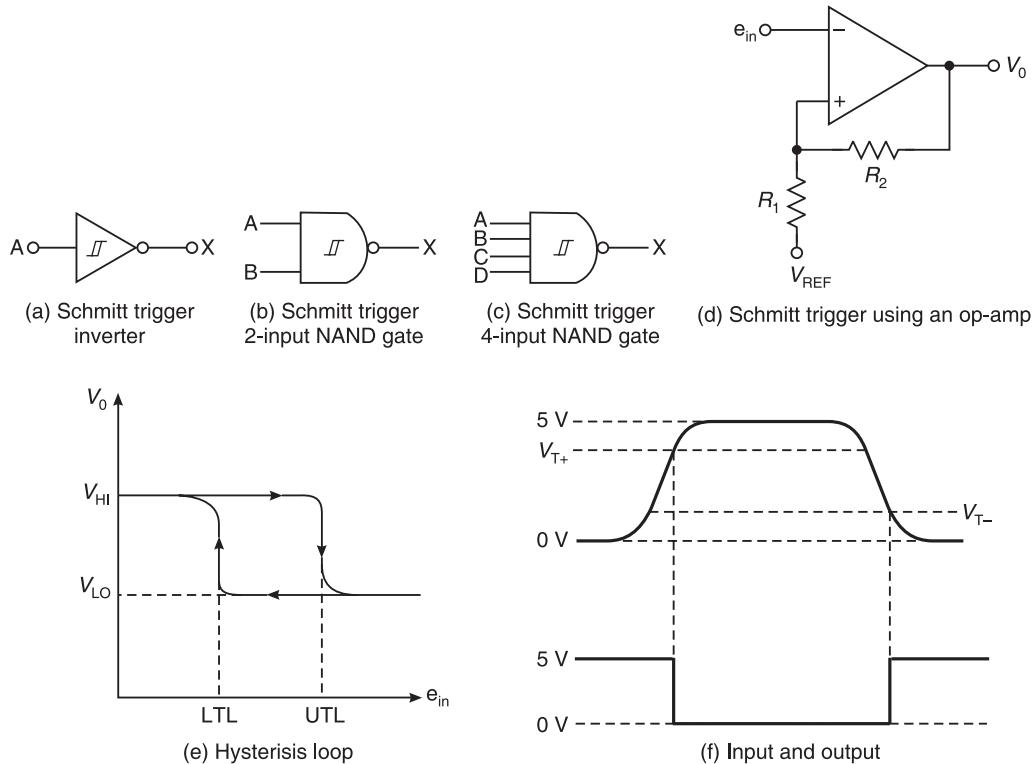


Figure 19.1 ICs with Schmitt trigger inputs.

## 19.2 MONOSTABLE MULTIVIBRATOR (ONE-SHOT)

There are three types of multivibrators: (1) Bistable multivibrator, commonly known as flip-flop, (2) monostable multivibrator, commonly known as one-shot or single-shot or simply monostable, and (3) astable multivibrator usually called free-running multivibrator.

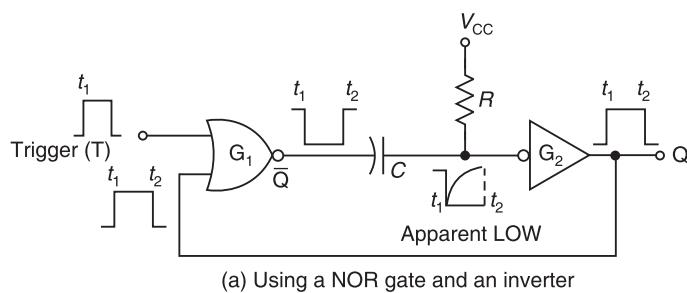
A bistable multivibrator (flip-flop) has two stable states. A stable state is a state in which the multivibrator can remain indefinitely. A triggering signal is required to change the state of the bistable multivibrator. It flip-flops (i.e. changes back and forth) between its two stable states when triggering pulses are applied.

A monostable multivibrator, as the name itself indicates, has only one stable state. The other state is quasi-stable. When triggered, it changes from its stable state (LOW state) to its quasi-stable state (HIGH state) and remains there for a specified length of time before returning automatically to its stable state, i.e. it produces a pulse of predetermined width in response to a trigger input. The trigger itself may be a pulse, whose LOW-to-HIGH or HIGH-to-LOW transition (depending on design) initiates the output pulse. The width of the output pulse is usually determined by the resistance and capacitance values in an RC network, called the timing circuit, connected to

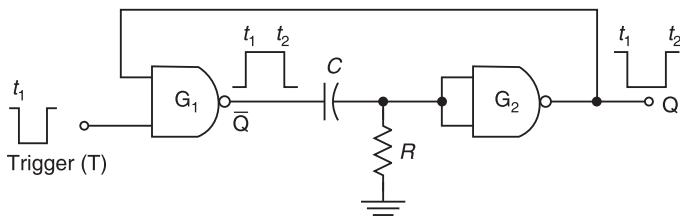
the device. A monostable multivibrator is often called simply a monostable, a one-shot or a single-shot because it produces a **single pulse** in response to a trigger input. The output of the monostable is labelled Q. Many monostables also have a output, the complement of Q which goes LOW when Q goes HIGH and vice versa.

Figure 19.2a shows a basic one-shot circuit composed of a NOR gate and an inverter. In the normal stable state, the input is LOW and Q output is also LOW. So, the output of the NOR gate ( $G_1$ ) is HIGH. When a positive triggering pulse is applied to  $G_1$ , the output of  $G_1$  goes LOW. This HIGH-to-LOW transition is coupled through the capacitor to the input to inverter ( $G_2$ ). The apparent LOW on  $G_2$  makes its output go HIGH. This HIGH is connected back into  $G_1$  keeping its output LOW. So, a trigger pulse has caused the Q output to go HIGH.

The capacitor immediately begins to charge through R towards the HIGH voltage level. The rate at which it charges is determined by the RC time constant. When the capacitor charges to a certain level which appears as a HIGH to  $G_2$ , the output goes back to LOW. So, a pulse of fixed time duration is generated at the Q terminal of the monostable circuit.



(a) Using a NOR gate and an inverter



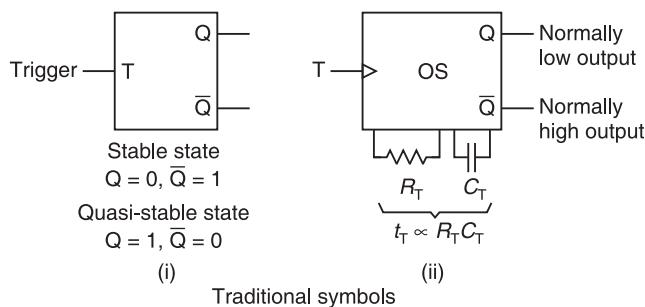
(b) Using NAND gates

Figure 19.2 Monostable circuits.

Figure 19.2b shows a monostable using NAND gates.  $G_1$  is a 2-input NAND gate.  $G_2$  is used as an inverter. Under the resting condition, voltage across R is zero which is the input to  $G_2$ . So, the output of  $G_2$  is HIGH. As both the inputs to  $G_1$  are HIGH, its output is LOW.

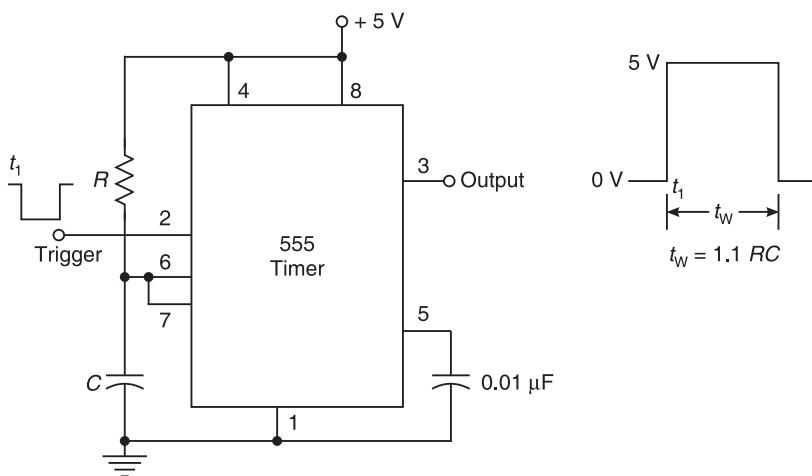
When a negative trigger is applied at  $G_1$ , the output of  $G_1$  goes HIGH. Since the voltage across C cannot change instantaneously, this appears as a HIGH at the input to  $G_2$ , and, therefore, its output goes LOW. As the capacitor charges, the voltage across R and, therefore, the input to  $G_2$  decreases with a time constant RC. When this voltage falls below a certain level, it appears as a LOW to  $G_2$ , and, therefore, its output goes back to HIGH. So, a triggering pulse causes the output of the circuit to go LOW for a specific time. Hence, the circuit acts as a monostable multivibrator.

Figure 19.3 shows the various logic symbols used for monostable multivibrators.



**Figure 19.3** Monostable multivibrators.

Figure 19.4 shows a monostable multivibrator using 555 timer. The 555 timer is TTL compatible. The output at pin 3 is normally in LOW state. When a negative triggering pulse is applied at pin 2, the output goes HIGH for a specific time ( $t_w = 1.1 RC$ ) and then comes back to its normal LOW state. So, a positive pulse of width  $t_w = 1.1 RC$  is generated. This is a non-retriggerable one-shot. Application of new trigger pulses during the timing cycle has no effect. However, the RESET input at pin 4 can be used to terminate an output pulse during the timing cycle, if desired.



**Figure 19.4** Monostable multivibrator using 555 timer.

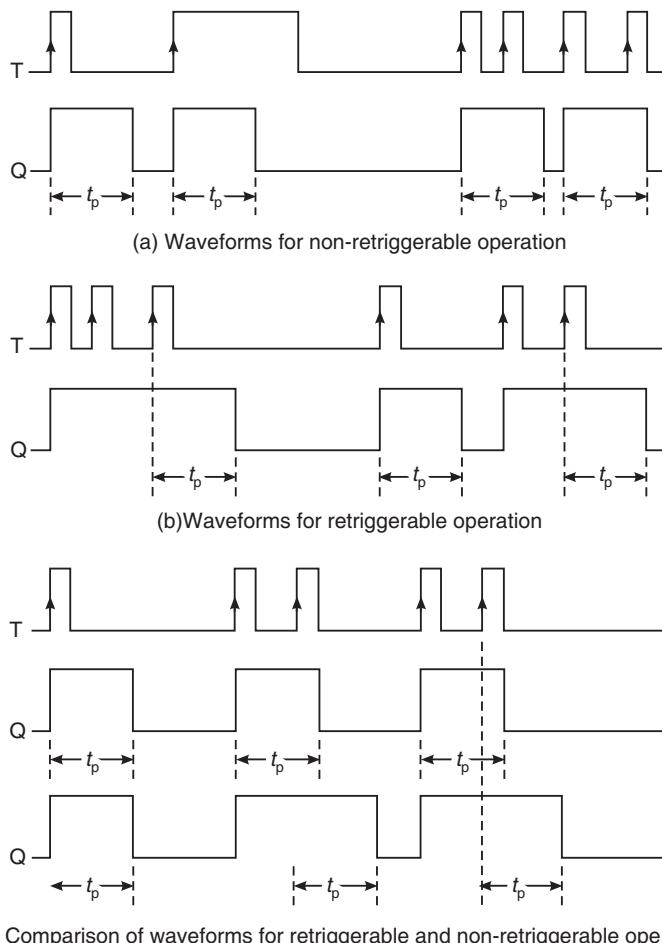
### 19.2.1 Integrated Circuit One-shots

Two types of one-shots are available in IC form—retriggerable type and non-retriggerable type. A retriggerable one-shot is one that accepts a new trigger input while the output pulse produced by the previous trigger is still in progress. The new trigger initiates a new timing cycle. So, the output pulse is extended, beginning from where the new trigger occurred, for a length of time equal to the monostable's full output pulse width. In other words, regardless of how long an output pulse had been HIGH, a new trigger input effectively restarts time and superimposes a new pulse beginning from where the trigger occurred. A non-retriggerable one-shot simply ignores any new trigger that occurs while a pulse output is in progress.

Figure 19.5a shows the waveforms for a non-retriggerable monostable multivibrator. From the waveforms, observe that if  $t_p$  is the pulse width of the one-shot, the output of the one-shot goes HIGH only for  $t_p$  whenever a positive going transition of the trigger occurs. Also, the duration of the triggering pulse is of no consequence. Also, a triggering pulse applied when the output is already HIGH, does not affect the output at all.

Figure 19.5b shows the waveforms for a retriggerable one-shot. Note that the output remains HIGH for a time  $t_p$  after the application of the trigger at any instant.

Figure 19.5c shows the comparison of waveforms for the outputs of retriggerable and non-retriggerable one-shots.



**Figure 19.5** Waveforms of non-retriggerable and retriggerable monoshots.

### 19.2.2 Actual Devices

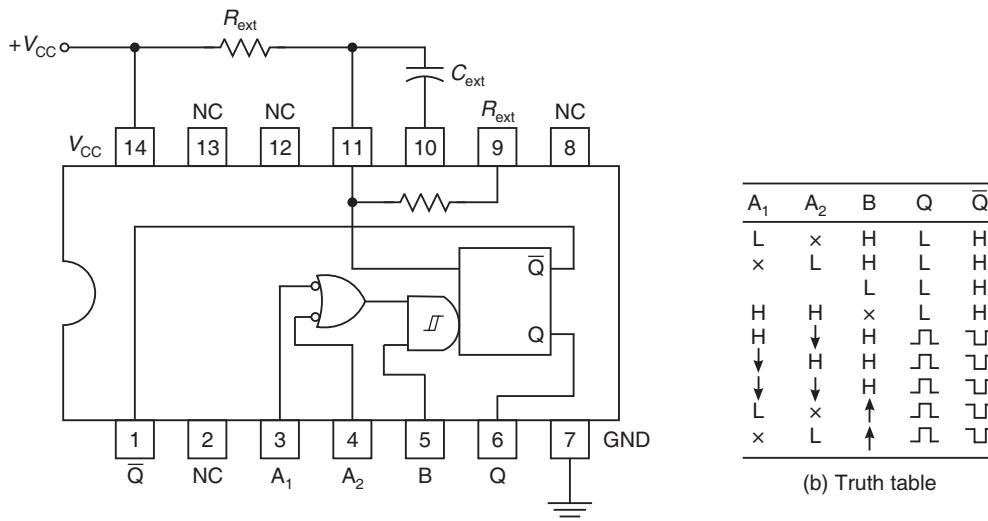
There are several one-shot ICs available in both the retriggerable and non-retriggerable versions. The 74121 and 74L121 are single non-retriggerable one-shot ICs. The 74221, 74LS221, and

74HC221 are dual non-retriggerable one-shot ICs; the 74122 and 74LS122 are single retriggerable one-shots. The 74123, 74LS123, and 74HC123 are dual retriggerable one-shot ICs.

The 74121 is a widely used non-retriggerable one-shot IC. It is constructed with logic gates at its inputs. Depending on how the external signal inputs are connected to these gates, the monostable can be triggered by a LOW-to-HIGH or by a HIGH-to-LOW level transition.

Figure 19.6a shows the wiring diagram of the 74121 with connections to the external  $RC$  timing circuit. The truth table governing the operation and triggering of this device is shown in Figure 19.6b. We see that the device can be triggered by a HIGH-to-LOW transition on input  $A_2$  when the other inputs ( $A_1$  and  $B$ ) are HIGH, or by a HIGH-to-LOW transition on  $A_1$ , when  $A_2$  and  $B$  are HIGH. It can also be triggered by a LOW-to-HIGH transition on  $B$  when either  $A_1$  or  $A_2$  is LOW. Note that the AND gate incorporates a Schmitt trigger to provide sharp triggering from slowly varying inputs. The width of the output pulse produced by the 74121 is given by

$$PW = (\ln 2)R_{\text{ext}}C_{\text{ext}} = 0.69 R_{\text{ext}}C_{\text{ext}}$$



(a) Wiring diagram showing connection of the external  $RC$  timing circuit

Figure 19.6 The 74121 non-triggerable one-shot IC.

Figure 19.7 shows the wiring diagram and the truth table for one-half of the dual 74123 retriggerable monostable multivibrator. Note that this version has a CLEAR input (CLR) which, when made LOW will cause an output pulse, already in progress, to be terminated.

When  $C_{\text{ext}} > 1000 \text{ pF}$ , the output pulse width is approximately given by

$$t_w \approx 0.28R_{\text{ext}}C_{\text{ext}}(1 + 0.7/R_{\text{ext}})$$

If  $C_{\text{ext}}$  is an electrolytic capacitor or if the CLR function is used, the manufacturer recommends that a diode be inserted between  $R_{\text{ext}}$  and pin 15 (cathode to pin 15), and the coefficient 0.28 in the equation be changed to 0.25.

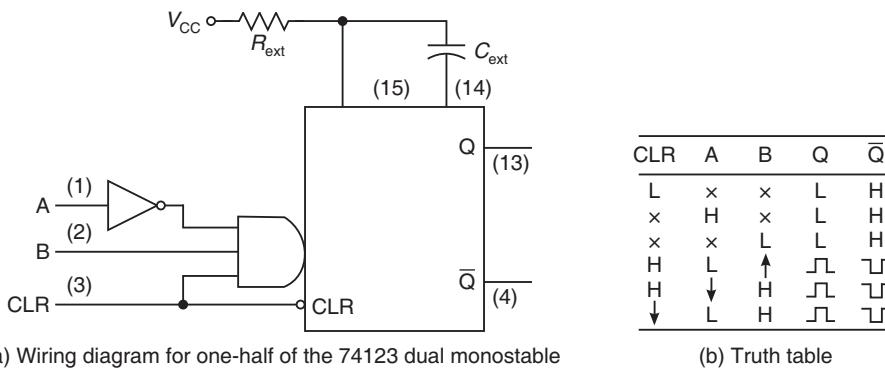


Figure 19.7 Dual 74123 retriggerable one-shot.

### 19.2.3 Applications of Monostable Multivibrators

Monostable multivibrators can be used for gating, creating time delays, generating a sequence of pulses, and for detection of missing pulses, etc.

**Gating.** In many digital systems, it is necessary to enable (disable) a logic gate to permit (stop) the passage of digital signals to another part of the system for a prescribed period of time. A monostable can be used to enable (or inhibit) logic gates for the necessary periods of time. The  $Q(\bar{Q})$  output of the monostable is connected as one input to an AND gate, whose other input is the clock signal. When a triggering signal is applied to the monostable, its  $Q$  output goes HIGH for a specific period of time enabling (disabling) the AND gate and thereby permitting (stopping) the passage of the clock signal.

**Time delays.** Monostables are widely used to deliver a pulse for a certain duration after the occurrence of another pulse, i.e. to create a prescribed time delay in delivering a pulse. For example, to delay a pulse of  $1\ \mu s$  by  $1\ ms$ , the  $1\ \mu s$  pulse is applied as the triggering signal to the first one-shot (of  $t_p = 1\ ms$ ) and the  $\bar{Q}$  output of the first one-shot is applied as the triggering input to the second one-shot (of  $t_p = 1\ \mu s$ ). The output of the second one-shot is then a  $1\ \mu s$  pulse delayed by  $1\ ms$ .

**Synchronizing.** Digital computer operations are often synchronized by sequences of pulses that occur on different control lines at different times. A sequence of  $N$  pulses can be generated by using  $N$  one-shots.

**Detection of missing pulses.** A retriggerable monostable can be used to detect a missing pulse or the cessation of pulses in a pulse train that is supposed to consist of a sequence of regularly recurring pulses. For this, the pulse width of the retriggerable monostable is set to between 1 and 2 periods of the pulse train. The pulse train continually re-triggers the monostable which never times out unless a pulse is missing or the pulse train ceases.

## 19.3 ASTABLE MULTIVIBRATOR

As the name indicates, an astable multivibrator is a multivibrator with no stable states. It has two states and both of them are quasi-stable. The moment it is connected to the supply it keeps on

switching back and forth (oscillating) between its quasi-stable states. Hence, it is also called a free-running multivibrator. It is useful for providing clock signals to synchronous digital circuits.

There are several types of astable multivibrators in common use. Some of them are presented here.

### 19.3.1 Astable Multivibrator Using Schmitt Trigger

Figure 19.8a shows how a Schmitt trigger INVERTER can be connected as an oscillator. The signal at  $V_{\text{OUT}}$  is an approximate square wave (Figure 19.8b) that depends on the values of  $R$  and  $C$ . The relation between the frequency of oscillation and the  $RC$  product is shown in Figure 19.8c for three different Schmitt trigger ICs. The maximum permitted value of  $R$  in each case is also given. The circuit will fail to oscillate if  $R$  is not kept below these limits.

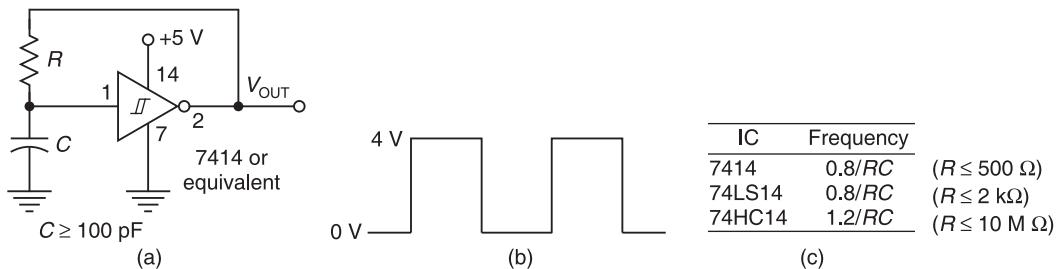


Figure 19.8 Astable multivibrator using Schmitt trigger.

### 19.3.2 Astable Multivibrator Using 555 Timer

The 555 timer is a TTL compatible device that can operate in several different modes. Figure 19.9 shows an astable multivibrator using the 555 timer. Its output is a repetitive rectangular waveform that switches between two logic levels; the time interval for each logic level is determined by the

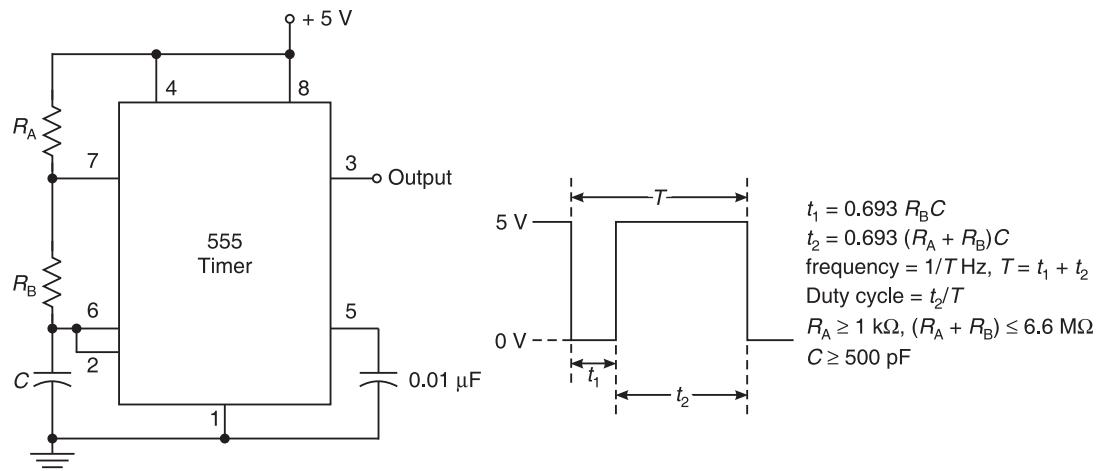


Figure 19.9 Astable multivibrator using 555 timer.

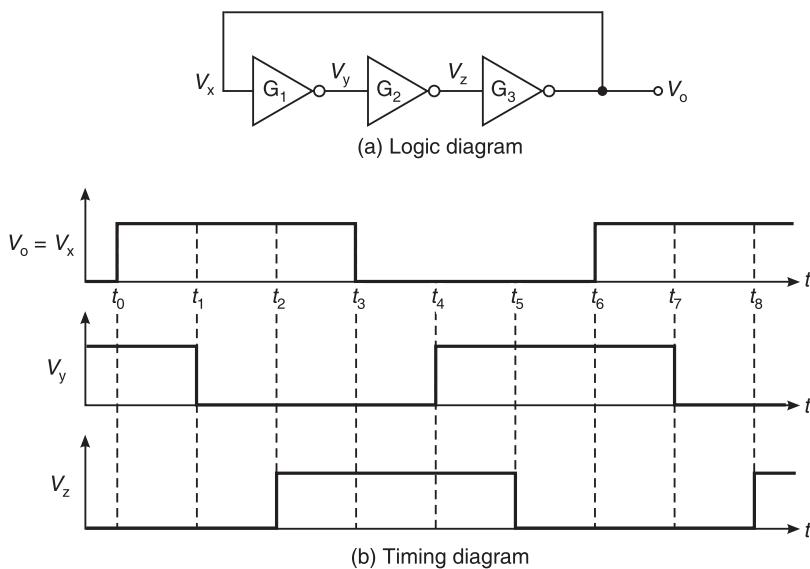
values of  $R$  and  $C$ . The formulas for these time intervals  $t_1$  and  $t_2$  and the overall period of the oscillations and the limiting values of the components are shown in Figure 19.9. The capacitor  $C$  charges from  $V_{CC}$  through  $R_A$  and  $R_B$  with a time constant  $(R_A + R_B)C$ . When the voltage across  $C$  reaches  $(2/3)V_{CC}$ , the output goes LOW, and the capacitor starts discharging through  $R_B$ , with a time constant  $R_B C$ . When it discharges to  $(1/3)V_{CC}$ , the output goes HIGH. So, the output is LOW for  $t_1$  and HIGH for  $t_2$  and then this cycle repeats itself. With this arrangement, we can only get a square wave with more than 50 per cent duty cycle.  $R_B$  can be made very large compared to  $R_A$  to get an approximate square wave (50 per cent duty cycle). A diode can be connected across  $R_B$  (with anode at pin 7 and cathode at pin 6) to get a perfect square wave output. *Even a square wave with less than 50 per cent duty cycle can be obtained when  $R_B$  is shunted by a diode.*

### 19.3.3 Astable Multivibrator Using Inverters

A very simple astable multivibrator can be constructed using three inverters. The inverters are connected in cascade and the output of the third inverter is connected as the input to the first inverter as shown in Figure 19.10a. Hence the circuit is called a *ring oscillator*.

We know that the output of an inverter changes to the opposite state after a propagation delay when its input is changed. So, the signal at any point in this circuit changes state after a time equal to the sum of the propagation delays of the preceding gates and, therefore, a square wave is generated.

If at time  $t_0$ ,  $V_x$  goes from LOW to HIGH,  $V_y$  will go from HIGH to LOW after a propagation delay of one gate,  $V_z$  will go from LOW to HIGH after a propagation delay of two gates, and  $V_o$ , i.e.  $V_x$  will go from HIGH to LOW after a propagation delay of three gates and the process continues as shown in Figure 19.10b and a square wave is thus generated.



**Figure 19.10** Astable multivibrator using only inverters.

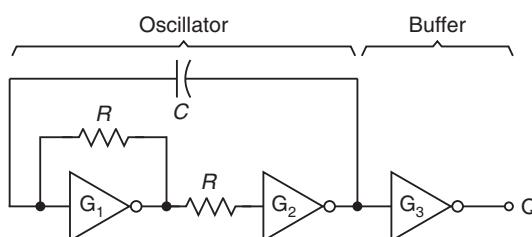
A major drawback of this circuit is that the frequency of the square wave cannot be controlled. It is difficult to determine the exact propagation delay of each logic gate and, therefore, the frequency

of the square wave. It is possible to have some control over the frequency of the square wave by using timing elements such as resistors and capacitors.

Figure 19.11 shows a simple, reliable, and highly flexible astable circuit. In this case, the frequency of oscillation is determined primarily by the resistor and capacitor timing components. Hence it is called an *RC* oscillator. The exact relationship between the oscillation frequency and the *R* and *C* components depends in part on the electrical characteristics of the logic gates. For the standard TTL family of components, a resistance of approximately  $400\ \Omega$  produces the relationship

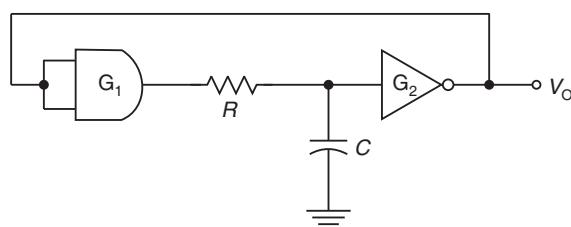
$$f = \frac{1}{0.001C}$$

where *C* is measured in  $\mu\text{F}$  and *f* in Hz.



**Figure 19.11** Astable multivibrator using *R* and *C* as timing components.

Figure 19.12 shows an astable multivibrator using an AND gate and an inverter. Both the inputs to the AND gate are shorted. So, it just provides a time delay equal to the propagation delay time of the gate and acts as a buffer. When the supply is switched on and as the capacitor is uncharged, the input to the inverter is 0 V. Therefore, its output is HIGH. Hence the output of the AND gate also goes HIGH. The capacitor now starts charging and when the voltage across it rises to a level which appears as a logic 1 to the inverter, the output goes LOW. Thus, the output of the AND gate also goes LOW. Now the capacitor starts discharging. When the voltage across it falls to a level which appears as a logic 0 to the inverter, the output of the inverter goes HIGH again. This cycle of events repeats itself, generating a square wave.



**Figure 19.12** Astable multivibrator using an AND gate and an inverter.

Figure 19.13a shows an astable multivibrator using two inverters. The output of each inverter is coupled to the input of the other through a capacitor. The capacitive coupling networks prevent either inverter from having a stable state. If designed properly, the circuit will start oscillating on its own and requires no initial input trigger.

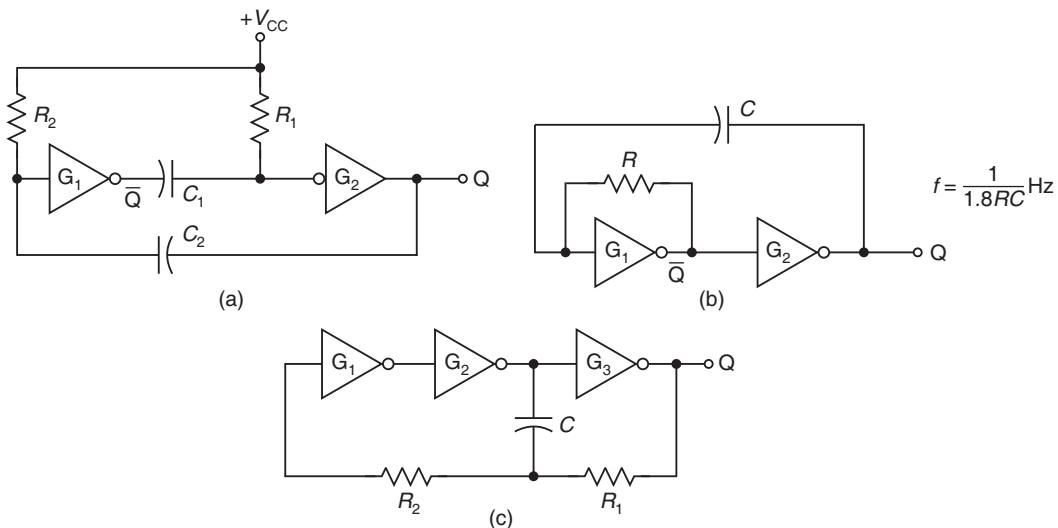


Figure 19.13 Astable multivibrator using CMOS inverters.

Figure 19.13b shows another astable multivibrator circuit using inverters. The device used is IC 74HC04, CMOS inverter.

Figure 19.13c uses the IC 74HC04. Although this circuit can operate reliably, its frequency is not easily predictable.

The advantage of using CMOS devices in astable multivibrators is that, they have a much higher input impedance than that of their TTL counterparts. This characteristic makes the performance of CMOS multivibrators more predictable than that of TTL designs—in the sense that CMOS multivibrators are less sensitive to variations in device characteristics.

#### 19.3.4 Astable Multivibrator Using Op-amps

Figure 19.14 shows an astable multivibrator using an op-amp. Assuming that the maximum positive and negative outputs of the comparator are  $\pm V_{\max}$ , capacitor  $C$  continually charges and

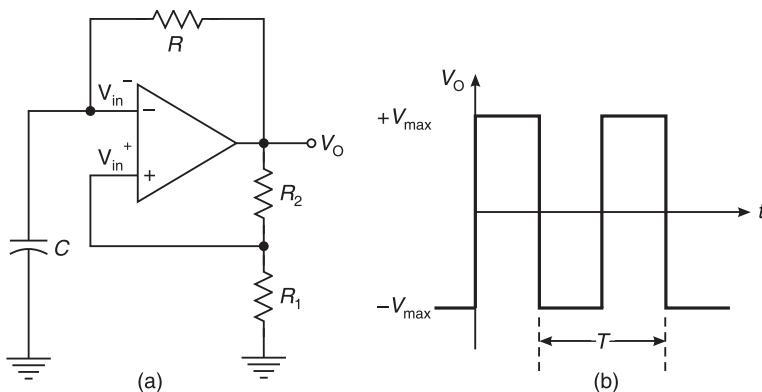


Figure 19.14 Astable multivibrator using op-amp.

discharges towards those values, and so, the output of the circuit will be a square wave with a period

$$T = 2RC \ln \left( \frac{1 + \beta}{1 - \beta} \right) \text{ s, where } \beta = \frac{R_1}{R_1 + R_2}$$

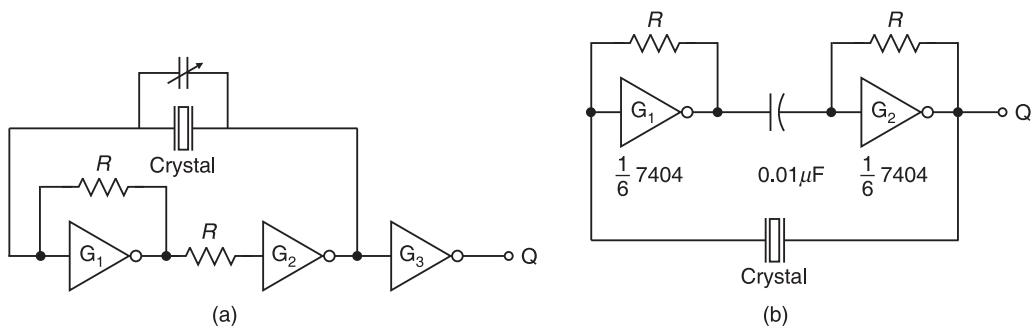
## 19.4 CRYSTAL-CONTROLLED CLOCK GENERATORS

A clock generator is a source for the square wave that synchronizes operations in digital systems. Some clock generators produce a reasonably sharp square wave, such as an astable multivibrator and others produce sinusoidal or rounded outputs that can be shaped by Schmitt triggers or clipping circuits. The term astable multivibrator is usually reserved for circuits whose operation depends on the charging and discharging of a capacitor, whereas an oscillator refers to any unstable system whose output continually changes value.

Many clock generators employ a crystal as the frequency sensitive component. Crystals are available in a wide range of frequencies and are more stable than inductor/capacitor networks. Crystal-controlled oscillators have good frequency stability, i.e. they generate signals whose frequency is less likely to drift. Also, since crystal-controlled oscillators produce signals whose frequencies equal their crystal frequency, they are predictable. The frequency stability is particularly important in applications where the frequency serves as a time reference. For example in digital watches.

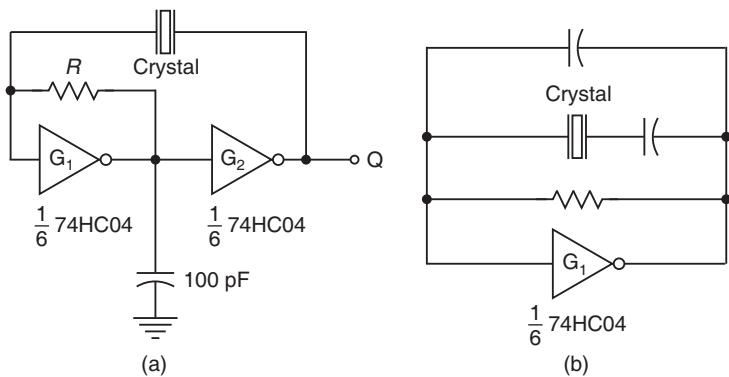
Figure 19.15a shows a simple quartz crystal oscillator using TTL 7404 inverters. The capacitor in parallel with the crystal is a trimming capacitor which allows very slight frequency adjustments.

Figure 19.15b shows a popular crystal-controlled clock generator constructed by using TTL 7404 inverters. Although the oscillations produced by this design are not perfectly square, the output is generally adequate for TTL synchronizing and triggering functions. The 7414 hex Schmitt trigger inverter can also be used. The value of  $R$  in the circuit controls the gain and is usually between  $300 \Omega$  and  $1.5 \text{ k}\Omega$ . The optimum value of  $R$  depends on the type of the crystal used and its frequency. If  $R$  is too low or too high, the generator may oscillate at a harmonic of the crystal frequency and thus have a smaller amplitude. This design has been used to produce clock frequencies from about 1 MHz to 20 MHz.



**Figure 19.15** Crystal clock generator using TTL inverters.

Figure 19.16 shows two crystal-controlled oscillators constructed with inverters from 74HC04 CMOS hex inverters. As in the TTL circuit of Figure 19.15, the oscillation frequencies may be sensitive to the values of  $R$  used. A typical value for  $R$  in Figure 19.16a is  $100\text{ k}\Omega$  but it may have to be specially selected to prevent oscillations at a harmonic frequency. The  $100\text{ pF}$  capacitor suppresses the spurious high frequency oscillations in the  $30\text{ MHz}$  to  $50\text{ MHz}$  range. The resistor in Figure 19.16b is of the order of  $1\text{--}5\text{ M}\Omega$  and the circuit will oscillate for crystal frequencies up to about  $9\text{ MHz}$ .



**Figure 19.16** Crystal clock generator using CMOS inverters.

**EXAMPLE 19.1** Calculate the frequency and duty cycle of the 555 astable multivibrator output for

$$C = 0.01\text{ }\mu\text{F}, \quad R_A = 10\text{ k}\Omega \quad \text{and} \quad R_B = 50\text{ k}\Omega.$$

**Solution**

$$t_1 = 0.693R_B C = 0.693 \times 50 \times 10^3 \times 0.01 \times 10^{-6} = 346.5\text{ }\mu\text{s}$$

$$t_2 = 0.693(R_A + R_B)C = 0.693 \times 60 \times 10^3 \times 0.01 \times 10^{-6} = 415.8\text{ }\mu\text{s}$$

$$T = t_1 + t_2 = (346.5 + 415.8)\text{ }\mu\text{s} = 762.3\text{ }\mu\text{s}$$

$$f = \frac{1}{T} = \frac{1}{762.3\text{ }\mu\text{s}} = 1.31\text{ kHz}$$

$$\text{Duty cycle} = \frac{t_2}{T} = \frac{415.8}{762.3} = 54.6\%$$

**EXAMPLE 19.2** Design an astable multivibrator using the 555 timer to generate a square wave of  $2\text{ kHz}$  frequency with  $50\%$  duty cycle.

**Solution**

To obtain a square wave using the 555 timer, a diode is connected across  $R_B$  (see Figure 19.9), such that it conducts and shorts  $R_B$  when  $C$  is charging and opens when  $C$  is discharging. Therefore,

$$t_1 = 0.693 R_B C$$

$$t_2 = 0.693 R_A C$$

$$\text{As } t_1 = t_2, R_A = R_B$$

As  $f = 2 \text{ kHz}$ ,

$$T = \frac{1}{2 \text{ kHz}} = 0.5 \text{ ms}$$

Hence,

$$t_1 = t_2 = \frac{T}{2} = 0.25 \text{ ms}$$

Let  $R_A = 2 \text{ k}\Omega$ ,

therefore,

$$C = \frac{0.25 \times 10^{-3}}{0.693 \times 2 \times 10^3} = 0.18 \mu\text{F}$$

**EXAMPLE 19.3** Design an astable multivibrator using the 555 timer to generate a square wave of 5 kHz with 70 per cent duty cycle.

**Solution**

$$T = \frac{1}{f} = \frac{1}{5 \text{ kHz}} = 0.2 \text{ ms}$$

$$t_1 = 0.693R_B C = 0.3 \times 0.2 = 0.06 \text{ ms}$$

$$t_2 = 0.693(R_A + R_B)C = 0.7 \times 0.2 = 0.14 \text{ ms}$$

Let  $C = 5000 \text{ pF}$ ,

therefore,

$$R_B = \frac{0.06 \times 10^{-3}}{0.693 \times 5000 \times 10^{-12}} = 17.25 \text{ k}\Omega$$

$$R_A + R_B = \frac{0.14 \times 10^{-3}}{0.693 \times 5000 \times 10^{-12}} = 40.4 \text{ k}\Omega$$

Therefore,

$$R_A = 40.4 - 17.25 = 23.15 \text{ k}\Omega$$

**EXAMPLE 19.4** For the astable multivibrator using the op-amp shown in Figure 19.14,

$$R = 10 \text{ k}\Omega, \quad C = 0.01 \mu\text{F} \quad \text{and} \quad R_1 = 5 \text{ k}\Omega.$$

It is desired to make the frequency of the output square wave adjustable from 10 kHz through 100 kHz by making  $R_2$  adjustable. Through what range of values should  $R_2$  be made adjustable to obtain the required frequency range?

**Solution**

The period of the output square wave must range from

$$T = \frac{1}{10 \text{ kHz}} = 0.1 \text{ ms} \quad \text{through} \quad T = \frac{1}{100 \text{ kHz}} = 0.01 \text{ ms}$$

We know that,

$$T = 2RC \ln\left(\frac{1+\beta}{1-\beta}\right) \text{ s, where } \beta = \frac{R_1}{R_1 + R_2}.$$

## 996 FUNDAMENTALS OF DIGITAL CIRCUITS

Therefore,

$$T = 10^{-4} \text{ s} = 2 \times 10 \times 10^3 \times 0.01 \times 10^{-6} \ln\left(\frac{1+}{1-}\right)$$

or  $\ln\left(\frac{1+}{1-}\right) = \frac{10^{-4}}{2 \times 10^4 \times 0.01 \times 10^{-6}} = 0.5$

or  $\left(\frac{1+}{1-}\right) = e^{0.5} = 1.6487$

or  $\beta = 0.25 = \frac{R_1}{R_1 + R_2} = \frac{5 \text{ k}\Omega}{5 \text{ k}\Omega + R_2}$

or  $R_2 = 15 \text{ k}\Omega$

Again,

$$T = 10^{-5} \text{ s} = 2 \times 10 \times 10^3 \times 0.01 \times 10^{-6} \ln\left(\frac{1+}{1-}\right)$$

or  $\ln\left(\frac{1+}{1-}\right) = \frac{10^{-5}}{2 \times 10^4 \times 0.01 \times 10^{-6}} = 0.05$

or  $\left(\frac{1+}{1-}\right) = e^{0.05} = 1.05$

or  $\beta = 0.025 = \frac{R_1}{R_1 + R_2} = \frac{5 \text{ k}\Omega}{5 \text{ k}\Omega + R_2}$

or  $R_2 = 192 \text{ k}\Omega$

Hence the range of  $R_2$  must be  $15 \text{ k}\Omega$  through  $192 \text{ k}\Omega$ .

## 19.5 DISPLAY DEVICES

Display devices provide a visual display of numbers, letters, and symbols in response to electrical input, and serve as constituents of an electronic display system.

### 19.5.1 Classification of Displays

The commonly used displays in the digital electronic field are as follows:

1. Cathode Ray Tube (CRT)
2. Light Emitting Diode (LED)
3. Liquid Crystal Display (LCD)
4. Gas Discharge Plasma Display (Cold-cathode Display or Nixie)
5. Electro-luminescent (EL) Display

6. Incandescent Display
7. Electrophoretic Image Display (EPID)
8. Liquid Vapour Display (LVD)

In general, displays are classified in a number of ways, such as follows:

1. On the methods of conversion of electrical data to visible light:
  - (a) Active displays (light emitters—CRTs, Gas discharge plasma, LEDs, etc.)
  - (b) Passive displays (light controllers—LCDs, EPIDs, etc.)
2. On the applications:
  - (a) Analog displays—Bargraph displays (CRT)
  - (b) Digital displays—Nixies, Alphanumeric, LEDs, etc.
3. According to the display size and physical dimensions:
  - (a) Symbolic displays—Alphanumeric, Nixie tubes, LEDs, etc.
  - (b) Console displays—CRTs, LCDs, etc.
  - (c) Large screen displays—Enlarged projection system.
4. According to the display format:
  - (a) Direct view type (flat panel planar)—Segmental dot matrix, CRTs
  - (b) Stacked electrode (non-planar type)—Nixie
5. In terms of resolution and legibility of characters:
  - (a) Simple, single-element indicator
  - (b) Multi-element displays.

### **19.5.2 The Light Emitting Diode (LED)**

In a forward-biased diode, free electrons cross the junction and fall into holes. When they recombine, these free electrons radiate energy as they fall from a higher energy level to a lower one. In a rectifier diode, the energy is dissipated as heat, but in an LED, the energy is radiated as light. By using elements like gallium, arsenic, and phosphorous, a manufacturer can produce LEDs that radiate red, green, yellow, orange, and infrared radiation (invisible). LEDs that produce visible radiation are used in instrument displays, calculators, digital watches, etc.

The infrared (IR) LED finds applications in burglar-alarm systems and other areas requiring invisible radiation. Generally, the infrared emitting LEDs are coated with phosphor so that, by the excitation of phosphor, visible light can be produced.

The advantages of using LEDs in electronic displays are:

1. LEDs are very small and can be considered as point sources of light. They can, therefore, be stacked in a high density matrix to serve as a numeric and alphanumeric display.
2. The light output from an LED is a function of current flowing through it. An LED can, therefore, be controlled smoothly by varying the current. This is particularly useful for operating LED displays under different ambient lighting conditions.
3. LEDs are highly efficient emitters of EM radiation. LEDs with light output of different colours, i.e. red, yellow, amber, and green are commonly available.
4. LEDs are very fast devices, having a turn-on/off time of less than 1 ms.
5. The low supply voltage and current requirements of LEDs make them compatible with TTL ICs.

6. LEDs are manufactured with the same type of technology as that used for transistors and ICs and, therefore, they are economical and have a high degree of reliability.
7. LEDs are rugged and can, therefore, withstand shocks and vibrations. They can be operated over a wide range of temperature say, 0–70°C.

The disadvantage of LEDs compared to LCDs is their high power requirement. Also, LEDs are not suited for large area displays, primarily because of their high cost.

### 19.5.3 The Liquid Crystal Display (LCD)

A liquid crystal is a state of matter between a solid and a liquid. The characteristic feature of a liquid crystal is its long cylindrical molecules. The alignment of molecules can exist only over a limited temperature range of 0–50°C with most available devices.

Liquid crystal displays (LCDs) are used in similar applications where LEDs are used. These applications are display of numeric and alphanumeric characters in dot matrix and segmental displays.

LCDs are of two types—dynamic scattering type and field effect type. The dynamic scattering liquid crystal cell is constructed by layering the liquid crystal between glass sheets with transparent electrodes deposited on the inside faces. The liquid crystal material may be one of the several organic compounds which exhibit optical properties of a crystal, though they remain in liquid form. When a potential is applied across the cell, charge carriers flowing through a liquid disrupt the molecular alignment and produce turbulence. When the liquid is not activated, it is transparent. When the liquid is activated, the molecular turbulence causes light to be scattered in all directions and the cell appears to be bright. The phenomenon is called *dynamic scattering*.

The construction of a field effect liquid crystal display is similar to that of the dynamic scattering type, with the exception that two thin polarizing optical filters are placed at the inside of each glass sheet. The liquid crystal material in the field effect cell is also of different type from that employed in the dynamic scattering cell. The material used is twisted nematic type and it actually twists the light passing through the cell when the latter is not energized. This allows the light to pass through the optical filters and the cell appears bright. When the cell is energized, no twisting of light takes place and the cell appears dull.

Liquid crystal cells are of two types—transmittive type and reflective type. In the transmittive type cell, both glass sheets are transparent, so that light from a rear source is scattered in the forward direction when the cell is activated. The reflective type cell has a reflective surface on one side of the glass sheets. The incident light on the front surface of the cell is dynamically scattered by an activated cell. Both types of cells appear quite bright when activated even under ambient light conditions.

The liquid crystals are light reflectors or transmitters and, therefore, consume small amounts of energy.

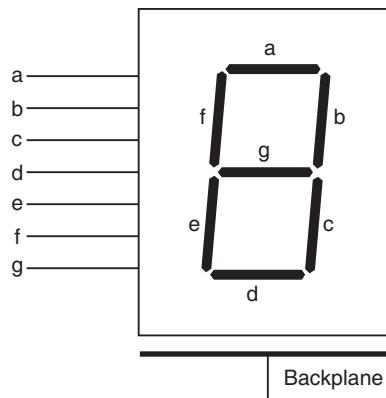
The advantages of LCDs are low power consumption and low cost. The disadvantages are that they occupy a large area and their operating speed is low. LCDs are normally used for seven segment displays.

### 19.5.4 Operation of Liquid Crystal Displays

LCDs operate by polarizing light so that a non-activated segment reflects incident light and thus appears invisible against its background. An activated segment does not reflect incident light and

thus appears dark. LCDs consume much less power than LED displays and are widely used in battery powered devices such as calculators and watches. An LCD does not emit light energy like an LED. So, it cannot be seen in the dark. It requires an external source of light.

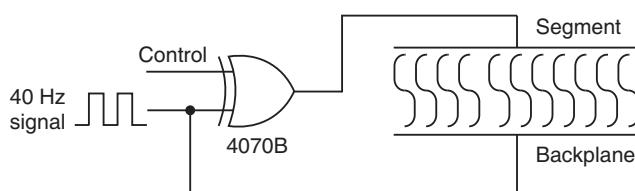
Basically, LCDs operate from a low voltage (typically 3–15 V rms), low frequency (20–60 Hz) ac signal and draw very little current. They are often arranged as seven segment displays for numerical read-outs as shown in Figure 19.17. The ac voltage needed to turn on a segment is applied between the segment and the backplane which is common to all segments. The segment and the backplane form a capacitor, that draws very little current as long as the ac frequency is kept low. The frequency is generally not lower than 25 Hz, because that would produce visible flicker.



**Figure 19.17** Liquid crystal display.

### Driving an LCD

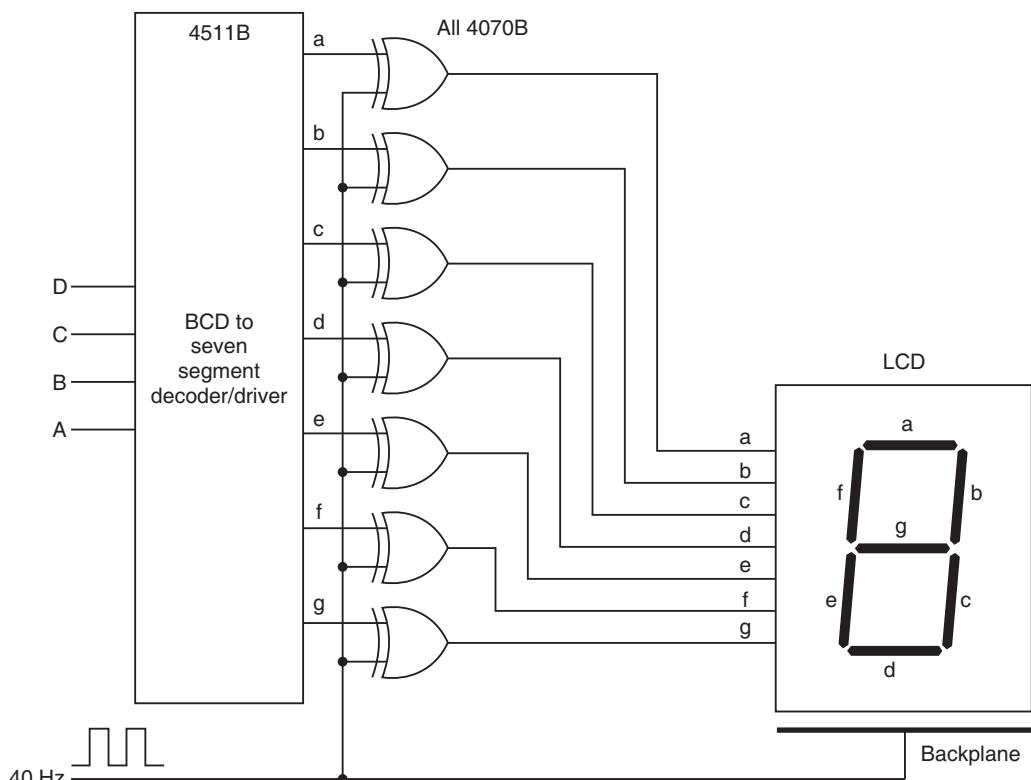
An LCD segment will turn on, when an ac voltage is applied between the segment and the backplane, and will turn off, when there is no voltage between the two. Rather than generating an ac signal, it is a common practice to produce the required ac voltage by applying out-of-phase square waves to the segment and the backplane. Figure 19.18 illustrates the driving arrangement for one segment. A 40 Hz square wave is applied to the backplane and also to the input of a CMOS 4070 X-OR gate. The other input to the X-OR is a control input, that controls the ON/OFF state of the segment.



**Figure 19.18** Method for driving the seven segment LCD.

When the CONTROL input is LOW, the X-OR output will be exactly the same as the 40 Hz square wave, so that the signals applied to the segment and the backplane are equal. Since there is no difference in voltage, the segment will be OFF. When the CONTROL input is HIGH, the X-OR output will be the inverse of the 40 Hz square wave, so that the signal applied to the segment is

out-of-phase with the signal applied to the backplane. As a result, the segment voltage will alternatively be at + 5 V and – 5 V relative to the backplane. This ac voltage will turn on the segment. The same idea can be extended to a complete seven segment LCD display as shown in Figure 19.19.



**Figure 19.19** Circuit for driving the seven segment LCD.

In general, CMOS devices are used to drive LCDs for two reasons. First, they require much less power than that by TTL and are more suited to the battery-operated applications where LCDs are used. Second, as the TTL LOW state voltage is not exactly 0 V, and can be as much as 0.4 V, it produces a dc component of voltage between the segment and the backplane which considerably shortens the life of an LCD.

### 19.5.5 Incandescent Seven Segment Displays

Incandescent displays use light-bulb filaments as the segments. When current is made to flow through these filaments, they become hot and thus glow. They emit a bright white light that is easy to read. They are sometimes covered with coloured filters to provide coloured light. These displays are much less efficient than the LED types because they require more current per segment, and therefore, are not used in battery-operated devices such as calculators and multimeters. They are, however, used in electronic cash registers and other line-operated devices where power consumption is not critical.

**SHORT QUESTIONS AND ANSWERS**

- 1.** What is the use of a Schmitt trigger inverter?  
 A. A Schmitt trigger inverter is used to convert slow changing input signals to clean fast changing output signals with oscillation free transitions.
- 2.** How many types of multivibrators are there? Name them.  
 A. There are three types of multivibrators . They are:
  1. bistable multivibrator commonly known as flip-flop.
  2. monostable multivibrator commonly known as one shot or single shot or simply monostable.
  3. astable multivibrator usually called free running multivibrator.
- 3.** How many stable states do the multivibrators have?  
 A. As the name indicates a bistable multivibrator has two stable states; a monostable multivibrator has only one stable state and the astable multivibrator has no stable state at all.
- 4.** What is a quasi-stable state?  
 A. A quasi-stable state means a temporarily stable state. When brought into this stage, the circuit remains in this state only for a short interval of time and afterwards it comes out of it without the need of any triggering signal.
- 5.** What are the two types of IC one shots?  
 A. The two types of IC one shots are:
 

1. retriggerable type and	2. non-retriggerable type
---------------------------	---------------------------
- 6.** What is a retriggerable one shot?  
 A. A retriggerable one shot is a one shot in which a new trigger always initiates a new timing cycle.
- 7.** What is a non-retriggerable one shot?  
 A. A non-retriggerable one shot is a one shot that simply ignores any new trigger that occurs while a pulse output is in progress.
- 8.** What are the applications of monostable multivibrators?  
 A. A monostable multivibrator can be used:
  1. As a gating circuit
  2. To provide time delay
  3. For synchronizing digital operations
  4. For detection of missing pulses
- 9.** What is the use of an astable multivibrator?  
 A. The astable multivibrator is used as a master oscillator for providing clock signals to synchronous digital circuits.
- 10.** What is the advantage of crystal controlled oscillators?  
 A. Crystal controlled oscillators have the advantage of good frequency stability. Also their frequency is predictable because their frequency is equal to the frequency of the crystal.
- 11.** What is the use of display devices?  
 A. Display devices provide a visual display of numbers, letters, and symbols in response to electrical input and serve as constituents of an electronic display system.
- 12.** What are the two types of LED displays?  
 A. The two types of LED displays are:
 

1. common anode type and	2. common cathode type.
--------------------------	-------------------------

## **1002 FUNDAMENTALS OF DIGITAL CIRCUITS**

- 13.** What are the disadvantages of LEDs compared to LCDs ?  
**A.** The disadvantage of LEDs compared to LCDs is their high power requirement. Also LEDs are not suited for large area displays, primarily because of their high cost.
- 14.** What are the advantages and disadvantages of LCDs?  
**A.** The advantages of LCDs are low power consumption and low cost. The disadvantages are that they occupy a large area and their operating speed is low.
- 15.** What are the two types of LCDs?  
**A.** The two types of LCDs are:
  1. dynamic scattering type and
  2. field effect type
- 16.** Why an LCD cannot be seen in the dark like an LED?  
**A.** An LCD does not emit light energy. So it cannot be seen in the dark like an LED.
- 17.** Why are LCDs used in calculators and watches?  
**A.** LCDs consume much less power than LED displays do and are, therefore, widely used in battery-powered devices such as calculators and watches.

### **REVIEW QUESTIONS**

- 1.** What is the difference between the retriggerable and non-retriggerable one shots?
- 2.** Describe the operation and applications of monostable multivibrators, astable multivibrators, bistable multivibrators, and Schmitt trigger inverters?
- 3.** Explain the difference in operation of a monostable and an astable multivibrator.
- 4.** What are the advantages of a crystal clock generator?
- 5.** Name the commonly used displays in the digital electronic field.
- 6.** How are displays classified?
- 7.** What are the applications of LEDs?
- 8.** What are the advantages of using LEDs in electronic displays?
- 9.** Explain the working of dynamic scattering type LCD.
- 10.** Explain the working of field effect type LCD.

### **FILL IN THE BLANKS**

- 1.** A \_\_\_\_\_ converts a slow changing signal into a fast changing signal.
- 2.** \_\_\_\_\_ represents the transfer characteristic of a device having hysteresis.
- 3.** A state in which the multivibrator can remain permanently is called a \_\_\_\_\_ state.
- 4.** A bistable multivibrator has both \_\_\_\_\_ states.
- 5.** A monostable multivibrator has one \_\_\_\_\_ state and one \_\_\_\_\_ state.
- 6.** An astable multivibrator has both \_\_\_\_\_ states.
- 7.** The two types of IC one shots are 1. \_\_\_\_\_ and 2. \_\_\_\_\_.
- 8.** A \_\_\_\_\_ one shot always initiates a new timing cycle any time a new trigger is applied.
- 9.** A \_\_\_\_\_ one shot simply ignores any new trigger that occurs while a pulse is in progress.

10. A \_\_\_\_\_ multivibrator is used as a gating circuit and as a delay element.
11. An \_\_\_\_\_ multivibrator is used to provide clock signals to synchronous digital circuits.
12. \_\_\_\_\_ oscillators have good frequency stability.
13. Display devices provide a visual display of \_\_\_\_\_, \_\_\_\_\_, and \_\_\_\_\_.
14. The \_\_\_\_\_ LED finds application in burglar alarm systems.
15. The disadvantage of LEDs compared to LCDs is their \_\_\_\_\_ requirement.
16. The advantages of LCDs are \_\_\_\_\_ and \_\_\_\_\_.
17. LCDs consume \_\_\_\_\_ power than LEDs.
18. \_\_\_\_\_ are used in calculators and watches.
19. \_\_\_\_\_ are normally used for seven segment displays.
20. In general \_\_\_\_\_ devices are used to drive LCDs.
21. \_\_\_\_\_ are used in electronic cash registers.

### PROBLEMS

- 19.1 Design a one-shot using the 555 timer to generate a pulse of width 1 ms.
- 19.2 Determine the values of  $R_{\text{ext}}$  and  $C_{\text{ext}}$  that will produce a pulse width of 2  $\mu\text{s}$  when connected to a 74123 as shown in Figure 19.7.
- 19.3 Design an astable multivibrator using the 555 timer to generate a square wave of 1 kHz with 60 per cent duty cycle.
- 19.4 Design an astable multivibrator using the 555 timer to generate a square wave of 5 kHz with 40 per cent duty cycle.
- 19.5 For the astable multivibrator using an op-amp as shown in Figure 19.14,  $R = 20 \text{ k}\Omega$ ,  $C = 0.05 \mu\text{F}$  and  $R_1 = 10 \text{ k}\Omega$ . It is desired to make the frequency of the output square wave adjustable from 15 kHz through 150 kHz by making  $R_2$  adjustable. Through what range of values should  $R_2$  be made adjustable to obtain the required frequency range?



# Appendix

## COMMONLY USED TTL ICs

NUMBER	DESCRIPTION
7400	Quad 2-input NAND gates
7401	Quad 2-input NAND gates (open collector)
7402	Quad 2-input NOR gates
7403	Quad 2-input NOR gates (open collector)
7404	Hex inverters
7405	Hex inverters (open collector)
7406	Hex inverter buffers/drivers
7407	Hex buffer-drivers (open collector high voltage outputs)
7408	Quad 2-input AND gates
7409	Quad 2-input AND gates (open collector)
7410	Triple 3-input NAND gates
7411	Triple 3-input AND gates
7412	Triple 3-input NAND gates (open collector)
7413	Dual 4-input NAND Schmitt trigger
7414	Hex Schmitt-trigger inverters
74H15	Triple 3-input AND gate (open collector)
7416	Hex inverter buffer drivers
7417	Hex buffer drivers
7420	Dual 4-input NAND gates
7421	Dual 4-input AND gates

**1006** Appendix: COMMONLY USED TTL ICs

NUMBER	DESCRIPTION
7422	Dual 4-input NAND gates (open collector)
7423	Dual 4-input NOR gates with strobe
7425	Dual 4-input NOR gates
7426	Quad 2-input TTL-MOS interface NAND gates
7427	Triple 3-input NOR gates
7428	Quad 2-input NOR buffers
7430	8-input NAND gate
74S31	Delay elements
7432	Quad 2-input OR gates
7433	Quad 2-input NOR buffers (open collector)
7437	Quad 2-input NAND buffers
7438, 39	Quad 2-input NAND buffers (open collector)
7440	Dual 4-input NAND buffers
7441, 42	BCD-to-Decimal decoder
7443, 43A	Excess-3-to-Decimal decoder
7444, 44A	Gray-to-Decimal decoder
7445	BCD-to-Decimal decoder/driver
7446	BCD-to-Seven segment decoder/drivers (30 v output)
7447	BCD-to-Seven segment decoder/drivers (15 v output)
7448	BCD-to-Seven segment decoder/driver
7450	Expandable dual 2-input 2-wide AOI gates
7451	Dual 2-input 2-wide AOI gates
7452	Expandable 2-input 4-wide AND/OR gates
7453	Expandable 2-input 4-wide AOI gates
7454	2-input 4-wide AOI gates
7455	Expandable 4-input 2-wide AOI gates
7459	Dual 2/3-inputs 2-wide AOI gates
7460	Dual 4-input expanders
7461	Triple 3-input expanders
7462	2-2-3-3 input 4-wide A/O expanders
7464	2-2-3-4 input 4-wide AOI gates
7465	4-wide AOI gates (open collector)
7470	AND gated positive edge-triggered J-K FF with preset and clear
7471	JK master-slave FF with preset
7472,74H72	AND gated J-K master-slave FF with PRESET and CLEAR
7473	Dual J-K master-slave FF with active-LOW CLEAR
7474	Dual positive edge-triggered D FF
7475	Quad D-latch
7476	Dual positive edge-triggered J-K flip-flops
74LS76A	Dual negative edge-triggered J-K flip-flops
7480	Gated full-adder
7482	2-bit binary full-adder

<b>NUMBER</b>	<b>DESCRIPTION</b>
7483	4-bit binary full-adder with fast carry
7485	4-bit magnitude comparator
7486	Quad X-OR gate
7489	64-bit random access read/write memory
7490	Decade counter
7491	8-bit shift register
7492	Divide-by-12 counter
7493	4-bit binary counter
7494	4-bit shift register
7495	4-bit bidirectional shift register
7496	5-bit parallel-in parallel-out shift register
74100	4-bit bistable latch
74H103	Dual J-K negative edge-triggered FF with CLEAR
74104	J-K master-slave flip-flop
74105	J-K master-slave flip-flop
74H106	Dual J-K negative edge-triggered FF with PRESET and CLEAR
74107	Dual J-K master-slave FF with CLEAR
74LS107A	Dual J-K negative edge-triggered FF
74109	Dual J-K positive edge-triggered FF
74110	AND gated J-K master-slave FF with data lock-out
74111	Dual J-K master-slave FF with data lock-out
74116	Dual 4-bit latches with CLEAR
74121	Non-retriggerable one-shot
74122, 23	Retriggerable one-shot with CLEAR
74125, 26	3-state quad bus-buffer
74128	Quad 2-input NOR buffer
74132	Quad 2-input NAND schmitt trigger
74133	13-input NAND gate
74134	12-input NAND gate
74136	Quad 2-input X-OR gate
74137,38	1:8 demultiplexer
74LS138	3-line to 8-line decoder/demultiplexer
74139	Dual 1:4 demultiplexer
74141	BCD-to-Decimal decoder/driver
74142	BCD counter/latch/decoder/ driver
74145	BCD-to-Decimal decoder/driver
74147	Decimal-to-BCD priority encoder
74148	Octal-to-Binary priority encoder
74150	16-line to 1-line multiplexer
74151	8-channel digital multiplexer
74152	8-channel data selector/MUX
74153	Dual 4-line to 1-line multiplexer

**1008** Appendix: COMMONLY USED TTL ICs

NUMBER	DESCRIPTION
74154	4-line to 16-line decoder/D MUX
74155,156	Dual 2-line to 4-line demultiplexer
74157	Quad 2-line to 1-line data selector
74160	Decade counter with asynchronous CLEAR
74161, 62, 63	Synchronous 4-bit counter
74164	8-bit parallel-out serial shift register
74165,66	Parallel load 8-bit serial shift register
74S168	Synchronous up/down decade counter
74S169	Synchronous up/down binary counter
74170	4 × 4 register files
74173	4-bit D type 3-state register
74174	Hex D flip-flops with CLEAR
74175	Quad D flip-flops with CLEAR
74176	35 MHz presettable decade counter
74177	35 MHz presettable binary counter
74179	4-bit parallel access shift register
74180	8-bit odd/even parity generator/checker
74181	Arithmetic logic unit
74182	Look-ahead carry generator
74184	BCD-to-Binary converter
74185	Binary-to-BCD converter
74189	3-state 64-bit RAM
74190	Synchronous up/down decade counter
74191,93	Synchronous binary up/down counter
74192	Synchronous BCD up/down counter
74194	4-bit bidirectional universal shift register
74195	4-bit parallel access shift register
74196	Presettable decade counter
74197	Presettable binary counter
74198	8-bit shift register
74199	8-bit parallel-access shift register
74221	Dual one-shot with schmitt trigger inputs
74245	Octal transceiver
74246, 47, 48	BCD-to-Seven segment decoder/driver
74251	3-state 8-channel multiplexer
74LS253	Dual 4-to-1 data MUX with 3-state output
74256	Dual 4-bit addressable latch
74257	Quad 2-1 multiplexer
74258	Data selector / multiplexer
74259	8-bit addressable latch
74LS266	Quad 2-input X-NOR gates
74279	Quad latches

<b>NUMBER</b>	<b>DESCRIPTION</b>
74LS280	9-bit odd/even parity generator/checker
74283	4-bit binary full-adder with fast carry
74284, 285	3-state 4-bit by 4-bit parallel binary multipliers
74290	Decade counter
74293	4-bit binary counter
74295	4-bit bidirectional shift register with 3-state outputs
74LS320	Crystal controlled oscillator
74365, 66, 67	3-state hex buffers
74373,74	Latches/ flip-flop
74390	Individual clocks with flip-flops
74393	Dual 4-bit binary counter
CMOS	
CD 4000	Series pin configuration
4510	BCD up/down counter
4511	BCD to seven segment decoder
4514	4-16 line decoder
4515	Binary up/down counter
4518	Dual up counter
4528	Monostable multivibrator
4543	BCD to seven segment decoder
4581	4-bit arithmetic logic Unit



## GLOSSARY

**Active-HIGH (LOW) input** Input which is normally LOW (HIGH) and goes HIGH (LOW) when circuit operation is required.

**Active logic level** Logic voltage level at which a circuit is considered active.

**A/D converter** The circuit which converts an analog signal into its digital form.

**Address** The number that uniquely identifies the location of a word in memory.

**Alphanumeric codes** The codes that represent numbers, alphabetic characters, and symbols.

**Analog** Being continuous or having a continuous range of values as opposed to a discrete set of values.

**Analog system** Interconnection of devices designed to manipulate physical quantities that are represented in analog form.

**AND gate** A digital logic circuit used to implement the AND operation. The output of this circuit is a 1 only when each one of its inputs is a 1.

**ANSI** American National Standards Institute.

**Anti-coincidence gate** An X-OR gate which outputs a HIGH only when its two inputs differ.

**Arbitration** Selection of the input with the highest priority out of a number of inputs which are simultaneously high in an encoder.

**Arithmetic Logic Unit (ALU)** A digital circuit used in computers to perform arithmetic and logic operations.

**ASCII code** American Standard Code for Information Interchange. A seven-bit alphanumeric code used by most computer manufacturers.

## 1012 GLOSSARY

**ASIC** Application specific integrated circuit. An IC designed to meet the specific requirements of a circuit.

**ASM chart** A special flow chart that has been developed specifically to define digital hardware algorithms.

**Asserted level** A term synonymous with active level, the level of the signal required to initiate the process.

**Astable multivibrator** A digital circuit with no stable state; it oscillates between two quasi-stable states.

**Asynchronous** Having no fixed time relationship.

**Asynchronous counter** A type of counter in which the external clock signal is applied only to the first FF and the output of each FF serves as the clock input to the next FF in the chain.

**Asynchronous inputs** Also called the overriding inputs of FFs. They can affect the operation of the FF independent of the clock and synchronous inputs.

**Asynchronous transfer** Transfer of data performed without the aid of clock.

**Backplane** Electrical connection common to all segments of the LCD.

**Base** The number of symbols in a number system. Also, one of the three regions in a BJT.

**BCD** Binary coded decimal, a digital code.

**BCD adder** An adder containing two 4-bit parallel adders and a correction detector circuit.

**BCD counter** A mod-10 counter that counts from 00002 to 10012.

**Bidirectional shift register** Shift-left, shift-right, shift register, in which data can be shifted in either direction.

**Bilateral switch** A CMOS circuit which acts like a single-pole, single-throw switch controlled by an input logic level.

**Binary** Having two values or states.

**Binary counter** A counter in which the states of FFs represent the binary number equivalent to the number of pulses that have occurred at the input of the counter.

**Binary multiplier** A digital circuit capable of performing the arithmetic operation of multiplication on two binary numbers.

**Binary number system** A number system with only two values, 0 and 1.

**Binary point** A mark which separates the integer and fractional parts of a binary number.

**Bipolar DAC** A DAC whose output can assume a positive or negative value depending on the signed binary input .

**Bipolar ICs** The ICs which use BJTs as the main circuit elements.

**Bistable multivibrator** A multivibrator which can remain indefinitely in any one of its two states. It is commonly known as flip-flop.

**Bit** Binary Digit, a 1 or a 0.

**Block parity** A method of providing parity row wise and column wise for a block of information words.

**Boolean algebra** It is the study of mathematical logic.

**Bubbled AND gate** The AND gate with inverted inputs. It performs the NOR operation.

**Bubbled OR gate** The OR gate with inverted inputs. It performs the NAND operation.

**Buffer driver** A circuit with greater output current and/or voltage capability than an ordinary logic circuit.

**Buffer register** The register that holds digital data temporarily.

**Byte** A group of 8 bits.

**Capacity** The amount of storage space in a memory expressed as number of bits or number of words.

**Carry** A digit that is carried to the next column when two digits are added.

**CCD** Charge-coupled device, a type of semiconductor technology.

**Cell** A single storage element in a memory.

**Check sum** A special data word that is derived from the addition of all other data words. It is used for error checking purposes.

**Circulating shift register** A shift register in which the output of the last FF is connected as the input to the first FF.

**CLEAR** An asynchronous FF input that makes  $Q = 0$  instantaneously.

**Clock** The basic timing signal in a digital system.

**Clock skew** Arrival of a clock signal at different times at the clock inputs of different FFs as a result of propagation delays.

**Clock transition times** Minimum rise and fall times for the clock signal transitions.

**Closed subgraph** A subgraph in which for every vertex all outgoing arcs and their terminating vertices also belong to the subgraph.

**CMOS** Complementary Metal Oxide Semiconductor. The IC technology which uses both NMOS and PMOS FETs as the principal circuit elements.

**Code** A combination of binary digits that represents information such as numbers, alphabets, and other symbols.

**Code converter** An electronic digital circuit that converts one type of coded information into another coded form.

**Coincidence gate** The X-NOR gate which outputs a HIGH only when its two inputs are the same.

**Combinational logic circuit** A combination of logic devices having no storage capability and used to generate a specified function.

**Compatibility graph** A graph whose vertices correspond to all compatible pairs and whose directed arcs lead from one vertex to another vertex if and only if the compatible pair of the first vertex implies the compatible pair of the second vertex.

**Complement** An invert function in Boolean algebra

**Computer word** The group of bits used as the primary unit of information in a computer.

**Conditional output box** A rectangular box with rounded corners or an oval shaped box which lists the outputs that occur when the path to a conditional output box is satisfied.

**Control inputs** Input signals synchronized with the active clock transition that determine the output state of a FF.

**Control Subsystem** A sequential circuit whose internal states decide the control commands for the system.

**Cyclic code** A code in which the successive code words differ in only one bit.

## 1014 GLOSSARY

**D/A converter** The circuit which converts a digital input into an analog output.

**Data** Information in numeric, alphanumeric, or other form.

**Data lock-out FFs** A master-slave FF that has a dynamic clock input.

**Datapath** Also known as data processing path manipulates data in registers according to system's requirements.

**DC CLEAR** Asynchronous FF input used to clear the FF.

**DC SET** Asynchronous FF input used to set the FF.

**De Morgan's theorems** (1) The complement of an OR operation on variables is equal to an AND operation on the complemented variables. (2) The complement of an AND operation on variables is equal to the OR operation on complemented variables.

**Decade counter** A digital counter having 10 different states.

**Decimal number system** The number system with 10 different symbols.

**Decision box** A diamond shaped box with one input path and two or more output paths to describe the effect of an input on the control subsystem.

**Decoder** A digital circuit that converts coded information into familiar form.

**Demultiplexer (DMUX)** The logic circuit that channels its data input to one of several data outputs.

**Dependency notation** A notational system for logic symbols that specifies the input and output relationships.

**Digit** A symbol representing a given quantity in a number system.

**Digital ICs** Self-contained digital circuits, made by using one of several IC technologies.

**Digital system** A combination of devices designed to manipulate physical quantities that are represented in digital form.

**DIP** Dual-in-line-package; the most common type of IC package.

**Dominating column** In a prime implicants chart, a column which has a 'X' in every row in which another column has a 'X'.

**Dominating row** In a prime implicants chart, a row which has a 'X' in every column in which another row has a 'X'.

**Don't care** A condition in a logic circuit in which the output is independent of the state of a given input.

**DRAM** Dynamic RAM: a type of semiconductor memory that stores data as capacitor charges that need to be refreshed periodically.

**Dynamic shift register** A register in which data continually circulates through the register under the control of a clock.

**EAROM** Electrically alterable read only memory.

**EBCDIC code** Extended Binary Coded Decimal Interchange Code: an alphanumeric code.

**ECL** Emitter Coupled Logic. Also referred to as current mode logic (CML).

**Edge detector** A circuit that produces a narrow positive spike, coincident with the active transition of a clock pulse.

**Edge-triggered FF** A type of FF which is activated by the clock signal transition.

**EEPROM** Electrically erasable programmable read only memory.

**Encoder** A digital circuit that converts information into coded form.

**EPROM** Erasable programmable read only memory.

**Equivalent states** States from which, every possible set of inputs applied to the machine generate exactly the same output and take the machine exactly to the same next state.

**Excess-3 code** A digital code in which each of the decimal digits is represented by a 4-bit code derived by adding 3 to each of the digits.

**Excess-3 Gray code** A digital code in which each decimal digit is encoded with a gray code pattern of the decimal digit that is greater by 3.

**Excitation table** A table showing the required input conditions for each possible state transition of a device/machine.

**Exclusive-NOR gate** A logic circuit which produces a 1 output only when its inputs coincide.

**Exclusive-OR gate** A logic circuit which produces a 1 output only when its inputs are different.

**Fan-out** The maximum number of equivalent gate inputs that the output of a gate can drive without impairing its own function.

**Finite state machine** An abstract model describing the synchronous sequential machine.

**Flat package** A type of IC package.

**Flip-flop** A memory device capable of storing a logic level.

**Floppy disk** Flexible magnetic disk used for mass storage.

**Frequency counter** A circuit that can measure and display the frequency of a signal.

**Full-adder** A digital circuit that adds two binary digits and an input carry, and produces a sum digit and an output carry.

**Full-subtractor** A digital circuit that subtracts one bit from another bit considering previous borrow.

**Function generator** A circuit that produces a variety of waveforms.

**Fuse map** A map indicating which fuses are blown and which fuses are intact.

**Gate** A circuit that performs a specified logic operation.

**Gated latch** A latch that responds to the input only when its ENABLE is HIGH.

**Glitch** A voltage or current spike of short duration which is usually unwanted.

**Gray code** A unit distance code in which the successive code words differ in only one bit.

**Half-adder** A digital circuit which can add only two bits, and produces a sum bit and a carry bit.

**Half-subtractor** A circuit which can subtract one bit from another.

**Hard disk** A rigid metal magnetic disk used for mass storage.

**Hexadecimal number system** A number system consisting of 16 symbols, 0–9 and A–F.

**Hold time** The time interval for which the control signal has to be maintained at the input terminal of the FF after the clock termination in order to obtain reliable operation.

**Hybrid circuits** Circuits containing both integrated and discrete components.

**Hybrid counter** A synchronous counter whose output drives the clock input of another counter.

**IC** Integrated circuit: a type of circuit in which all the components are integrated on a single silicon chip of very small size.

**Incompletely specified machines** The sequential circuits in which the state transitions or output variables are not completely specified.

## 1016 GLOSSARY

**Inhibit circuits** Logic circuits that control the passage of an input signal through to the output.

**Interfacing** Joining of dissimilar devices in such a way that they are able to function in a compatible and coordinated manner; connection of the output of a system to the input of a different system with different electrical characteristics.

**Invert** Causing a logic level to go to the opposite state.

**Inverter** A logic circuit that implements the NOT operation.

**J-K FF** A type of FF that can operate in the ‘no change’, ‘set’, ‘reset’ and ‘toggle’ modes.

**Johnson counter** A type of shift register counter in which the inverted output of the last FF is connected as data input to the first FF.

**Karnaugh map** An arrangement of cells representing the combinations of variables in a Boolean expression and used for a systematic simplification of the expression.

**Latch** A non-clocked FF.

**Linearity error** The maximum deviation in step size from the ideal step size in a DAC.

**Lock-out** The state of a counter when successive clock pulses take the counter only from one invalid state to another invalid state.

**Logic circuit** A circuit that behaves according to a set of logic rules.

**Logic level** State of a voltage variable. States HIGH and LOW correspond to the two usable voltage levels of a digital device.

**Look-ahead carry** A method of binary addition whereby carries from the preceding stages are anticipated, thus avoiding the carry propagation delays.

**Looping** Combining of adjacent squares in a K-map containing 1s (0s) for the purpose of simplification of a SOP (POS) expression.

**LSB** The least significant bit, i.e. the rightmost bit in the binary number.

**LSD** The least significant digit, i.e. the digit that carries the least weight in a particular number.

**LSI** Large scale integration. A level of integration in which 100 to 9999 gate circuits are integrated on a single chip.

**Magnetic bubble** A tiny magnetic region in magnetic material created by an external magnetic field.

**Magnetic bubble memory (MBM)** Solid state, non-volatile, sequential access, mass storage memory device consisting of tiny magnetic domains (bubbles).

**Magnetic core memory** Non-volatile random access memory made up of small ferrite cores.

**Magnetic disk memory** Mass storage memory that stores data as magnetized spots on a rotating flat disk surface.

**Magnetic tape memory** Mass storage memory that stores data as magnetized spots on an iron-coated plastic tape.

**Magnitude comparator** A digital circuit used to compare the magnitudes of two binary numbers and indicate whether they are equal or not, and if not which one has larger magnitude.

**Mass storage** Storage of large amounts of data, not part of the computer’s internal memory.

**Master slave FF** A type of FF in which the input data affects the first of its two FFs at the leading edge of the clock pulse and then the contents of the first FF appear at the output of the second FF at the trailing edge of clock pulse.

**Maxterm** The product term in the standard POS form.

**Mealy model** A sequential circuit in which the output is a function of the present state as well as the present input.

**Memory array** An arrangement of memory cells.

**Memory cell** An individual storage element in a memory.

**Memory word** Groups of bits in memory that represent instructions or data of some type.

**Merger graph** A graph whose vertices correspond to all the states of the machine and whose vertices are joined by lines with compatible pairs written in the line break. It is a state reducing tool used to reduce states in the incompletely specified machine.

**Merger table** A table whose each cell corresponds to one compatible pair.

**Microprocessor** A large-scale integrated circuit that can be programmed to perform arithmetic and logic functions and to manipulate data.

**Minimal cover table** A table consisting of states of minimal state machine.

**Minterm** The sum term in the standard SOP form.

**Modified modulus counter** A counter that does not sequence through all of its natural states.

**Modulus** The maximum number of states in a counter sequence.

**Monostable multivibrator** A multivibrator having only one stable state. The other one is a quasi-stable state. It is also called one-shot.

**Monotonicity** A property whereby the output of a DAC either increases or stays at the same value, but never decreases as the input is increased.

**Moore model** A sequential circuit in which the output is a function of the present state only.

**MROM** A type of ROM whose storage locations are written into by the manufacturer.

**MSB** The most significant bit. The leftmost bit in a binary number.

**MSD** The most significant digit. The digit that carries the most weight in a particular number, i.e. the extreme left digit in the number.

**MSI** Medium scale integration. A level of integration in which 12–99 gate circuits are fabricated on a single chip.

**Multiplex** To put information from several sources on to a single line or transmission path.

**Multiplexer (MUX)** A digital circuit, depending on the status of its control inputs, that channels one of several data inputs to its output.

**NAND gate** The logic gate that outputs a 0 only when all its inputs are 1s. It gives the complement of the AND output.

**NAND-gate latch** Basic flip-flop constructed using two cross-coupled NAND gates.

**Negative logic** The system of logic in which a LOW represents a 1 and a HIGH represents a 0.

**NMOS** N-channel metal oxide semiconductor.

**Noise** Unwanted (spurious) signals.

**Noise immunity** The ability of a circuit to tolerate noise voltages on its inputs.

**Noise margin** Quantitative measure of the noise immunity. It is the maximum noise voltage that can be added at the input of a gate without affecting its operation.

**Non-retriggerable one-shot** A type of one-shot that does not respond to a trigger input signal while in its quasi-stable state.

## 1018 GLOSSARY

**Non-volatile memory** Memory that is able to keep its information stored in the event of failure of electrical power.

**NOR gate** A logic circuit that outputs a 1 only when each one of its inputs is a 0. It is equivalent to an OR gate followed by an inverter.

**NOR-gate latch** A flip-flop constructed using two cross-coupled NOR gates.

**NOT circuit** A logic circuit that inverts its only input.

**Octal number system** A number system with 8 digits (0, 1, 2, 3, 4, 5, 6, 7).

**Octet** A group of 8 1s or 0s that are adjacent to each other in a Karnaugh map.

**Offset error** The voltage present at the output of a DAC when the input is all 0s.

**One's complement form** A form of representation obtained by complementing each bit of a binary number.

**One-shot** A multivibrator with only one stable state—the other name of monostable multivibrator.

**Open-collector gates** The TTL gates which use only one transistor with a floating collector in the output structure.

**OR gate** A logic circuit that outputs a HIGH whenever one of its inputs is a HIGH.

**Override inputs** Asynchronous inputs in a FF which override the effects of all other input signals.

**Oscillator** An electronic circuit that switches back and forth between two states.

**PAL** A type of PLD with programmable AND array and a fixed OR array.

**Parallel adder** A digital circuit with full-adders that adds all the bits from two numbers simultaneously.

**Parallel counter** A counter in which all the FFs are triggered simultaneously—the other name of synchronous counter.

**Parallel data transfer** The operation by which the entire contents of a register are transferred to another register.

**Parallel-in, parallel-out, shift register** A type of shift register that can be loaded with parallel data and has also parallel outputs available.

**Parallel-in, serial-out, shift register** A type of shift register that can be loaded with parallel data but has only one serial output terminal.

**Parallel transmission** Simultaneous transfer of all bits of a binary number from one place to another.

**Parity bit** An additional bit that is attached to each code group so that the total number of 1s being transmitted is either odd or even.

**Parity checker** A circuit that is used to check parity among the group of bits received.

**Parity generator** A digital circuit that takes a set of data bits and produces the correct parity bit for the data.

**Partition technique** A procedure for minimization of completely specified sequential machines.

**Percentage resolution** The ratio of the step size to the full-scale value of a DAC. It can also be defined as the reciprocal of the maximum number of steps of a DAC.

**PLA** A type of PLD with programmable AND array and a programmable OR array.

**PLD** An IC that is user configurable and capable of implementing logic functions.

**PMOS** P-channel metal oxide semiconductor.

**POS form** Product of sums form: A form of Boolean expression in which a number of sum terms are multiplied.

**Positional-value system** The system in which the value of a digit is dependent on its relative position.

**Positive logic system** The system of logic in which a HIGH represents a 1 and a LOW represents a 0.

**PRESET** Asynchronous input used to instantaneously set Q = 1.

**Presetable counter** A counter that can be PRESET to any initial count either synchronously or asynchronously.

**Prime implicant** A term which cannot be combined further in the tabular method.

**Priority encoder** An encoder that produces a coded output corresponding to the highest-valued input when two or more inputs are applied simultaneously.

**Programming** Fuse blowing process of PLDs.

**PROM** A ROM that can be programmed by the user. It cannot be erased and reprogrammed.

**Propagation delay** The time interval between the occurrence of an input transition and the corresponding output transition.

**Pulse** A sudden change from one level to another followed by a sudden change back to the original level.

**Quantization error** The error caused by non-zero resolution of an ADC; it is an inherent error of the device.

**Quasi-stable state** A temporary stable state. A monostable circuit is triggered to this state before returning to the normal stable state. Both the states of an astable multivibrator are quasi-stable.

**Quine-McClusky method** A tabular method used to minimize Boolean expressions.

**Race** A condition in a logic network in which the differences in propagation times through two or more signal paths in the network can produce an erroneous output.

**Radix** The base of a number system. The number of symbols in a given number system.

**RAM** Random access memory—a memory in which the access time is the same for all locations.

**Read** The process of retrieving information from a memory.

**Read/write memory** A memory that can be both read from and written into.

**Redundant states** States whose functions can be accomplished by other states.

**Reflected code** A code with the property that an N-bit code can be obtained by reflecting an N – 1 bit code about an axis at the end of the code and putting 0s above the axis and 1s below the axis.

**Refresh** The process of renewing the contents of a dynamic memory.

**Register** A group of flip-flops capable of storing data.

**RESET** The state of a FF, register or counter when 0s are stored. This term is synonymous with CLEAR.

**Resolution** In a DAC, the smallest change that can occur in the output for a change in the digital input. Also, called the step size. In an ADC, the smallest amount by which the analog input must change to produce a change in the digital output.

**Retriggerable one-shot** A type of one-shot that will respond to a trigger input signal while in its quasi-stable state.

**Ring counter** Serial-in, serial-out, shift register in which the output of the last FF is connected to the input of the first FF.

**Ripple counter** A counter in which the external clock signal is applied to the first FF and then the clock input to every other FF is the output of the preceding FF.

**ROM** Read only memory.

**Schmitt trigger** A digital circuit that converts a slow-changing signal into a fast-changing signal.

**Schottky TTL** A TTL subfamily that uses the basic standard TTL circuit except that it uses a Schottky barrier diode between the base and collector of each transistor.

**Self-complementing code** A code in which the code word of the 9's complement of  $N$ , i.e. of  $9 - N$  can be obtained from the code word of  $N$  by interchanging all 0s and 1s.

**Sequential code** A code in which each succeeding code word is one binary number greater than its preceding code word.

**Sequence detector** A digital circuit used to detect a sequence in the input.

**Sequential logic** A system of logic in which the logic output states and the sequence of operations depend on both the present and the past input conditions.

**Serial adder** A type of adder which adds two numbers by taking the bits from them serially with a carry.

**Serial data transmission** Transfer of data from one place to another one bit at a time on a single line.

**Serial-in, parallel-out, shift register** A type of shift register that can be loaded with data serially, but has parallel outputs available.

**Serial-in, serial-out, shift register** A type of shift register that can be loaded with data serially, and also has a serial output terminal available.

**SET** The state of a flip-flop when it is in the binary 1 state.

**Set-up time** The time interval for which the control signals must be held constant at the input terminals of a FF, prior to the arrival of the triggering edge of the clock pulse.

**Settling time** The time taken by the output of a DAC to rise and settle within one-half step size of its full scale value as the input is changed from all 0s to all 1s.

**Shift register** A digital circuit capable of storing and shifting binary data.

**Sign bit** A binary digit that is inserted at the leftmost position of a binary number to indicate whether that number represents a positive or negative quantity.

**SOP form** Sum of products form: A form of Boolean expression in which a number of product terms are summed that is, the ORing of ANDed terms.

**Speed power product** Numerical value (in joules) often used to compare different logic families. It is obtained by multiplying the propagation delay (ns) by the power dissipation (mW) of a logic gate.

**SSI** Small scale integration (less than 12 gates per chip).

**Stage** One storage element in a register or counter.

**Standard POS form** A form of Boolean expression in which each sum term contains all the variables of the function and all these sum terms are multiplied together.

- Standard SOP form** A form of Boolean expression in which each product term contains all the variables of the function and all these product terms are summed together.
- State assignment** The process of assigning the states of a physical device to the states of a sequential machine.
- State box** A rectangle shaped box with one input path and one output path used in ASM charts to indicate a state of the machine.
- State diagram** A picture showing the relationship between the present state, the input, the next state and the output of a sequential machine.
- State machine** Any sequential circuit exhibiting a specified sequence of states.
- State table** A table showing the relationship between the present state, the input, the next state and the output of a sequential machine.
- State variables** The output values of the physical device in a sequential machine.
- SRAM (Static RAM)** A RAM that stores information in FF cells which do not have to be refreshed unlike those of the DRAM.
- Storage** The memory capability of a digital device. Also, the process of storing digital data for later use.
- Straight binary coding** Representation of a decimal number by its equivalent binary number.
- Strobing** A technique often used to eliminate decoding spikes.
- Strongly connected machine** A sequential machine in which for every pair of states  $S_{i-}, S_j$  of the machine there exists an input sequence which takes the machine from  $S_i$  to  $S_j$ .
- Subgraph** Any part of a compatibility graph.
- Substrate** A piece of semiconductor material over which the components are fabricated in an IC.
- Synchronous** Having a fixed time relationship.
- Synchronous counter** A counter in which the circuit outputs can change states only on the transitions of a clock.
- Synchronous inputs** Input signals synchronized with the active clock transition that determine the output state of a FF.
- Synchronous systems** Systems in which the circuit outputs can change states only on the transition of a clock.
- Synchronous transfer** Data transfer performed by using synchronous and clock inputs of a FF.
- Terminal count** The final state of a counter sequence.
- Terminal state** A state with no outgoing arcs which start from it and terminate in other states.
- Threshold function** A logic function that can be realized by a single threshold element.
- Threshold gate** A logic element with one or more binary inputs with associated weights outputting a 1 when the weighted sum of inputs is greater than or equal to a threshold value and outputting a 0 otherwise.
- Threshold logic** A form of logic realization using threshold elements.
- Timing diagram** Depiction of logic levels as related to time.
- Toggle mode** The mode in which a FF changes state for each clock pulse.
- Totem-pole** A term used to describe the way in which two bipolar transistors are connected one above the other at the outputs of most TTL gates.

## 1022 GLOSSARY

**Transition** A change from one level to another.

**Transition and output table** A table showing the state transitions and the output of the sequential machine in terms of the assigned states.

**Transmission gate** A digitally controlled bi-lateral CMOS switch.

**Trigger** A pulse used to initiate a change in the state of a logic circuit.

**Tri-state** A type of output structure which allows three types of output states—HIGH, LOW and high-impedance.

**Truth table** A tabular representation of the outputs of a logic circuit for all possible combinations of inputs.

**TTL (Transistor Transistor Logic)** An IC technology that uses the bipolar transistor as the basic circuit element.

**ULSI** Ultra large scale integration (more than 100,000 gate circuits per chip).

**Unate function** A switching function that is positive or negative in each of its variables.

**Unipolar ICs** The ICs in which the MOSFETs are the main circuit elements.

**Unit distance code** A code having the property such that the bit patterns for two consecutive numbers differ in one bit position only.

**Unit load** The current drawn by the input of a logic gate when connected to the output of an identical gate.

**Universal gates** The gates using which the basic logic functions (AND, OR and NOT) can be realized.

**Universal shift register** Shift-right, shift-left, shift register which can input and output data either serially or parallelly.

**Up-counter** A counter that counts upwards from zero to a maximum count.

**UVEPROM** Ultra violet erasable programmable read only memory.

**Variable modulus counter** A counter in which the maximum number of states can be changed.

**VLSI** Very large scale integration (10,000 to 99,999 gate circuits per chip).

**Volatile memory** Memory requiring electrical power to keep information stored.

**Voltage level translator** A circuit that takes one set of input voltage levels and translates it to a different set of output voltage levels.

**Weight** The positional value of a digit in a number.

**Weighing machine** A logic circuit used to count the number of 1s in the given binary number.

**Weighted code** A digital code that utilizes weighted numbers as the individual code words.

**Wired AND** A term used to describe the logic function created when open-collector outputs are tied together.

**Word** A group of bits representing a complete piece of digital information.

**Wrapping around** Folding at the centre of a Karnaugh map such that the edges coincide.

**Writing** The process of storing information in a memory.

# ANSWERS

## CHAPTER 1

### Answers to Fill in the Blanks

- |   |   |                               |                      |
|---|---|-------------------------------|----------------------|
| 1. digital                                      | 2. digital                                    | 3. analog                     | 4. analog, digital   |
| 5. analog                                       | 6. digital                                    | 7. switching                  |                      |
| 8. combinational, sequential                    |   | 9. asynchronous, synchronous  |                      |
| 10. combinational                               | 11. sequential                                | 12. asynchronous              | 13. synchronous      |
| 14. switching, logic                            | 15. circuit's logic                           | 16. hybrid                    |                      |
| 17. system design, logic design, circuit design |   | 18. System                    | 19. Logic            |
| 20. Circuit                                     | 21. positive logic                            | 22. negative logic            | 23. amplitude        |
| 24. 5, 0  | 25. duty cycle                                | 26. pulse width               | 27. rise time        |
| 28. fall time                                   | 29. Encoding                                  | 30. Decoding                  | 31. multiplexing     |
| 32. sharing                                     | 33. $N, 1$                                    | 34. Demultiplexing            | 35. 1, $N$           |
| 36. registers                                   | 37. serial, parallel                          | 38. analog, digital           | 39. digital          |
| 40. unipolar ICs, bipolar ICs                   |   | 41. SSI, MSI, LSI, VLSI, ULSI |                      |
| 42. < 12  | 43. 12 to 99                                  | 44. 100 to 9,999              | 45. 10,000 to 99,999 |
| 46. > 1,00,000                                  | 47. microcomputers, minicomputers, mainframes |                               |                      |
| 48. program                                     | 49. TTL, ECL, IIL, MOS, CMOS.                 |                               |                      |

### Answers to Objective Type Questions

- |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|
| 1. b  | 2. b  | 3. b  | 4. c  | 5. a  | 6. a  | 7. d  |
| 8. b  | 9. c  | 10. c | 11. b | 12. c | 13. b | 14. b |
| 15. b | 16. c | 17. c | 18. a | 19. b | 20. b | 21. a |
| 22. a |       |       |       |       |       |       |

## CHAPTER 2

### Answers to Fill in the Blanks

- |   |  |                             |
|---|--|-----------------------------|
| 1. base, radix                            | 2. position-weighted system                    | 3. base or radix point      |
| 4. LSD, MSD                               | 5. bit   | 6. nibble                   |
| 8. hex                                    | 9. 1, 128                                      | 10. 10                      |
| 12. binary                                | 13. $(b - 1)$ 's complement, $b$ 's complement |                             |
| 14. ignored, add to the LSD               | 15. original number                            | 7. four                     |
| 17. double precision, triple precision    | 18. two  | 11. sign                    |
| 20. ease of conversion to and from binary | 21. 1/3rd, 1/4th                               | 16. unique, two             |
| 23. 4-bit binary equivalent               |  | 19. floating point          |
|   |  | 22. 3-bit binary equivalent |

### Answers to Objective Type Questions

- |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|
| 1. d  | 2. d  | 3. b  | 4. b  | 5. b  | 6. c  | 7. b  |
| 8. b  | 9. b  | 10. b | 11. c | 12. d | 13. a | 14. b |
| 15. d | 16. c | 17. a | 18. b | 19. c | 20. c | 21. b |
| 22. b | 23. c | 24. d | 25. b | 26. c | 27. b | 28. c |
| 29. d | 30. b | 31. a | 32. b |       |       |       |

### Answers to Problems

- |                                   |                        |                        |                        |
|-----------------------------------|------------------------|------------------------|------------------------|
| <b>2.1</b> (a) 188                | (b) -522               | (c) 294.9              | (d) -389.3             |
| <b>2.2</b> (a) 11                 | (b) 109                | (c) 13.75              | (d) 110.375            |
| <b>2.3</b> (a) 100101             |                        | (b) 11100              |                        |
|                                   | (c) 11000101.100011    |                        | (d) 11001101.000011    |
| <b>2.4</b> (a) 101000             | (b) 110000             | (c) 1001110.111        | (d) 110001.11          |
| <b>2.5</b> (a) 0110               | (b) 1011               | (c) 101.01             | (d) 1.10               |
| <b>2.6</b> (a) 1000001            | (b) 110010             | (c) 1001011.101        | (d) 110111             |
| <b>2.7</b> (a) 11.0101            | (b) 110                | (c) 1010.11001         | (d) 1011.000111        |
| <b>2.8</b> (a) 0010 0001          | (b) 0010 1101          | (c) 0100 1000          | (d) 0011 1100          |
| <b>2.9</b> (a) R = 0001, Q = 0101 |                        | (b) R = 0000, Q = 1010 |                        |
|                                   | (c) R = 1001, Q = 0111 |                        | (d) R = 0100, Q = 0011 |
| <b>2.10</b> (a) 1111 1101 1011    |                        | (b) 1111 0101 0011     |                        |
|                                   | (c) 1011 1110.1000     |                        | (d) 0011 1010.1000     |
| <b>2.11</b> (a) 1111 1001 1110    |                        | (b) 1111 0001 1111     |                        |
|                                   | (c) 0011 0010.0011     |                        | (d) 1110 0010.1001     |
| <b>2.12</b> (a) 27                | (b) -48                | (c) 78.5               | (d) -52.75             |
| <b>2.13</b> (a) 35                | (b) -38                | (c) 46.25              | (d) -39.25             |
| <b>2.14</b> (a) 0000 1110         | 0000 1110              | 0000 1110              |                        |
|                                   | (b) 0001 1011          | 0001 1011              |                        |
|                                   | (c) 0010 1101          | 0010 1101              |                        |
|                                   | (d) 1001 0001          | 1110 1110              |                        |
|                                   | (e) 1010 0101          | 1101 1010              |                        |
|                                   | (f) 1100 1100          | 1011 0011              |                        |
|                                   |                        | 1011 0100              |                        |

- 2.15** (a) AE (b) 41D (c) 3F2.98 (d) C2C.598  
**2.16** (a) 1253 (b) 41375 (c) 2367.52 (d) 136160.034  
**2.17** (a) 307 (b) 1070 (c) 1071.81 (d) 3420.06  
**2.18** (a) 437 (b) 7564 (c) 644.463 (d) 20434.360  
**2.19** (a) 460 (b) 3322 (c) 44.55 (d) 720.65  
**2.20** (a) 25 (b) 265 (c) 125.7 (d) 126.67  
**2.21** (a) 230 (b) 1623 (c) 73.31 (d) 4427.70  
**2.22** (a) 66.314 (b) 351.125 (c) 17.311 (d) 366.05  
**2.23** (a) 51 (b) -365 (c) 104.7 (d) 442.04  
**2.24** (a) 1100 0010 0000 (b) 1111 0010 1001 0111  
         (c) 1010 1111 1001.1011 0000 1101  
         (d) 1110 0111 1001 1010.0110 1010 0100  
**2.25** (a) 16 (b) 2DB (c) 1B7.78 (d) 1B6D.B4  
**2.26** (a) 2742 (b) 11959 (c) 41103.914 (d) 36423.668  
**2.27** (a) 1C4 (b) 12BC (c) 4E0.8F5 (d) 22FD.C  
**2.28** (a) 161F (b) 1585B (c) 12ADA.54 (d) 1AD84.6E  
**2.29** (a) 19A (b) 389 (c) C341.D2 (d) 1DE3.C5  
**2.30** (a) 1BEE (b) 27A3D (c) 1E12E.94 (d) 664.DF  
**2.31** (a) E4.C (b) 726 (c) 41B.0E (d) A17.5  
**2.32** (a) 60 (b) 371 (c) 9976.FB (d) 3AF6.2  
**2.33** (a) 1111101.11 (b) 11122.202 (c) 1331.3 (d) A5.9

CHAPTER 3

## **Answers to Fill in the Blanks**

- |   |                        |                           |                             |
|---|------------------------|---------------------------|-----------------------------|
| 1. numeric, alphanumeric  | 2. alphanumeric codes  |                           |                             |
| 3. numeric codes  | 4. BCD                 | 5. code word              |                             |
| 6. (a) weighted codes, (b) non-weighted codes                   | 7. weighted codes      |                           |                             |
| 8. (a) positively-weighted codes, (b) negatively-weighted codes |                        |                           |                             |
| 9. negatively-weighted codes                                    | 10. straight binary    |                           |                             |
| 11. non-weighted codes  | 12. coding             | 13. sequential            |                             |
| 14. sequential  | 15. self-complementing | 16. nine                  | 17. cyclic codes            |
| 18. unit distance   | 19. 8421 BCD           | 20. six                   | 21. sequential              |
| 22. Gray code   | 23. reflective code    | 24. parity bit            | 25. odd parity, even parity |
| 26. odd, even   | 27. distance           | 28. two or more           | 29. two                     |
| 30. three or more   | 31. Hamming            | 32. 3                     | 33. 3                       |
| 34. 2   | 35. 3                  | 36. (a) ASCII, (b) EBCDIC |                             |
| 37. seven, eight  | 38. $2^{k-1}$          | 39. one                   | 40. correct, detect         |

## **Answers to Objective Type Questions**

- 1. d            2. d            3. c            4. a            5. b            6. b            7. c**  
**8. b            9. a            10. c          11. c          12. b          13. a          14. a**

**1026** ANSWERS

15. a

16. c

17. b

18. b

19. c

20.

21. b

22. b

23. c

24. b

25. c

26. a

27.

28. d

## Answers to Problems

- 3.15** (a) 1100110      (b) 0011001      (c) 0111100      (d) 1010101
- 3.16** 0011001, 0011001, 1000011, 1010101.
- 3.17** (i) (a) 1000010, 1001001, 1010010, 1010100, 1001000  
           (b) 1000001, 1001011, 0110100, 0110111  
       (ii) (a) C2, C9, D9, E3, C8  
           (b) C1, D2, F4, F7
- 3.18** (b) 011, 0000, 0110, 0101
- 3.19** (a) 0000, 0001, 0010, 0011, 0101, 1010, 1100, 1101, 1110, 1111  
       (b) 0000, 0011, 0101, 0010, 0100, 1011, 1101, 1010, 1100, 1111
- 3.20** (a) A, C, D      (b) C, D      (c) D      (d) C, D  
       (e) D      (f) C, D
- 3.21** (a) 010, 0101, 0101, 1010      (b) 000, 1101, 1110, 1011
- 3.22** (a) 1111, 0110, 1001  
       (b) (i) 0101, 0110  
           (ii) 1011, 0101  
           (iii) 1100, 1101
- 3.23** (a) 0001, 1011, 1011      (b) 1011, 0101, 1110

## CHAPTER 4

### Answers to Fill in the Blanks

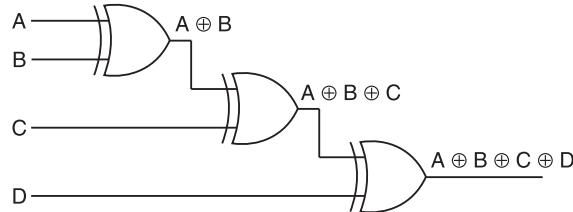
- |                            |                  |               |            |
|----------------------------|------------------|---------------|------------|
| 1. logic design            | 2. truth table   | 3. AND gate   | 4. OR gate |
| 5. only one                | 6. X-NOR         | 7. X-OR       | 8. NOR     |
| 9. NAND                    | 10. AND, OR, NOT | 11. NAND, NOR |            |
| 12. NAND, NOR, X-OR, X-NOR |                  | 13. OR        | 14. AND    |

### Answers to Objective Type Questions

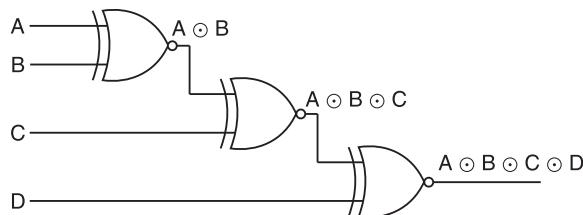
- |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|
| 1. d  | 2. d  | 3. c  | 4. a  | 5. a  | 6. a  | 7. b  |
| 8. a  | 9. d  | 10. c | 11. a | 12. c | 13. a | 14. b |
| 15. b | 16. b | 17. d | 18. c | 19. d | 20. b | 21. a |
| 22. b | 23. b | 24. b | 25. c | 26. a | 27. b | 28. a |
| 29. b | 30. c | 31. b | 32. a | 33. a | 34. c | 35. b |
| 36. d | 37. a | 38. d | 39. b | 40. b | 41. a | 42. c |
| 43. a | 44. b | 45. a | 46. d | 47. a | 48. b | 49. c |
| 50. d | 51. d | 52. b | 53. c | 54. a | 55. d | 56. c |
| 57. a | 58. d | 59. b | 60. a | 61. b | 62. c | 63. a |
| 64. d | 65. b |       |       |       |       |       |

## Answers to Problems

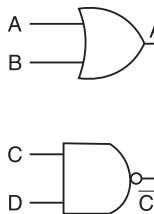
## 4.1 Arrangement using X-OR



## Arrangement using X-NOR gates

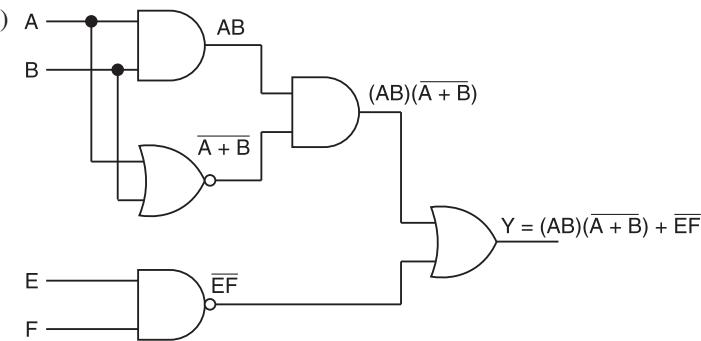


## 4.2 (a)

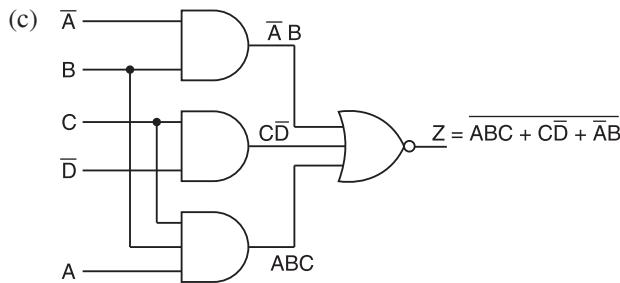


A	B	C	D	$A + B$	$\overline{CD}$	$(A + B) + \overline{CD}$
0	0	0	0	0	1	1
0	0	0	1	0	1	1
0	0	1	0	0	1	1
0	0	1	1	1	0	0
0	1	0	0	1	1	1
0	1	0	1	1	1	1
0	1	1	0	1	1	1
1	0	0	0	1	1	1
1	0	0	1	1	1	1
1	0	1	0	1	1	1
1	0	1	1	1	0	1
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	0	1

(b)



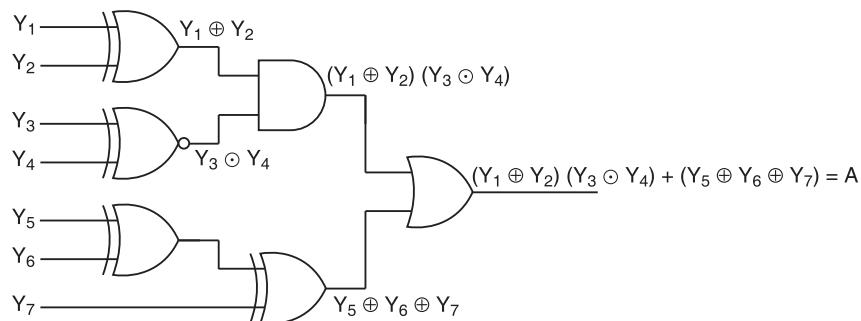
A	B	E	F	AB	$(\bar{A} + B)$	$AB(\bar{A} + \bar{B})$	$\bar{E}\bar{F}$	Y
0	0	0	0	0	1	0	1	1
0	0	0	1	0	1	0	1	1
0	0	1	0	0	1	0	1	1
0	0	1	1	0	1	0	0	0
0	1	0	0	0	0	0	1	1
0	1	0	1	0	0	0	1	1
0	1	1	0	0	0	0	1	1
0	1	1	1	0	0	0	0	0
1	0	0	0	0	0	0	1	1
1	0	0	1	0	0	0	1	1
1	0	1	0	0	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	1	0	0	1	1
1	1	0	1	1	0	0	1	1
1	1	1	0	1	0	0	1	1
1	1	1	1	1	0	0	0	0



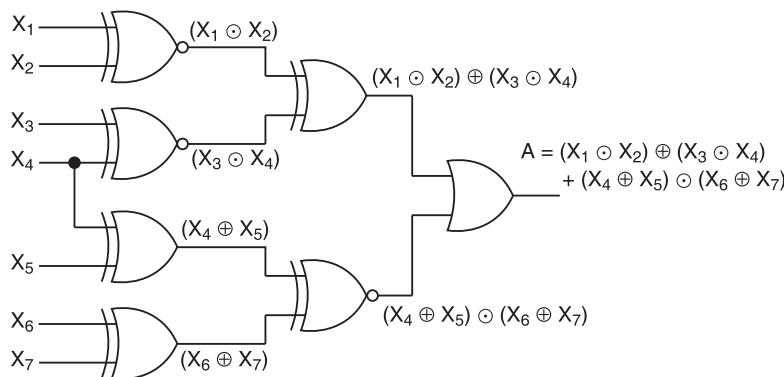
A	B	C	D	$\bar{A}B$	$C\bar{D}$	$ABC$	Z
0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	1
0	0	1	0	0	1	0	0
0	0	1	1	0	0	0	1
0	1	0	0	1	0	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	1	0	0
0	1	1	1	1	0	0	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	1
1	0	1	0	0	1	0	0
1	0	1	1	0	0	0	1
1	1	0	0	0	0	0	1
1	1	0	1	0	0	0	1
1	1	1	0	0	1	1	0
1	1	1	1	0	0	1	0

**1030 ANSWERS**

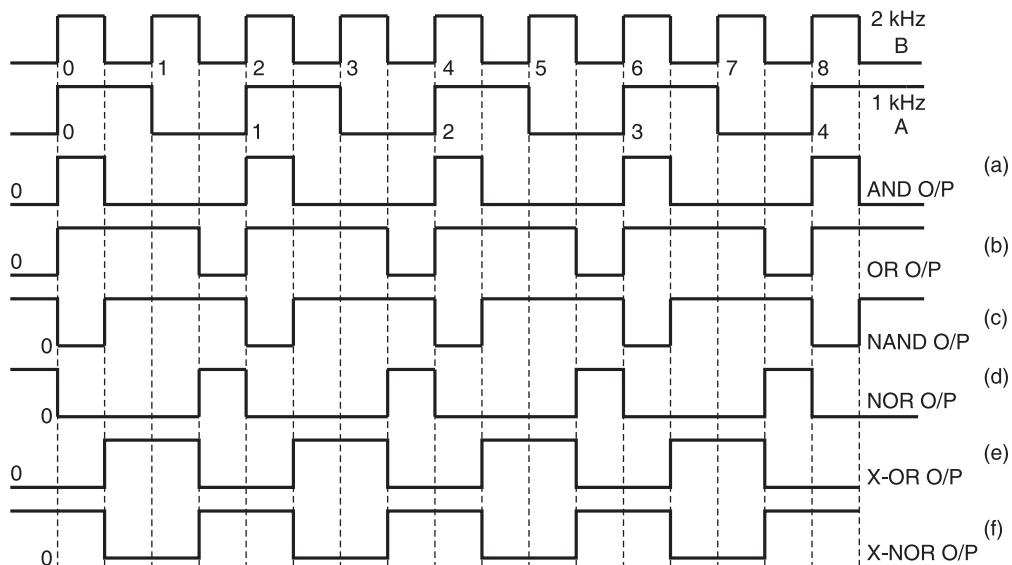
**4.3 (a)**



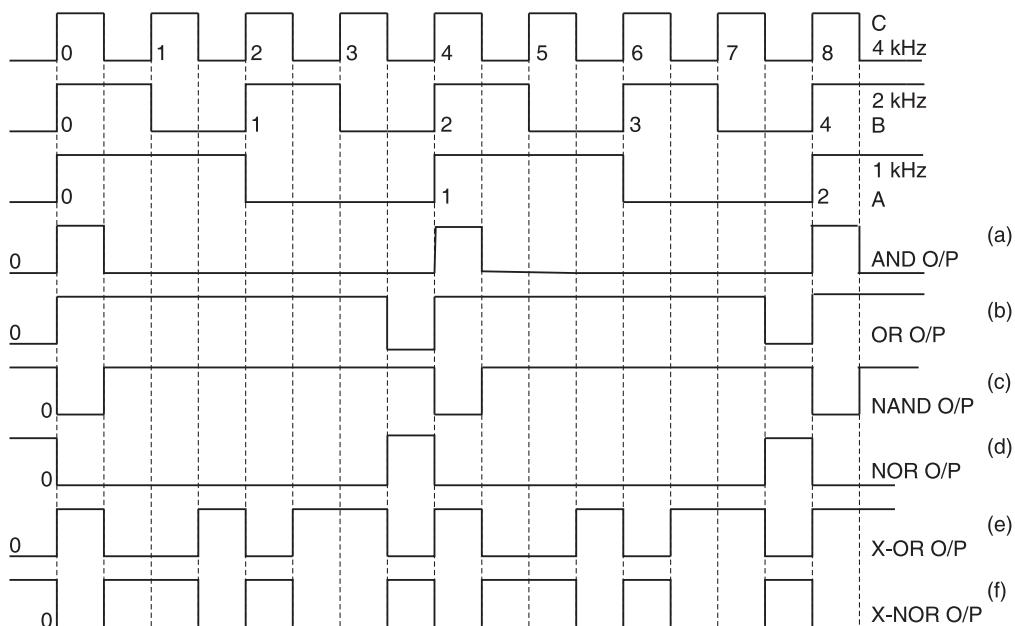
**(b)**



**4.4**



4.5



## CHAPTER 5

### Answers to Fill in the Blanks

1. (a) AND operation, (b) OR operation, (c) NOT operation
2. corresponding element of hardware
3. SOP, POS
4. active-LOW input
5. an AND gate
6. an OR gate
7.  $A + B = B + A; AB = BA$
8.  $(A + B) + C = A + (B + C); (AB) \cdot C = A \cdot (BC)$
9.  $A \cdot (B + C) = AB + AC; A + BC = (A + B)(A + C)$
10.  $A + \bar{A}B = A + B; A(\bar{A} + B) = AB$
11.  $A \cdot A = A; A + A = A$
12.  $A + AB = A; A(A + B) = A$
13.  $AB + \bar{A}C + BC = AB + \bar{A}C$
14. included factor
15.  $AB + \bar{A}C = (A + C)(\bar{A} + B)$
16.  $\overline{A + B} = \bar{A} \cdot \bar{B}; \overline{AB} = \bar{A} + \bar{B}$
17.  $f(A, B, C, \dots) = A \cdot f(1, B, C, \dots) + \bar{A} \cdot f(0, B, C, \dots)$   
 $f(A, B, C, \dots) = [A + f(0, B, C, \dots)] \cdot [\bar{A} + f(1, B, C, \dots)]$
18. logic design
19. active, inactive
20. signal
21. multilevel, non uniform, logic race
22. two-level, uniform, logic race
23. SOP
24. POS

### Answers to Objective Type Questions

- |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|
| 1. b  | 2. a  | 3. a  | 4. c  | 5. d  | 6. b  | 7. c  |
| 8. b  | 9. a  | 10. b | 11. c | 12. b | 13. a | 14. b |
| 15. b | 16. c | 17. a | 18. b | 19. c | 20. d | 21. c |

## **Answers to Problems**

- | <b>5.1</b>  | (a)  | <table border="1"> <thead> <tr> <th>A</th><th>B</th><th><math>\bar{A}B</math></th><th>AB</th><th><math>A + \bar{A}B + AB</math></th><th><math>A + B</math></th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr> <td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td></tr> <tr> <td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>  | A  | B                            | $\bar{A}B$    | AB               | $A + \bar{A}B + AB$          | $A + B$ | 0                | 0                     | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |   |   |   |   |   |   |   |   |   |   |
|-------------|--|--|--|------------------------------|---------------|------------------|------------------------------|---------|------------------|-----------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A           | B  | $\bar{A}B$   | AB   | $A + \bar{A}B + AB$          | $A + B$       |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0           | 0  | 0  | 0  | 0                            | 0             |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0           | 1  | 1  | 0  | 1                            | 1             |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1           | 0  | 0  | 0  | 1                            | 1             |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1           | 1  | 0  | 1  | 1                            | 1             |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|             | (b)  | <table border="1"> <thead> <tr> <th>A</th><th>B</th><th><math>A + \bar{B}</math></th><th><math>\bar{A} + B</math></th><th><math>(A + \bar{B})(\bar{A} + B)</math></th><th>AB</th><th><math>\bar{A}\bar{B}</math></th><th><math>AB + \bar{A}\bar{B}</math></th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td></tr> <tr> <td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr> <td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr> </tbody> </table> | A  | B                            | $A + \bar{B}$ | $\bar{A} + B$    | $(A + \bar{B})(\bar{A} + B)$ | AB      | $\bar{A}\bar{B}$ | $AB + \bar{A}\bar{B}$ | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| A           | B  | $A + \bar{B}$  | $\bar{A} + B$  | $(A + \bar{B})(\bar{A} + B)$ | AB            | $\bar{A}\bar{B}$ | $AB + \bar{A}\bar{B}$        |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0           | 0  | 1  | 1  | 1                            | 0             | 1                | 1                            |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0           | 1  | 0  | 1  | 0                            | 0             | 0                | 0                            |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1           | 0  | 1  | 0  | 0                            | 0             | 0                | 0                            |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1           | 1  | 1  | 1  | 1                            | 1             | 0                | 1                            |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| <b>5.2</b>  | (a) ABC  | (b) ABC  | (c) 0  | (d) 0                        |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|             | (e) 0  | (f) 0  |  |                              |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| <b>5.3</b>  | (a) P + Q  | (b) 1  | (c) 1  | (d) 1                        |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|             | (e) 1  | (f) P  |  |                              |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| <b>5.4</b>  | (a) Y  | (b) 0  | (c) 0  | (d) XY                       |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|             | (e) 1  | (f) $Z + X\bar{Y}$   |  |                              |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| <b>5.5</b>  | (a) XYZ  | (b) XZ   | (c) 0  | (d) ABC                      |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| <b>5.6</b>  | (a) $AB + AC + \bar{BD}$   | (b) $X \cdot \bar{YZ}$   | (c) $AB + EF$  | (d) ABC                      |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|             | (e) $\bar{AB}$   | (f) $A + B + C$  | (g) $\bar{B}\bar{C}$   | (h) $\bar{B} + \bar{C}$      |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|             | (i) $W(X + \bar{Y})$   | (j) 1  | (k) $\bar{C}$  |                              |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| <b>5.8</b>  | (a) $\bar{P} + \bar{Q}\bar{R}$   |  | (b) $\bar{P}Q + R\bar{S}$  |                              |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|             | (c) $\bar{A}\bar{B}\bar{C}\bar{D} + \bar{E}\bar{F}\bar{G}\bar{H}$  |  | (d) $\bar{A}\bar{B}\bar{C}\bar{D} + A + \bar{B} + \bar{C} + D$                       |                              |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| <b>5.9</b>  | (a) $\overline{\overline{A} \cdot \overline{\overline{B} \cdot \overline{\overline{C} \cdot \overline{D}}}}$ |  | (b) $\overline{\overline{A} \cdot \overline{\overline{C}}} \overline{ABC \cdot ACD}$ |                              |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|             | $\overline{\overline{\overline{A} \cdot \overline{\overline{B} \cdot \overline{C} \cdot \overline{D}}}}$     |  |  |                              |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|             | (c) $\overline{\overline{A} \cdot \overline{\overline{B} \cdot C \cdot D}}$                                  |  | (d) $\overline{AB} \cdot CD(\overline{A \cdot \overline{B} \cdot CD})$               |                              |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| <b>5.10</b> | (a) $\overline{\overline{A} \cdot \overline{BC} \cdot \overline{ABC}}$                                       | (b) $\overline{\overline{XY} \cdot \bar{Z} \overline{\overline{XY} \cdot \overline{P}}}$   | (c) $\overline{\overline{1} \cdot \overline{A} \cdot \overline{ABC}}$                |                              |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|             | $\overline{\overline{\overline{A} \cdot \overline{BC} \cdot \overline{ABC}}}$                                |  |  |                              |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| <b>5.11</b> | (a) $X + Y + \overline{\overline{X}} + \overline{\overline{Y}}$  | (b) $\overline{\overline{X} + Y} + \overline{X + \overline{\overline{X}} + \overline{Y}}$  | (c) $\overline{(1+A)} + \overline{A} + \overline{C}$                                 |                              |               |                  |                              |         |                  |                       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

CHAPTER 6

## Answers to Fill in the Blanks

- 1.** systematic, very simple                           **2.** 5 or more, programmed  
**3.**  $2^n$    **4.** minterm                                   **5.** maxterm                                   **6.** minterm  
**7.** maxterm                                   **8.** number of gate inputs                   **9.** true  
**10.** false                                       **11.** missing minterms of the SOP form as the maxterms of the POS form  
**12.** missing maxterms of the POS form as the minterms of the SOP form  
**13.** physically adjacent to each other or can be made adjacent by wrapping

- |                                |                                     |                               |                            |
|--------------------------------|-------------------------------------|-------------------------------|----------------------------|
| 14. Gray code                  | 15. reduction of cost               | 16. squares, rectangles       | 17. SOP and POS            |
| 18. a pair                     | 19. a quad                          | 20. an octet                  | 21. adjacent to each other |
| 22. four                       | 23. don't care                      | 24. two-level                 | 25. two-level              |
| 26. hybrid                     | 27. variable mapping                | 28. looping                   | 29. index, weight          |
| 30. literal                    | 31. fully algorithmic, programmable | 32. prime implicants          |                            |
| 33. essential prime implicants | 34. prime implicant chart           |                               |                            |
| 35. columns, rows              | 36. subcube                         | 37. essential prime implicant |                            |
| 38. redundant prime implicant  |                                     | 39. selective prime implicant |                            |
| 40. false prime implicant      |                                     |                               |                            |

## **Answers to Objective Type Questions**

- 1.** b      **2.** a      **3.** b      **4.** a      **5.** c      **6.** b      **7.** c  
**8.** a      **9.** a      **10.** a      **11.** a      **12.** b      **13.** b      **14.** c  
**15.** d      **16.** b      **17.** a      **18.** d      **19.** b      **20.** c      **21.** b  
**22.** d      **23.** b      **24.** d      **25.** b      **26.** d      **27.** d      **28.** a

## Answers to Problems

**1034 ANSWERS**

**6.9** (a) (SOP form)  $AC + AD + BC + BD$

$$(POS \text{ form}) (A + B) \cdot (C + D) = \overline{\overline{(A + B)} + \overline{(C + D)}}$$

$$(b) (\text{SOP form}) \overline{B} + \overline{AD} = B \cdot \overline{\overline{AD}}$$

$$(POS \text{ form}) (\overline{A} + \overline{B})(\overline{B} + \overline{D})$$

$$(c) (\text{SOP form}) \overline{BD} + A\overline{CD} + \overline{AC} = \overline{\overline{BD} \cdot \overline{ACD} \cdot \overline{AC}}$$

$$(POS \text{ form}) (A + C + \overline{D})(\overline{A} + \overline{C} + \overline{D})(\overline{B} + D)$$

$$(d) (\text{SOP form}) \overline{BCD} + BCD + \overline{BCD} + \overline{A} \overline{BD} = \overline{\overline{BCD} \cdot \overline{BCD} \cdot \overline{BCD} \cdot \overline{ABD}}$$

$$(POS \text{ form}) (\overline{A} + B + D)(B + \overline{C} + \overline{D})(\overline{B} + \overline{C} + D)(\overline{B} + C + \overline{D})$$

$$(e) (\text{SOP form}) \overline{AC} + BC + A\overline{BD} = \overline{\overline{AC} \cdot BC \cdot A\overline{BD}}$$

$$(POS \text{ form}) (A + B + \overline{C})(\overline{A} + \overline{B} + C)(\overline{A} + D)$$

$$(f) (\text{SOP form}) AB + AD + AC$$

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

**6.10** (a) (SOP form)  $ABE + A\overline{B}\overline{CD} + B\overline{D}\overline{E} + \overline{A} \overline{B}CD \overline{E}$

$$(POS \text{ form}) (B + D)(\overline{B} + E)(A + \overline{D} + \overline{E})(\overline{A} + B + \overline{C})(A + B + C)$$

$$(b) (\text{SOP form}) \overline{CD} + A\overline{BD} + \overline{BCE} + \overline{ABCD}$$

$$(POS \text{ form}) (A + B + \overline{C})(\overline{C} + \overline{D})(A + D + \overline{E})(\overline{A} + \overline{B} + D)(\overline{B} + C + D)$$

$$(c) (\text{SOP form}) \overline{ABCD}\overline{E} + \overline{ACD}\overline{E}\overline{F} + \overline{A} \overline{BCDE} + \overline{A} \overline{B}\overline{CDF} + \overline{AB}\overline{CDF} + \overline{A} \overline{BD}\overline{E} + \overline{B}\overline{DE}\overline{F} \\ + \overline{BCD}\overline{E}\overline{F} + A\overline{BCD}$$

$$(POS \text{ form}) (B + \overline{C} + \overline{D} + E)(C + \overline{D} + \overline{E} + F)(A + B + D + \overline{E} + \overline{F})(\overline{B} + D + F)$$

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

$$(\overline{A} + \overline{D} + \overline{E})(\overline{A} + \overline{B})$$

$$(d) (\text{SOP form}) \overline{BCDF} + \overline{BCDF} + A\overline{BD}\overline{E} + \overline{A} \overline{BC}\overline{E} + \overline{AC}\overline{EF} + A\overline{D}\overline{EF} + A\overline{BC}\overline{DF} + \overline{ABC}\overline{F} \\ + \overline{ABC}\overline{DE} + B\overline{C}\overline{D}\overline{E}\overline{F}$$

$$(POS \text{ form}) (\overline{C} + \overline{E} + \overline{F})(A + \overline{C} + \overline{D} + \overline{E})(\overline{A} + \overline{B} + \overline{C} + \overline{E})(B + C + D + F)(\overline{B} + C + \overline{D})$$

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

**6.11** (a) (SOP form)  $A\overline{CD} + BD + \overline{A}\overline{C} + \overline{AD}$

$$(POS \text{ form}) (A + \overline{C} + D)(\overline{A} + B + \overline{D})(\overline{A} + C + D)$$

$$(b) (\text{SOP form}) \overline{BD} + CD + \overline{AC} + A\overline{CD}$$

$$(POS \text{ form}) (C + \overline{D})(A + \overline{B} + C)(\overline{A} + \overline{B} + \overline{C} + D)$$

$$(c) (\text{SOP form}) BD + \overline{AB} + \overline{AC} + \overline{AD}$$

$$(POS \text{ form}) (B + \overline{C} + D)(\overline{A} + D)(\overline{A} + B)$$

$$(d) (\text{SOP form}) \overline{A}\overline{B} + B\overline{C}$$

$$(POS \text{ form}) \overline{A}(\overline{B} + \overline{C})$$

$$(e) (\text{SOP form}) \overline{A}\overline{BC} + A\overline{BC} + \overline{ACD} + ACD$$

$$(POS \text{ form}) (A + C + \overline{D})(\overline{A} + \overline{C} + \overline{D})(A + \overline{B} + \overline{C})(\overline{A} + \overline{B} + C)$$

(f) (SOP form)  $\bar{B}\bar{D} + \bar{B}\bar{C} + \bar{A}\bar{C}\bar{D}$   
 (POS form)  $(\bar{C} + \bar{D})(\bar{A} + B)(A + C + D)$

**6.12**  $F = \bar{A}\bar{B} + AB + BD + CD$  for AND-NOR and NAND-AND forms

$F = (\bar{A} + B + D)(A + \bar{B} + D)(B + C)(C + D)$  for OR-NAND and NOR-OR forms

**6.13**  $F = \bar{ABC} + \bar{ACD} + \bar{ACD}$  for AND-NOR and NAND-AND forms

$F = (A + D)(\bar{C} + \bar{D})(\bar{A} + B + C)$  for OR-NAND and NOR-OR forms

**6.14** (a)  $f_{1m} = \bar{ACD} + A\bar{BD} + \bar{ACD} + A\bar{BC}$

$$f_{2m} = \bar{ACD} + A\bar{BD} + \bar{BCD} + ABCD$$

$$f_c = \bar{ACD} + A\bar{BD}$$

(b)  $f_{1m} = \bar{AD} + \bar{BC} + A\bar{BD}$

$$f_{2m} = \bar{AD} + \bar{BCD} + \bar{AB} + \bar{BCD}$$

$$f_c = \bar{AD}$$

(c)  $f_{1m} = \bar{ABC} + \bar{AD} + BC + ACD$

$$f_{2m} = \bar{ABC} + A\bar{BD} + \bar{BCD} + ACD$$

$$f_c = ACD$$

(d)  $f_{1m} = \bar{BC} + ABC + \bar{AD}$

$$f_{2m} = \bar{AC} + BD + ABC$$

$$f_c = ABC$$

**6.15** (a)  $AC + BCD + ABD$

(b)  $ACD + AB\bar{E} + \bar{ABC}$

(c)  $\bar{BC} + \bar{ACD} + \bar{BCD} + ABC$

(d)  $BD + \bar{CD} + \bar{AC}$

(e)  $A\bar{BD} + BCD + \bar{AE} + \bar{BC} + \bar{ABC}$

(f)  $\bar{BCD} + A\bar{BD} + B\bar{CD}$

**6.16** (a)  $A\bar{DE} + BCF + A\bar{BD} + \bar{A}\bar{BD} + \bar{ABCD}$

(b)  $\bar{A}\bar{BD} + AB\bar{CD} + ABCD + B\bar{CDF} + BCDG + AC\bar{DE} + A\bar{BC}\bar{E} + A\bar{BCDF}$

(c)  $CDF + ACDG + BCF + \bar{BCDH} + \bar{ABC}G + \bar{A}\bar{BCD}$

(d)  $BCD\bar{E} + \bar{ABC}G + \bar{BCDF} + \bar{BCDE} + A\bar{BD} + \bar{A}\bar{BCDH}$

(e)  $\bar{ACDE} + \bar{ABD} + A\bar{BD} + AC\bar{DF}$

(f)  $\bar{A}\bar{BDE} + \bar{A}\bar{BC} + \bar{BCDG} + ABDF + ABC + \bar{ACD}\bar{E}$

## CHAPTER 7

### Answers to Fill in the Blanks

- |                   |                       |                     |                    |
|-------------------|-----------------------|---------------------|--------------------|
| 1. half-adder     | 2. full-adder         | 3. half-subtractor  | 4. full-subtractor |
| 5. parallel adder | 6. ripple-carry adder | 7. look-ahead-carry | 8. serial          |
| 9. comparator     | 10. decoder           | 11. enable          | 12. an encoder     |

## 1036 ANSWERS

- |   |                             |              |                   |
|---|-----------------------------|--------------|-------------------|
| 13. priority encoder                          | 14. multiplexer             | 15. 16:1     | 16. demultiplexer |
| 17. distributor                               | 18. X-NOR                   | 19. encoding | 20. multiplexing  |
| 21. time multiplexing, frequency multiplexing |                             | 22. decoder  | 23. 6             |
| 24. 3, 8                                      | 25. binary-to-octal, 1-of-8 |              | 26. 4, 10         |
| 27. 4, 10: 1, 10                              | 28. 8, 3                    | 29. 10, 4    | 30. 5             |
| 31. 5   | 32. 9                       | 33. 12       |                   |

### Answers to Objective Type Questions

- |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|
| 1. c  | 2. b  | 3. c  | 4. d  | 5. b  | 6. c  | 7. c  |
| 8. b  | 9. d  | 10. b | 11. a | 12. d | 13. c | 14. a |
| 15. b | 16. b | 17. c | 18. d | 19. b | 20. d | 21. d |
| 22. c | 23. a | 24. b | 25. d | 26. a | 27. b | 28. a |
| 29. c | 30. c | 31. b | 32. a | 33. c | 34. d | 35. b |
| 36. c | 37. a | 38. a | 39. a | 40. c | 41. d | 42. a |
| 43. b | 44. b | 45. b | 46. d | 47. d | 48. c | 49. d |

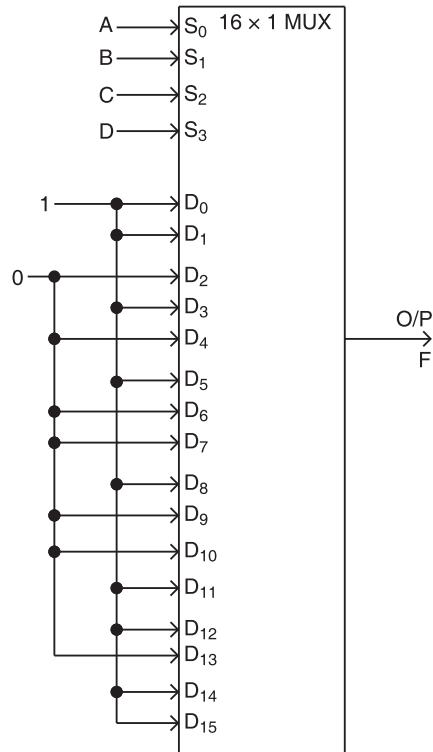
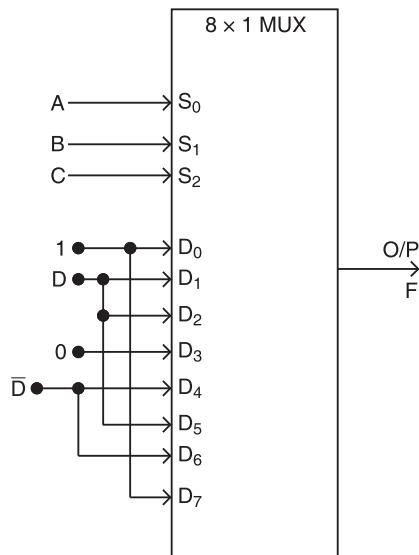
### Answers to Problems

- 7.1** If A, B, C and D are the BCD inputs and  $f_3, f_2, f_1$  and  $f_0$  are the 2421 outputs, then  
 $f_3 = A + BC + BD, f_2 = A + BC + \bar{B}\bar{D}, f_1 = A + \bar{B}C + \bar{B}\bar{C}D, f_0 = D$
- 7.2** If A, B, C and D are the 2421 inputs and  $f_4, f_3, f_2, f_1$  and  $f_0$  are the 51111 outputs, then  
 $f_4 = A, f_3 = B, f_2 = \bar{A}B + BC + BD + \bar{A}\bar{C}D, f_1 = \bar{A}\bar{B} + A\bar{C} + BC,$   
 $f_0 = \bar{A}\bar{B} + \bar{A}\bar{C} + \bar{A}\bar{D} + BCD$
- 7.3** If A, B, C, and D are the excess-3 code inputs, the decimal digits are given by
- |                                      |                                      |                                      |                                      |
|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|
| $D_0 = \bar{A}\bar{B}\bar{C}\bar{D}$ | $D_1 = \bar{A}\bar{B}\bar{C}\bar{D}$ | $D_2 = \bar{A}\bar{B}\bar{C}\bar{D}$ | $D_3 = \bar{A}\bar{B}\bar{C}\bar{D}$ |
| $D_4 = \bar{A}\bar{B}\bar{C}\bar{D}$ | $D_5 = A\bar{B}\bar{C}\bar{D}$       | $D_6 = A\bar{B}\bar{C}\bar{D}$       | $D_7 = A\bar{B}\bar{C}\bar{D}$       |
| $D_8 = A\bar{B}\bar{C}\bar{D}$       | $D_9 = AB\bar{C}\bar{D}$             |                                      |                                      |
- 7.4** If A, B, C and D are the 2421 code bits, the decimal digits are given by
- |                                      |                                      |                                      |                                      |
|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|
| $D_0 = \bar{A}\bar{B}\bar{C}\bar{D}$ | $D_1 = \bar{A}\bar{B}\bar{C}\bar{D}$ | $D_2 = \bar{A}\bar{B}\bar{C}\bar{D}$ | $D_3 = \bar{A}\bar{B}\bar{C}\bar{D}$ |
| $D_4 = \bar{A}\bar{B}\bar{C}\bar{D}$ | $D_5 = A\bar{B}\bar{C}\bar{D}$       | $D_6 = AB\bar{C}\bar{D}$             | $D_7 = AB\bar{C}\bar{D}$             |
| $D_8 = ABC\bar{D}$                   | $D_9 = ABCD$                         |                                      |                                      |
- 7.5** If A, B, C and D are the 2421 code bits,  $f_e = (A \oplus B) + (C \oplus D)$
- 7.6** If A, B, C and D are the 3321 BCD code bits,  $f_0 = (A \oplus B) + (C \odot D)$
- 7.7** (a) If A, B, C and D are the 5211 inputs, and  $f_3, f_2, f_1$  and  $f_0$  are the 2421 outputs,  
 $f_3 = A, f_2 = AB + \bar{C}\bar{D} + BC, f_1 = \bar{A}\bar{B}C + ABC + \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C}, f_0 = \bar{C}D + AD + A\bar{C}$
- (b) If A, B, C and D are the binary inputs and  $f_6, f_5, f_4, f_3, f_2, f_1$  and  $f_0$  are the Excess-3 outputs, then  
 $f_6 = AB + AC, f_5 = \bar{A} + \bar{B}\bar{C} = f_4, f_3 = A\bar{B}\bar{C} + \bar{A}\bar{B}D + \bar{A}\bar{B}C + BCD,$   
 $f_2 = \bar{B}D + \bar{B}\bar{C}\bar{D} + ABD + ABC + \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C}, f_1 = \bar{A}\bar{C}\bar{D} + \bar{A}\bar{C}D + A\bar{C}\bar{D} + A\bar{B}\bar{D} + AB\bar{C}D, f_0 = \bar{D}$
- (c) If A, B, C and D are the BCD inputs and  $G_3, G_2, G_1$  and  $G_0$  are the gray code outputs, then  
 $G_3 = A, G_2 = A + B, G_1 = B\bar{C} + \bar{B}C, G_0 = \bar{C}D + CD$

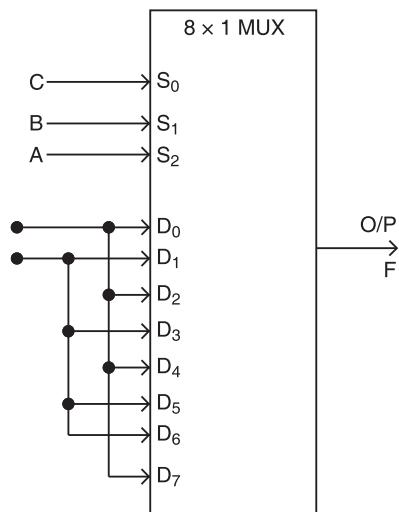
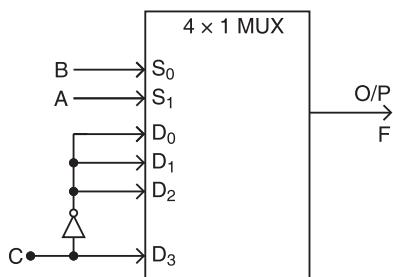
$$7.8 \text{ (a)} f = A \oplus B \oplus C$$

$$(b) f = \overline{A \oplus B \oplus C}$$

7.9



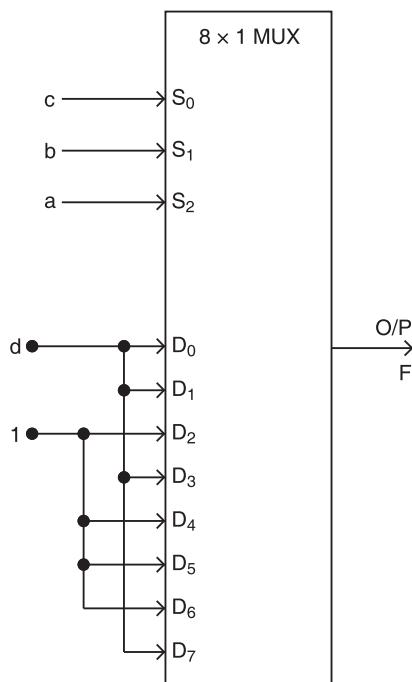
7.10



**1038 ANSWERS**

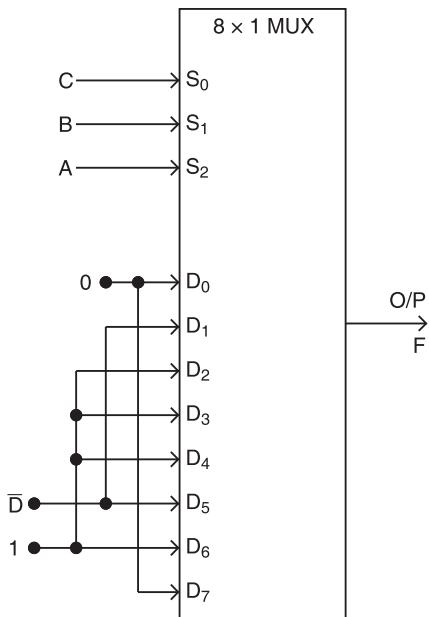
**7.11**

$S_3$	$S_2$	$S_1$	$S_0$	$F$
a	b	c	d	
0	0	0	0	0 $F = d$
0	0	0	1	1
0	0	1	0	0 $F = d$
0	0	1	1	1
0	1	0	0	1 $F = 1$
0	1	0	1	1
0	1	1	0	0 $F = d$
0	1	1	1	1
1	0	0	0	1 $F = 1$
1	0	0	1	1
1	0	1	0	1 $F = 1$
1	0	1	1	1
1	1	0	0	1 $F = 1$
1	1	0	1	1
1	1	1	0	0 $F = d$
1	1	1	1	1



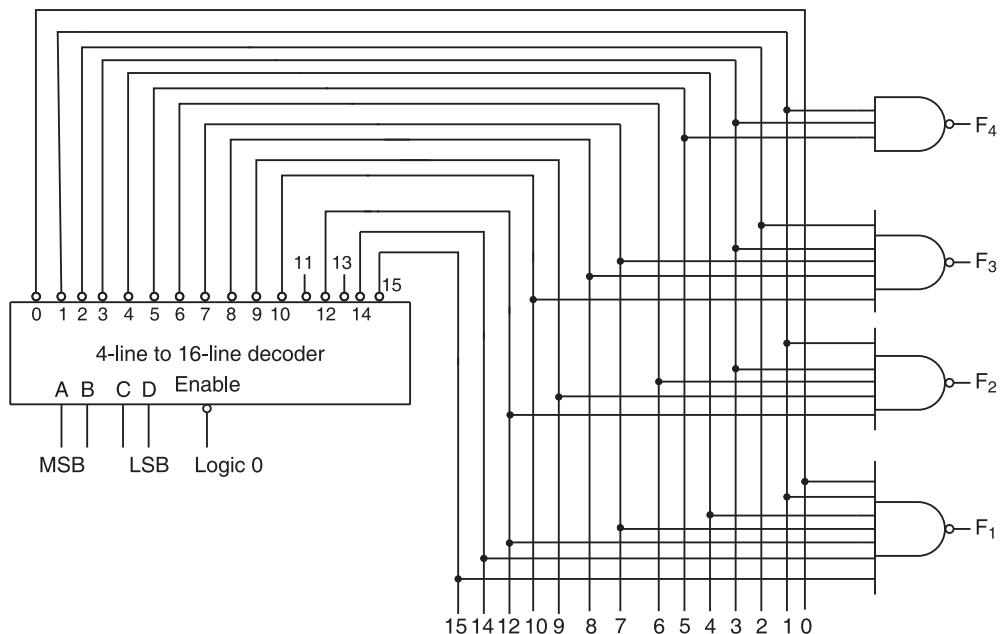
(b) Logic diagram

**7.12**

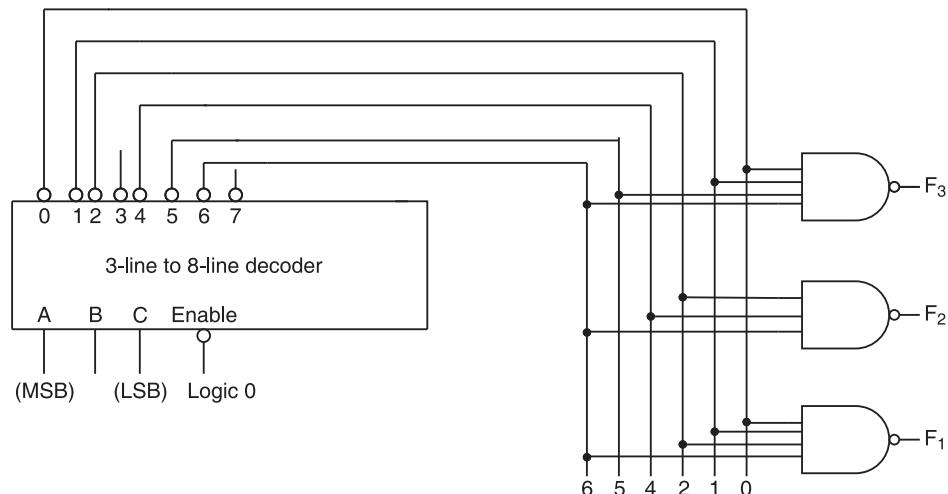


(b) Logic diagram

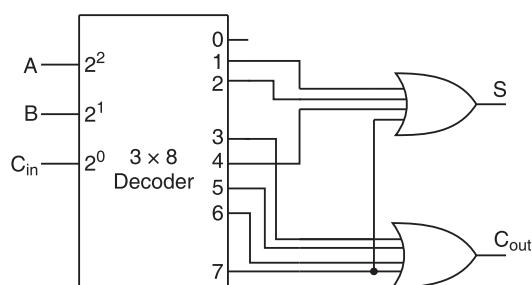
7.13



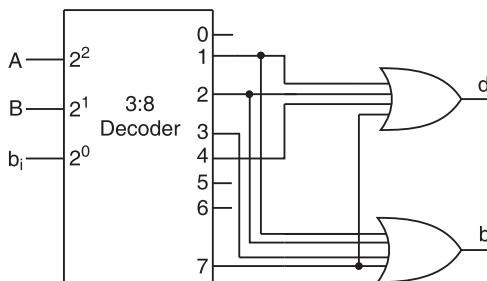
7.14



7.15



7.16



## CHAPTER 8

## Answers to Fill in the Blanks

- |                              |                                |                    |                   |
|------------------------------|--------------------------------|--------------------|-------------------|
| 1. memory device             | 2. PROM                        | 3. $5 \times 32$   | 4. $4 \times 16$  |
| 5. MROM, PROM, EPROM, EEPROM | 6. MROM                        | 7. 32              |                   |
| 8. 17                        | 9. ultraviolet light           | 10. EEPROM         | 11. EPROM, EEPROM |
| 12. fuse blowing             | 13. PLD                        | 14. PROM, PAL, PLA | 15. AND, OR       |
| 16. OR, AND                  | 17. programmable, programmable |                    | 18. FPLA          |
| 19. programming table        |                                |                    |                   |

## Answers to Objective Type Questions

- |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|
| 1. b  | 2. a  | 3. a  | 4. b  | 5. a  | 6. b  | 7. d  |
| 8. b  | 9. c  | 10. d | 11. c | 12. b | 13. b | 14. c |
| 15. a | 16. d | 17. a | 18. b | 19. b | 20. c | 21. d |
| 22. d | 23. a | 24. c | 25. b | 26. d | 27. c | 28. b |
| 29. a | 30. c | 31. a | 32. b | 33. d | 34. a | 35. d |
| 36. c | 37. a | 38. b | 39. c |       |       |       |

## Answers to Problems

## 8.1 For PROM

$$\begin{aligned}
 E_3 &= \bar{B}_3 B_2 \bar{B}_1 B_0 + \bar{B}_3 B_2 B_1 \bar{B}_0 + \bar{B}_3 B_2 B_1 B_0 + B_3 \bar{B}_2 \bar{B}_1 \bar{B}_0 + B_3 \bar{B}_2 \bar{B}_1 B_0 \\
 E_2 &= \bar{B}_3 \bar{B}_2 \bar{B}_1 B_0 + \bar{B}_3 \bar{B}_2 B_1 \bar{B}_0 + \bar{B}_3 \bar{B}_2 B_1 B_0 + \bar{B}_3 B_2 \bar{B}_1 \bar{B}_0 + B_3 \bar{B}_2 \bar{B}_1 B_0 \\
 E_1 &= \bar{B}_3 \bar{B}_2 \bar{B}_1 \bar{B}_0 + \bar{B}_3 \bar{B}_2 B_1 B_0 + \bar{B}_3 B_2 \bar{B}_1 \bar{B}_0 + \bar{B}_3 B_2 B_1 B_0 + B_3 \bar{B}_2 \bar{B}_1 \bar{B}_0 \\
 E_0 &= \bar{B}_3 \bar{B}_2 \bar{B}_1 \bar{B}_0 + \bar{B}_3 \bar{B}_2 B_1 \bar{B}_0 + \bar{B}_3 B_2 \bar{B}_1 \bar{B}_0 + \bar{B}_3 B_2 B_1 \bar{B}_0 + B_3 \bar{B}_2 \bar{B}_1 \bar{B}_0
 \end{aligned}$$

For PAL

$$\begin{aligned}
 E_3 &= B_3 + B_2 B_1 + B_2 B_0 \\
 E_2 &= B_2 \bar{B}_1 \bar{B}_0 + \bar{B}_2 B_0 + \bar{B}_2 B_1 \\
 E_1 &= \bar{B}_1 \bar{B}_0 + B_1 B_0 \\
 E_0 &= \bar{B}_0
 \end{aligned}$$

For PLA

$$\begin{aligned}E_3(T) &= B_3 + B_2B_1 + B_2B_0 \\E_2(C) &= \overline{\overline{B}_2\overline{B}_1\overline{B}_0 + B_2B_1 + B_2B_0} \\E_1(T) &= \overline{\overline{B}_1\overline{B}_0} + B_1B_0 \\E_0(T) &= \overline{B}_0\end{aligned}$$

### 8.2 For PROM

$$\begin{aligned}B_3 &= E_3\overline{E}_2E_1E_0 + E_3E_2\overline{E}_1\overline{E}_0 \\B_2 &= \overline{E}_3E_2E_1E_0 + E_3\overline{E}_2\overline{E}_1\overline{E}_0 + E_3\overline{E}_2\overline{E}_1E_0 + E_3\overline{E}_2E_1\overline{E}_0 \\B_1 &= \overline{E}_3E_2\overline{E}_1E_0 + \overline{E}_3E_2E_1\overline{E}_0 + E_3\overline{E}_2\overline{E}_1E_0 + E_3\overline{E}_2E_1\overline{E}_0 \\B_0 &= \overline{E}_3E_2\overline{E}_1\overline{E}_0 + \overline{E}_3E_2E_1\overline{E}_0 + E_3\overline{E}_2\overline{E}_1\overline{E}_0 + E_3\overline{E}_2E_1\overline{E}_0 + E_3E_2\overline{E}_1\overline{E}_0\end{aligned}$$

For PAL

$$\begin{aligned}B_3 &= E_3E_2 + E_3E_1E_0 \\B_2 &= E_2E_1E_0 + \overline{E}_2\overline{E}_0 + \overline{E}_2\overline{E}_1 \\B_1 &= \overline{E}_1E_0 + E_1\overline{E}_0 \\B_0 &= \overline{E}_0\end{aligned}$$

For PLA

$$\begin{aligned}B_3(C) &= \overline{E}_3 + E_1\overline{E}_0 + \overline{E}_2\overline{E}_1 \\B_2(T) &= E_2E_1E_0 + \overline{E}_2\overline{E}_0 + \overline{E}_2\overline{E}_1 \\B_1(T) &= \overline{E}_1E_0 + E_1\overline{E}_0 \\B_0(T) &= \overline{E}_0\end{aligned}$$

### 8.3 $F_1(C) = A\overline{C} + AB + B\overline{C}$

$$F_2(C) = A\overline{C} + B\overline{C} + \overline{A}\overline{B}C$$

Product term	Inputs			Outputs	
	A	B	C	(C)	(C)
	F <sub>1</sub>	F <sub>2</sub>			
A $\overline{C}$	1	—	0	1	1
B $\overline{C}$	—	1	0	1	1
AB	1	1	—	1	—
$\overline{A}\overline{B}C$	0	0	1	—	1

## 8.4

Product term	AND inputs					Outputs
	W	X	Y	Z	$F_3$	
1	—	0	—	0	—	$F_1 = \bar{x}\bar{z} + wx\bar{y} + \bar{w}xz$
2	1	1	0	—	—	
3	0	1	—	1	—	
4	—	—	—	—	1	$F_2 = F_3 + \bar{w}y\bar{z} + w\bar{x}\bar{y}$
5	0	—	1	0	—	
6	1	0	0	—	—	
7	—	0	0	0	—	$F_3 = \bar{x}\bar{y}\bar{z} + wxy$
8	1	1	1	—	—	
9	—	—	—	—	—	
10	—	0	0	—	—	$F_4 = \bar{x}\bar{y} + \bar{x}\bar{z} + \bar{w}\bar{y}$
11	—	0	—	0	—	
12	0	—	0	—	—	

$$8.5 \quad F_1(T) = \bar{C} + \bar{A}\bar{B}$$

$$F_2(T) = \bar{A}\bar{C} + AB$$

$$F_3(C) = \bar{B} + \bar{A}\bar{C} + AC$$

$$F_4(C) = \bar{C}$$

Product term	Inputs				Outputs			
	(T)	(T)	(C)	(C)	$F_1$	$F_2$	$F_3$	$F_4$
	A	B	C	D				
$\bar{C}$	—	—	0	—	1	—	—	1
$\bar{A}\bar{B}$	0	0	—	—	1	—	—	—
$\bar{A}\bar{C}$	0	—	0	—	—	1	1	—
AB	1	1	—	—	—	1	—	—
$\bar{B}$	—	0	—	—	—	—	1	—
AC	1	—	1	—	—	—	1	—

## CHAPTER 9

## Answers to Fill in the Blanks

- |                 |              |                   |                   |
|-----------------|--------------|-------------------|-------------------|
| 1. lower        | 2. difficult | 3. cannot         | 4. uncomplemented |
| 5. complemented | 6. unique    | 7. unate          | 8. -5             |
| 9. $\geq 3$     | 10. OR       | 11. non-threshold | 12. threshold     |

**Answers to Objective Type Questions**

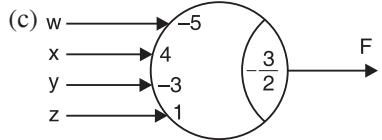
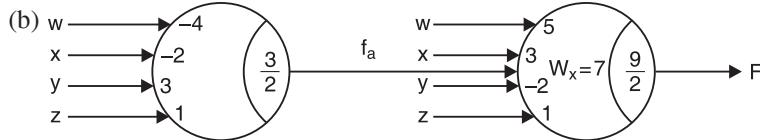
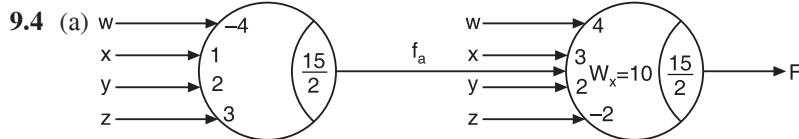
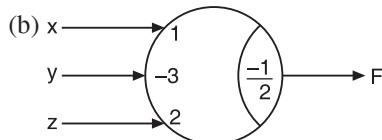
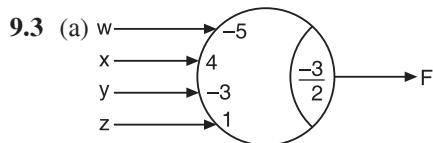
- 1.** a      **2.** c      **3.** b      **4.** d      **5.** a      **6.** a      **7.** b  
**8.** c      **9.** a      **10.** d      **11.** d      **12.** d      **13.** c      **14.** a  
**15.** b

**Answers to Problems****9.1** (a) Unate function

(b) Unate function

**9.2** (a) No

(b) No

**CHAPTER 10****Answers to Fill in the Blanks**

- 1.** flip-flop      **2.** bistable multivibrator      **3.** two      **4.** unclocked flip-flop  
**5.** asynchronous, synchronous      **6.** NAND, NOR  
**7.** active-LOW, active-HIGH      **8.** active-HIGH      **9.** active-LOW

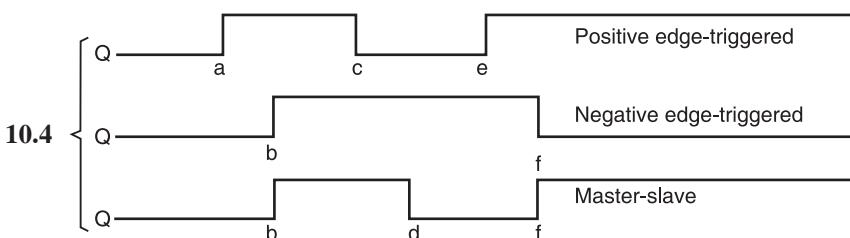
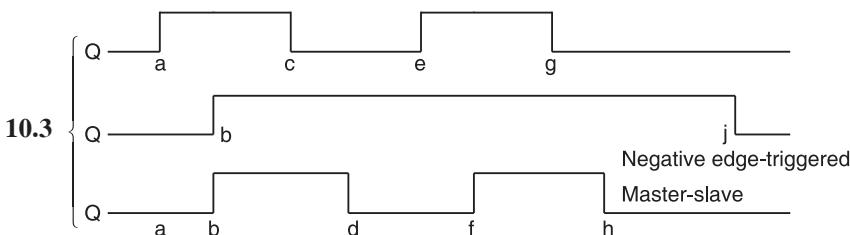
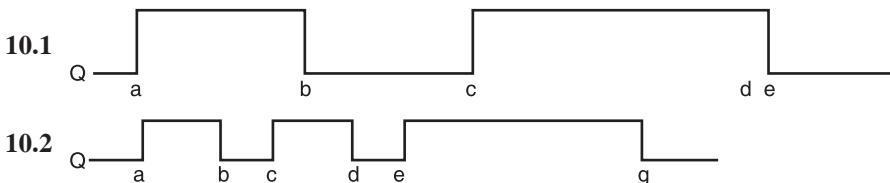
**1044** ANSWERS

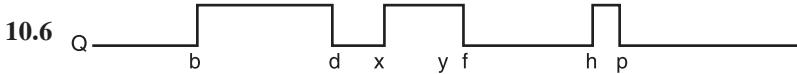
- |                             |                             |                             |                  |
|-----------------------------|-----------------------------|-----------------------------|------------------|
| 10. transparent             | 11. level, edge             | 12. dynamic                 | 13. J-K          |
| 14. T                       | 15. PRESET, CLEAR           | 16. level triggered         |                  |
| 17. negative edge-triggered |                             | 18. positive edge-triggered |                  |
| 19. clock skew              | 20. is not                  | 21. toggle                  | 22. 1, 0         |
| 23. race around condition   |                             | 24. trigger                 | 25. pulse        |
| 26. master-slave            | 27. characteristic equation |                             | 28. J-K, D       |
| 29. asynchronous            | 30. overriding              | 31. back, forth             | 32. stable state |
| 33. excitations             |                             |                             |                  |

## **Answers to Objective Type Questions**

- |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|
| 1. c  | 2. b  | 3. a  | 4. a  | 5. d  | 6. c  | 7. c  |
| 8. d  | 9. d  | 10. c | 11. a | 12. d | 13. d | 14. c |
| 15. b | 16. c | 17. d | 18. b | 19. c | 20. c | 21. b |
| 22. a | 23. d | 24. b | 25. c | 26. c | 27. c | 28. d |
| 29. c | 30. d | 31. b | 32. d | 33. a | 34. a | 35. a |
| 36. a |       |       |       |       |       |       |

## Answers to Problems





**10.7** 0 1 1 1 0 0 0 0 0

## CHAPTER 11

### Answers to Fill in the Blanks

- |   |                            |                             |
|---|----------------------------|-----------------------------|
| 1. sequentially                         | 2. simultaneously          | 3. only one bit of data     |
| 4. flip-flop                            | 5. register                | 6. buffer registers         |
| 8. serial form, parallel form           |                            | 9. serial-in, serial-out    |
| 10. serial-in, parallel-out             |                            | 11. parallel-in, serial-out |
| 12. parallel-in, parallel-out           |                            | 13. bidirectional           |
| 15. static                              | 16. static                 | 14. universal               |
| 19. speed                               | 20. dynamic shift register | 17. MOS inverters           |
| 21. small power consumption, simplicity |                            | 18. dynamic MOS             |
| 23. UART                                | 24. static                 | 22. data transfer is slow   |

### Answers to Objective Type Questions

1. c      2. b      3. d

## CHAPTER 12

### Answers to Fill in the Blanks

- |                        |  |                      |
|------------------------|--|----------------------|
| 1. counter             | 2. asynchronous (ripple), synchronous (parallel) | 3. ripple            |
| 4. serial (series)     | 5. parallel                                      | 6. state             |
| 8. shortened modulus   | 9. full modulus                                  | 10. variable modulus |
| 12. faster             | 13. more   | 14. lock-out         |
| 16. basic ring counter | 17. Johnson                                      | 18. twisted, basic   |
| 20. basic              | 21. decoding                                     | 22. 11111, 00001     |
| 24. glitches           | 25. presetting                                   | 26. 7                |
| 28. sequential         | 29. synchronous                                  | 30. asynchronous     |

### Answers to Objective Type Questions

- |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|
| 1. d  | 2. c  | 3. b  | 4. c  | 5. b  | 6. d  | 7. b  |
| 8. d  | 9. b  | 10. c | 11. d | 12. d | 13. a | 14. b |
| 15. c | 16. b | 17. c | 18. d | 19. d |       |       |

### Answers to Problems

- 12.1** (a) Connect all Js and Ks to 1. Provide a RESET pulse  $\bar{R} = \overline{Q_3 Q_2 Q_1}$ .
- (b) Connect all Js and Ks to 1. Provide a RESET pulse  $\bar{R} = \overline{Q_4 Q_3}$ .
- (c) Design equations are  $J_4 = Q_3 Q_2 Q_1$ ,  $K_4 = 1$ ,  $J_3 = Q_2 Q_1$ ,  $K_3 = Q_2 Q_1$ ,  $J_2 = Q_1$ ,  $K_2 = Q_1$ ,  $J_1 = \bar{Q}_4$ ,  $K_1 = 1$
- (d)  $S_3 = Q_2 Q_1$ ,  $R_3 = Q_2 \bar{Q}_1$ ,  $S_2 = \bar{Q}_2 Q_1$ ,  $R_2 = Q_3 Q_2 + Q_2 Q_1$ ,  $S_1 = \bar{Q}_2 \bar{Q}_1 + \bar{Q}_3 \bar{Q}_1$ ,  $R_1 = Q_1$
- (e)  $T_4 = Q_4 Q_2 Q_1 + Q_3 Q_2 Q_1$ ,  $T_3 = \bar{Q}_4 Q_2 Q_1$ ,  $T_2 = Q_1$ ,  $T_1 = 1$
- (f)  $D_4 = Q_4 \bar{Q}_3 + Q_4 \bar{Q}_1 + Q_3 Q_2 Q_1$ ,  $D_3 = \bar{Q}_4 Q_3 \bar{Q}_2 + \bar{Q}_3 Q_2 Q_1 + Q_3 \bar{Q}_1$   
 $D_2 = Q_2 \bar{Q}_3 + \bar{Q}_4 \bar{Q}_2 Q_1 + \bar{Q}_3 \bar{Q}_2 Q_1$ ,  $D_1 = \bar{Q}_1$
- 12.2**  $J_3 = 1$ ,  $K_3 = Q_2$ ,  $J_2 = Q_1$ ,  $K_2 = \bar{Q}_3$ ,  $J_1 = Q_3$ ,  $K_1 = Q_2$ ; yes;  $R = \bar{Q}_3 \bar{Q}_2 + \bar{Q}_3 Q_1 + Q_3 Q_2 \bar{Q}_1$
- 12.3** (a)  $D_3 = \bar{Q}_3 Q_2 \bar{Q}_1 + Q_3 \bar{Q}_2 \bar{Q}_1$ ,  $D_2 = \bar{Q}_2 \bar{Q}_1$ ,  $D_1 = 0$
- (b)  $T_4 = Q_4 \bar{Q}_3 + Q_4 Q_2 + \bar{Q}_2 Q_1 + Q_3 Q_1 + \bar{Q}_3 Q_2 \bar{Q}_1$ ,  $T_3 = \bar{Q}_4 \bar{Q}_2 + Q_2 Q_1 + Q_3 Q_2$   
 $T_2 = \bar{Q}_3 \bar{Q}_1 + Q_4 Q_3 + Q_3 Q_1$ ,  $T_1 = \bar{Q}_4 Q_3 + \bar{Q}_4 \bar{Q}_2 Q_1 + \bar{Q}_4 Q_2 \bar{Q}_1$
- (c)  $T_3 = \bar{Q}_2 \bar{Q}_1$ ,  $T_2 = Q_2 + Q_3 \bar{Q}_1$ ,  $T_1 = \bar{Q}_3 \bar{Q}_2 + Q_1$ ; yes, self starting
- (d)  $T_3 = Q_2 + \bar{X}$ ,  $T_2 = Q_3 + X$ ,  $T_1 = \bar{X} Q_2$
- 12.4** (a)  $J_3 = Q_2$ ,  $K_3 = \bar{Q}_1$ ,  $J_2 = \bar{Q}_3 \bar{Q}_1$ ,  $K_2 = 1$ ,  $J_1 = 1$ ,  $K_1 = \bar{Q}_2$
- (b)  $T_3 = Q_2 \bar{Q}_1$ ,  $T_2 = \bar{Q}_3 + Q_2 + Q_1$ ,  $T_1 = Q_3 + Q_2 Q_1$
- (c)  $f_1 = \bar{Q}_3 \bar{Q}_2 \bar{Q}_1 + Q_3 Q_1 + Q_3 Q_2$ ,  $f_2 = Q_3 \bar{Q}_1 + \bar{Q}_2 \bar{Q}_1 + \bar{Q}_3 Q_2 Q_1$  with a 3-bit ripple counter.
- (d)  $f = \bar{Q}_1 + \bar{Q}_3 + \bar{Q}_4 \bar{Q}_2$ , with a 4-bit shift register.
- (e)  $f = \bar{Q}_4 + \bar{Q}_2 \bar{Q}_1 + \bar{Q}_3 Q_2$  with a 4-bit shift register.
- 12.5**  $D_1 = Q_3 + Q_2 Q_1$ ,  $D_2 = \bar{Q}_2 + \bar{Q}_1$ ,  $D_3 = Q_3 Q_2 + \bar{Q}_1$
- 12.6**  $f = \bar{Q}_2 \bar{Q}_1 + Q_2 Q_1$  with a 3-bit ripple counter.
- 12.7**  $S_4 = Q_3 Q_2 Q_1 M + \bar{Q}_4 \bar{Q}_3 \bar{Q}_2 \bar{Q}_1 \bar{M}$        $S_3 = \bar{Q}_3 Q_2 Q_1 M + Q_4 \bar{Q}_1 \bar{M}$   
 $S_2 = \bar{Q}_4 \bar{Q}_2 Q_1 M + (Q_4 \bar{Q}_1 + Q_3 \bar{Q}_2 \bar{Q}_1) \bar{M}$        $S_1 = \bar{Q}_1$   
 $R_4 = Q_4 \bar{Q}_1 \bar{M} + Q_4 Q_1 M$        $R_3 = Q_3 Q_2 Q_1 M + Q_3 \bar{Q}_2 \bar{Q}_1 \bar{M}$   
 $R_2 = Q_2 Q_1 M + Q_2 \bar{Q}_1 \bar{M}$        $R_1 = Q_1$
- 12.8**  $D_1 = \bar{Q}_3 \bar{Q}_2 M + \bar{Q}_3 \bar{Q}_2 \bar{Q}_1 \bar{M}$        $D_2 = \bar{Q}_3 M + Q_3 \bar{M} + Q_2 \bar{Q}_1 \bar{M}$   
 $D_3 = Q_2 \bar{Q}_1 M + \bar{Q}_2 \bar{M}$

## CHAPTER 13

### Answers to Fill in the Blanks

- |  |                |                     |
|--|----------------|---------------------|
| 1. node  | 2. final state | 3. state variables  |
| 4. forward/backward, bidirectional                           |                |                     |
| 5. high speed, less severe decoding problems, more circuitry |                |                     |
| 6. loading   | 7. transition  | 8. transition       |
|  |                | 9. state assignment |

### Answers to Objective Type Questions

- |      |      |       |      |      |      |      |
|------|------|-------|------|------|------|------|
| 1. d | 2. a | 3. c  | 4. b | 5. d | 6. b | 7. a |
| 8. d | 9. a | 10. c |      |      |      |      |

### Answers to Problems

**13.1**  $J_1 = y_2x$ ,  $K_1 = y_2$ ,  $J_2 = y_1\bar{x} + \bar{y}_1x$ ,  $K_2 = \bar{y}_1 + \bar{x}$ ,  $z = y_1y_2x$

**13.2**  $D_3 = Q_2$ ,  $D_2 = \bar{Q}_3\bar{Q}_2 + Q_3Q_1$ ,  $D_1 = \bar{Q}_2Q_1 + Q_2\bar{Q}_1$

**13.3**  $S_1 = \bar{y}_2x$ ,  $R_1 = y_2$ ,  $S_2 = y_1x$ ,  $R_2 = \bar{y}_1$ ,  $z_1 = y_1$ ,  $z_2 = y_2$

**13.4**  $T_1 = y_2\bar{x} + y_2y_3 + y_1x + y_1y_3$ ,  $T_2 = y_1y_2 + \bar{y}_1y_3x + \bar{y}_2y_3\bar{x} + y_1\bar{x} + \bar{y}_3y_2\bar{x}$   
 $T_3 = \bar{y}_1\bar{y}_2x + y_2y_3\bar{x} + y_1y_3x$ ,  $z = y_1x$

**13.5**  $S_1 = y_2y_3x$ ,  $R_1 = y_1$ ,  $S_2 = y_1\bar{x} + \bar{y}_2y_3\bar{x}$ ,  $R_2 = y_2y_3 + x$ ,  $S_3 = y_2\bar{y}_3 + \bar{y}_3x$ ,  
 $R_3 = y_2y_3 + y_3\bar{x}$ ,

**13.6**  $D_1 = y_1\bar{y}_2x + \bar{y}_1y_2x$ ,  $D_2 = y_1\bar{y}_2 + \bar{x}$

**13.7**  $J_1 = K_1 = 1$ ,  $J_2 = K_2 = y_1M + \bar{y}_1\bar{M}$ ,  $J_3 = K_3 = y_1y_2M + \bar{y}_1\bar{y}_2\bar{M}$   
 $M = \text{clock ANDed with control } x \text{ and applied to J-K FFs.}$

**13.8**  $D_1 = \bar{y}_1\bar{y}_2\bar{y}_3\bar{x} + \bar{y}_3y_4\bar{x} + \bar{y}_2y_4\bar{x} + \bar{y}_4y_3x + \bar{y}_4y_2x$   
 $D_2 = \bar{y}_2y_3x + y_2\bar{y}_3\bar{x} + \bar{y}_3y_4\bar{x} + \bar{y}_4y_3x + \bar{y}_2\bar{y}_4y_1x$   
 $D_3 = y_2\bar{y}_3\bar{x} + \bar{y}_4y_2x + \bar{y}_2y_3\bar{x} + \bar{y}_2y_4x$ ;  $D_4 = y_2y_3\bar{y}_4x + y_2\bar{y}_3y_4x$

**13.9**  $J_1 = y_2y_3$ ,  $K_1 = 1$ ,  $J_2 = \bar{y}_1y_3\bar{x}$ ,  $K_2 = x + y_3$ ,  $J_3 = \bar{y}_1\bar{x}$ ,  $K_3 = \bar{x} + \bar{y}_2$ ,  $z = y_1y_3\bar{x} + y_1\bar{y}_3x$

## CHAPTER 14

### Answers to Fill in the Blanks

- |                               |                |                                   |
|-------------------------------|----------------|-----------------------------------|
| 1. Moore                      | 2. Mealy       | 3. sequential circuit             |
| 5. P-distinguishable          | 6. Unspecified | 4. terminal                       |
| 8. merger graph, merger table |                | 7. Paull–Unger, implication chart |
|                               |                | 9. one state                      |
|                               |                | 10. minimal cover table.          |

**Answers to Problems****14.1** (a)  $P_2 = (A)(C, D)(B)(E, F)$ (b)  $P_4 = (C)(H)(A)(B)(E)(D)(G)(F)$  and no minimization is possible.(c)  $P_3 = (A)(B, D)(C)(E)(F)(G)(H)$ (d)  $P_4 = (A, B)(C)(D)(E)(F, G)$ 

(a)

PS	NS, Z	
	X = 0	X = 1
A	C, 0	A, 1
B	E, 1	C, 1
C	C, 0	E, 1
E	C, 1	C, 1

(c)

PS	NS, Z	
	X = 0	X = 1
A	E, 0	B, 1
B	F, 0	B, 1
C	E, 0	B, 0
E	C, 0	F, 0
F	B, 0	C, 0
G	B, 1	C, 1
H	B, 1	A, 1

(d)

PS	NS, Z	
	X = 0	X = 1
A	F, 0	A, 1
C	D, 0	C, 1
D	C, 0	A, 1
E	D, 0	A, 1
F	E, 1	F, 1

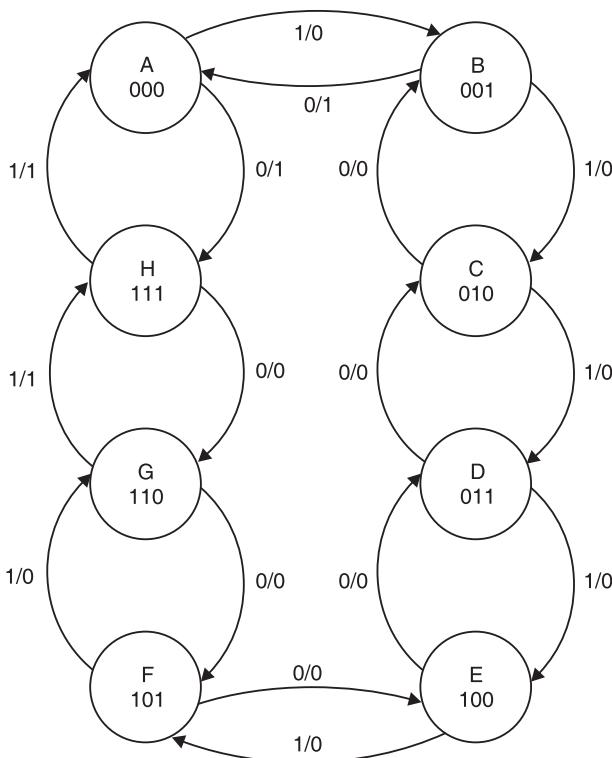
**14.2** (a) (C, D)(B, F)(A, F)

(b) (A, D)(B, F)(C, D)(E, F)

(c) (A, B, D, F)(A, C, F)(A, E, F)

(d) (B, D, E) (A, B, C) (C, G) (A, F) (A, B, H)

**CHAPTER 15****Answers to Fill in the Blanks****1.** state box**2.** state box**3.** Mealy**4.** combinational**5.** data, control**6.** data**7.** Moore**8.** sequential circuit**9.** Mealy**10.** link path

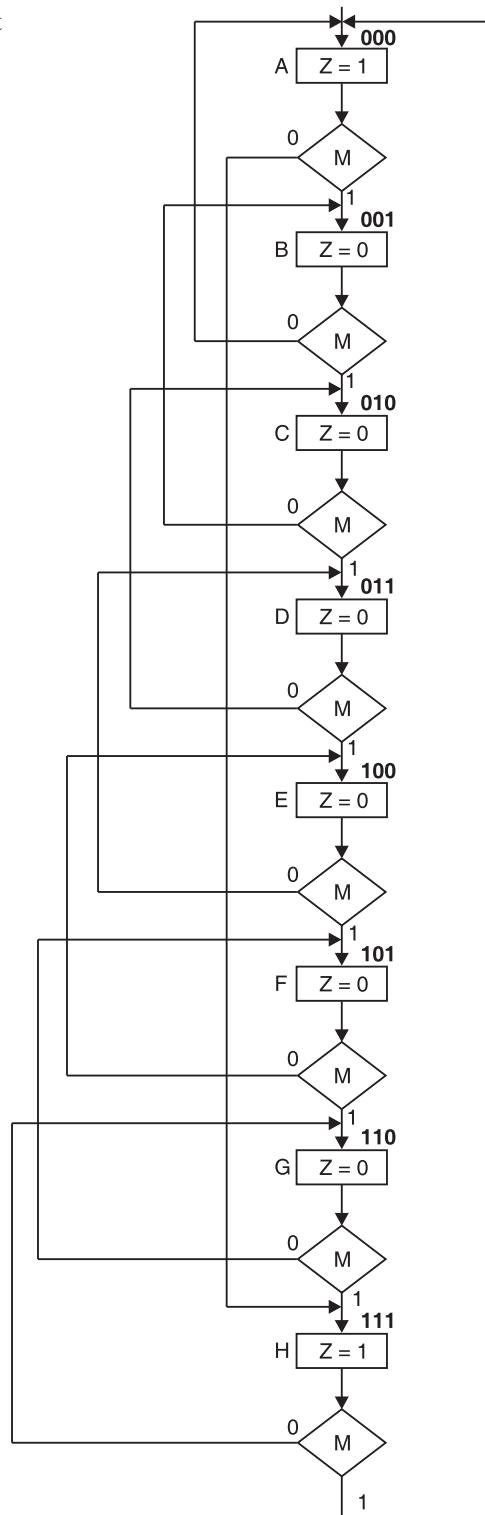
**Answers to Objective Type Questions**1. d  
8. d2. a  
9. a3. d  
10. c4. b  
11. a5. d  
12. c6. d  
13. c7. a  
14. d**Answers to Problems****15.1 (a)**

(b) State table

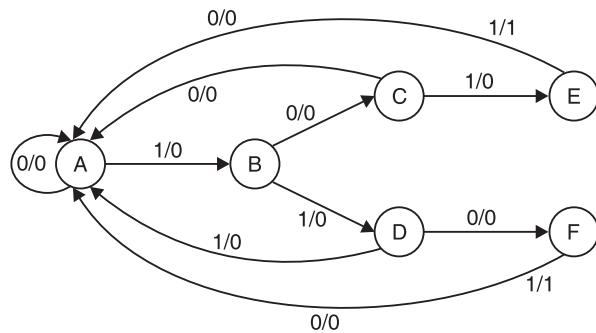
PS	NS, O/P	
	INPUT M	
	M = 0	M = 1
A	H,1	B,0
B	A,1	C,0
C	B,0	D,0
D	C,0	E,0
E	D,0	F,0
F	E,0	G,0
G	F,0	H,1
H	G,0	A,1

**1050 ANSWERS**

(c) ASM Chart



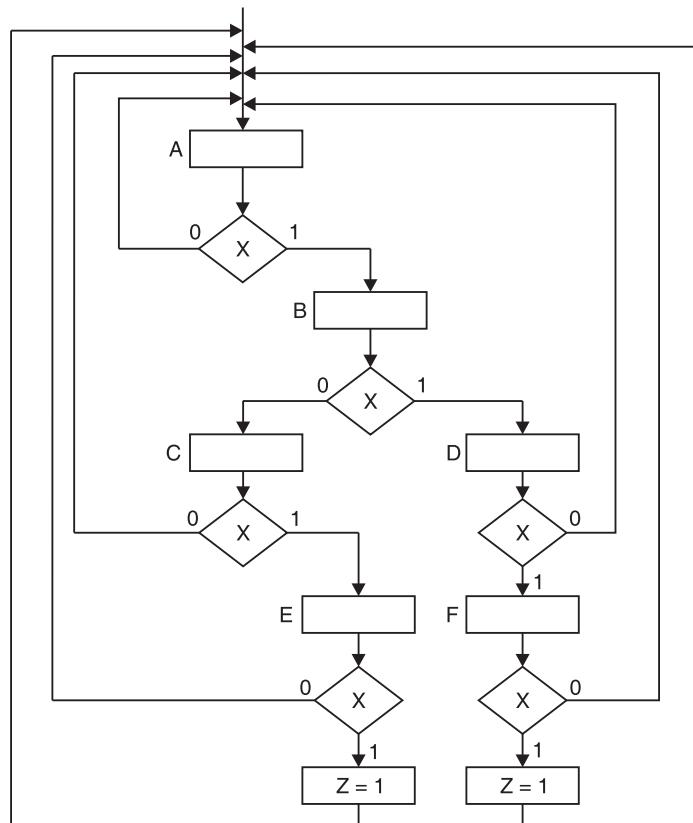
## 15.2 (a) State diagram



## (b) State table

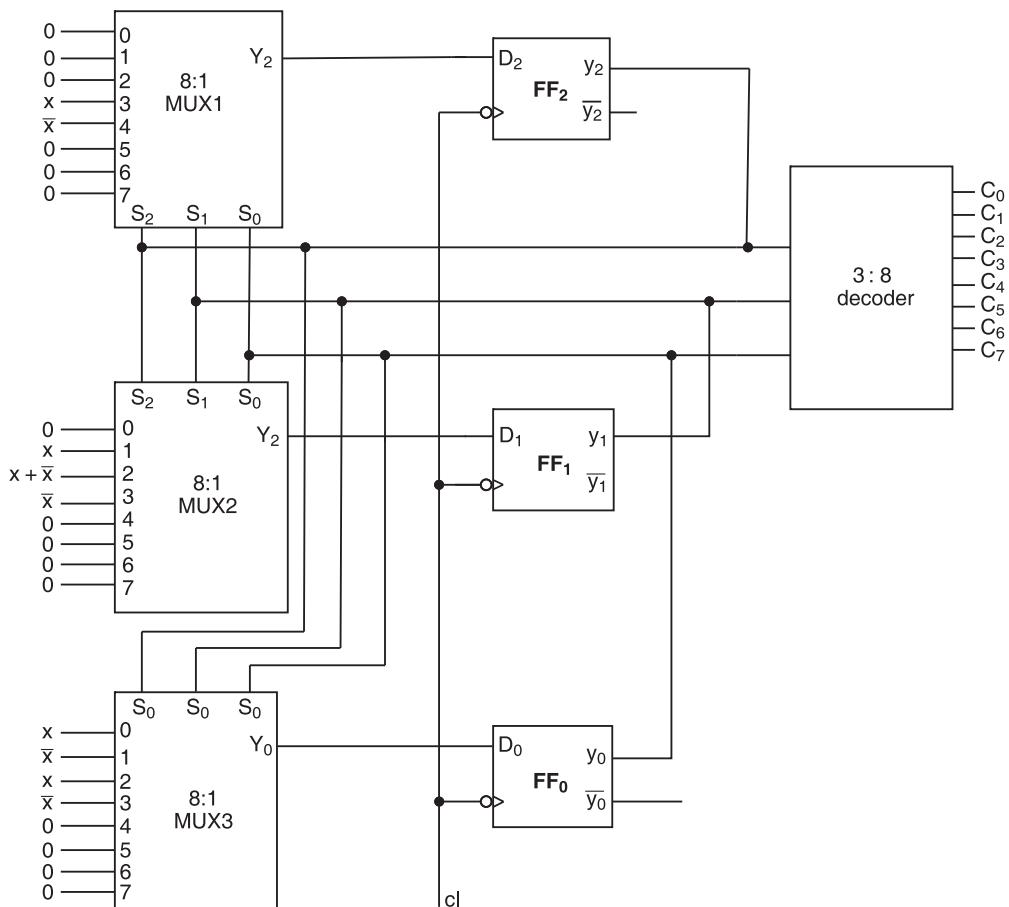
PS	NS, O/P	
	INPUT X	
	X = 0	X = 1
A	A, 0	B, 0
B	C, 0	D, 0
C	A, 0	E, 0
D	A, 0	F, 0
E	A, 0	A, 1
F	A, 0	A, 1

## (c) ASM Chart

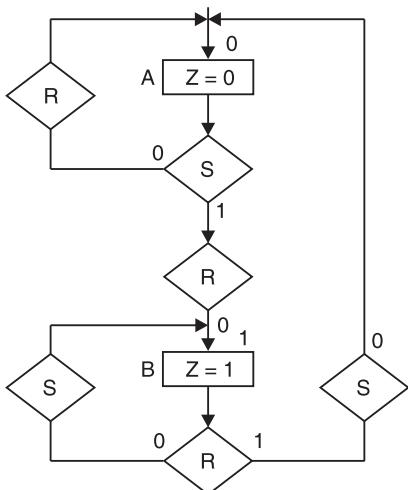


**1052** ANSWERS

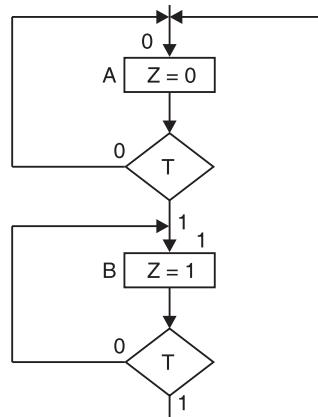
### 15.3 (c) Logic circuit using multiplexer

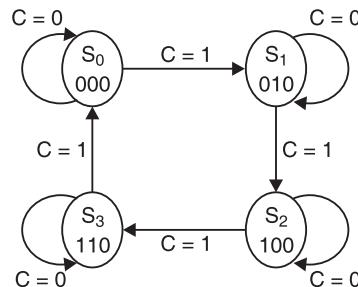


### 15.4 (a)



(b)

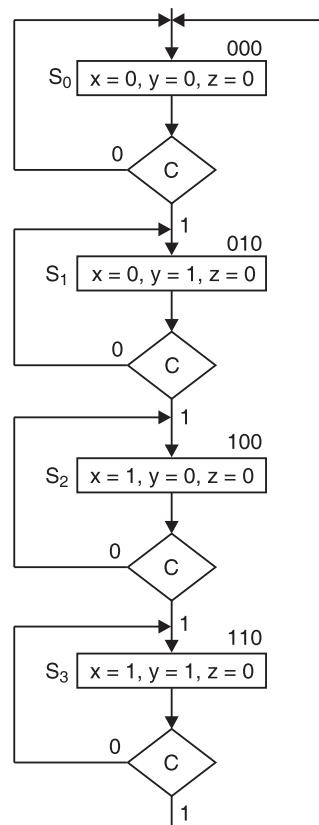


**15.5** (a) State diagram

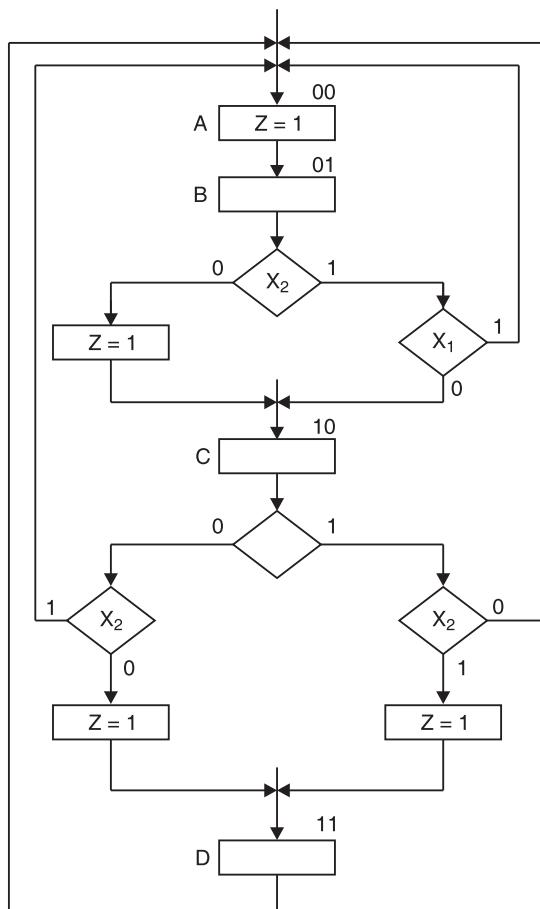
## (b) State table

PS	NS, O/P	
	INPUT C	
	C = 0	C = 1
S <sub>0</sub>	S <sub>0</sub> , 000	S <sub>1</sub> , 010
S <sub>1</sub>	S <sub>1</sub> , 010	S <sub>2</sub> , 100
S <sub>2</sub>	S <sub>2</sub> , 100	S <sub>3</sub> , 110
S <sub>3</sub>	S <sub>3</sub> , 100	S <sub>0</sub> , 000

## (c) ASM Chart



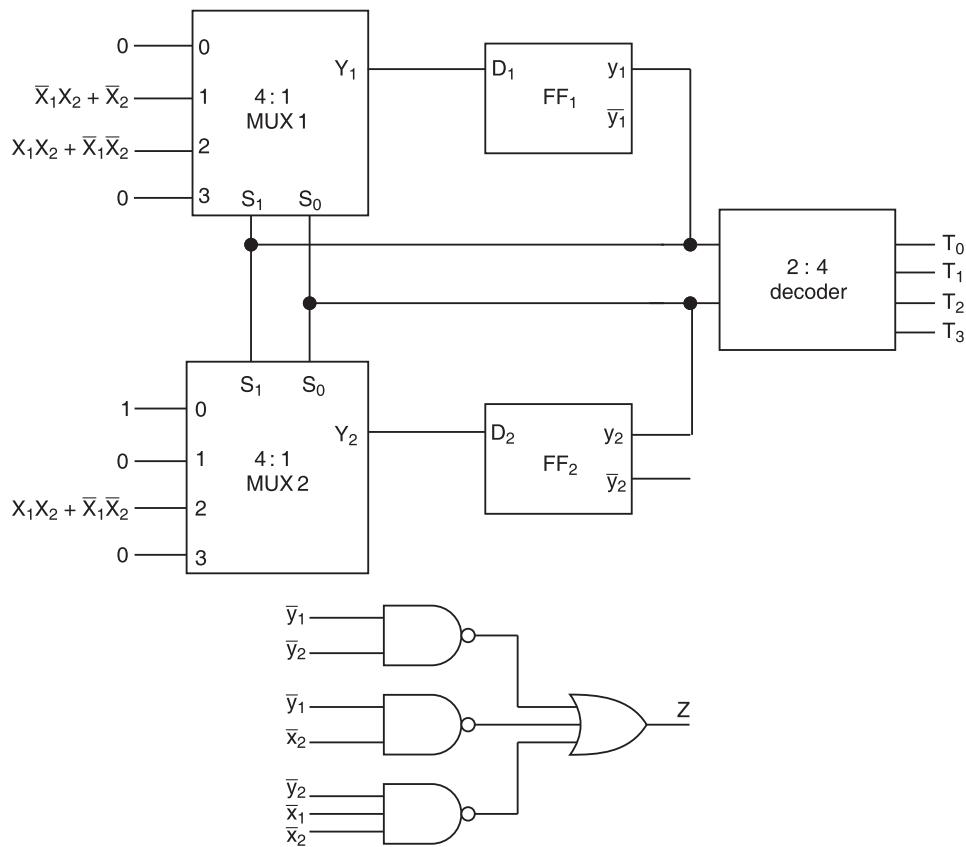
**15.6 (a)**



(b)

PS	NS, O/P			
	INPUT $X_1X_2$			
	00	01	10	11
A	B,1	B,1	B,1	B,1
B	C,1	C,0	C,1	A,0
C	D,1	A,0	A,0	X
D	A,0	X	X	X

(c) Logic diagram using multiplexers



## CHAPTER 16

### Answers to Fill in the Blanks

- |                                      |  |                   |                        |
|--------------------------------------|--|-------------------|------------------------|
| 1. TTL, ECL, IIL                     | 2. MOS, CMOS   | 3. threshold      | 4. propagation delay   |
| 5. power dissipation                 | 6. fan-in  | 7. fan-out        | 8. noise margin        |
| 9. speed power product               |  | 10. standard load | 11. TTL                |
| 12. TTL                              | 13. totem-pole type, open-collector type, tri-state type |                   |                        |
| 14. low, high, high impedance        |  | 15. F(fast) TTL   |                        |
| 16. ECL                              | 17. ECL  | 18. ECL           | 19. NMOS               |
| 20. depletion type, enhancement type |  | 21. enhancement   |                        |
| 22. LSI, VLSI, ULSI                  | 23. CMOS   | 24. CMOS          | 25. CMOS               |
| 26. ECL, CMOS                        | 27. CMOS, IIL  | 28. CMOS, ECL     | 29. TTL open collector |
| 30. ECL open emitter                 | 31. high speed, low power dissipation                    |                   | 32. TTL, CMOS          |

## 1056 ANSWERS

- |                             |                                |                         |                       |
|-----------------------------|--------------------------------|-------------------------|-----------------------|
| 33. MOS                     | 34. MOS, IIL                   | 35. transmission        | 36. MOS               |
| 37. MOS                     | 38. refreshing                 | 39. CMOS                | 40. level translators |
| 41. the speed power product |                                | 42. RTL, DCTL, DTL, HTL |                       |
| 43. eight                   | 44. totem pole, open collector |                         | 45. large, small      |
| 46. dynamic MOS             | 47. level translators          | 48. ECL                 | 49. 74C               |
| 50. IIL                     | 51. CMOS                       | 52. speed.              |                       |

### Answers to Objective Type Questions

- |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|
| 1. a  | 2. a  | 3. b  | 4. c  | 5. a  | 6. b  | 7. b  |
| 8. b  | 9. d  | 10. c | 11. a | 12. d | 13. d | 14. a |
| 15. b | 16. b | 17. d | 18. b | 19. b | 20. c | 21. c |
| 22. c |       |       |       |       |       |       |

## CHAPTER 17

### Answers to Fill in the Blanks

- |                              |   |  |
|------------------------------|---|--|
| 1. A/D, D/A                  | 2. ADCs, DACs                               | 3. true analog, pseudo analog          |
| 4. resolution                | 5. offset voltage                           | 6. monotonic      7. step size         |
| 8. full scale error          | 9. $R-2R$ ladder type                       | 10. switched current, switched voltage |
| 11. switched voltage         | 12. quantization                            | 13. counter, flash      14. flash      |
| 15. flash                    | 16. dual-slope                              | 17. successive approximation           |
| 18. voltage, current         | 19. voltage to frequency, flash, dual slope |  |
| 20. successive approximation | 21. dual-slope                              | 22. digital ramp.                      |

### Answers to Objective Type Questions

- |      |      |       |      |      |      |      |
|------|------|-------|------|------|------|------|
| 1. a | 2. a | 3. b  | 4. b | 5. d | 6. c | 7. c |
| 8. c | 9. d | 10. a |      |      |      |      |

### Answers to Problems

- 17.1** 6 bits  
**17.2** 3.1 V; 2.1 V  
**17.3** (a) 55/32 V (b) 215/64 V  
**17.4** 11001100  
**17.5**  $R_f = 10 \text{ k}\Omega$ ,  $R = 7.8125 \text{ k}\Omega$ ,  $2R = 15.625 \text{ k}\Omega$ ,  $4R = 31.25 \text{ k}\Omega$ ,  $8R = 62.4 \text{ k}\Omega$ ,  $V_0 = -8.8 \text{ V}$   
**17.6** 10 mA  
**17.7**  $I_3 = 0.75 \text{ mA}$ ,  $I_2 = 0.375 \text{ mA}$ ,  $I_1 = 0.1875 \text{ mA}$ ,  $I_0 = 0.09375 \text{ mA}$   
**17.8** 1.8125 mA, 9.0625 V  
**17.9** 6.875 V  
**17.10** 0.625  
**17.11**  $f_{\max} = 5 \text{ MHz}$

**17.12** 31 comparators and 32 resistors; increment between voltages = 0.625 V

**17.13** 40.95 V

**17.14** 10  $\mu$ s

**17.15** 10110000

## CHAPTER 18

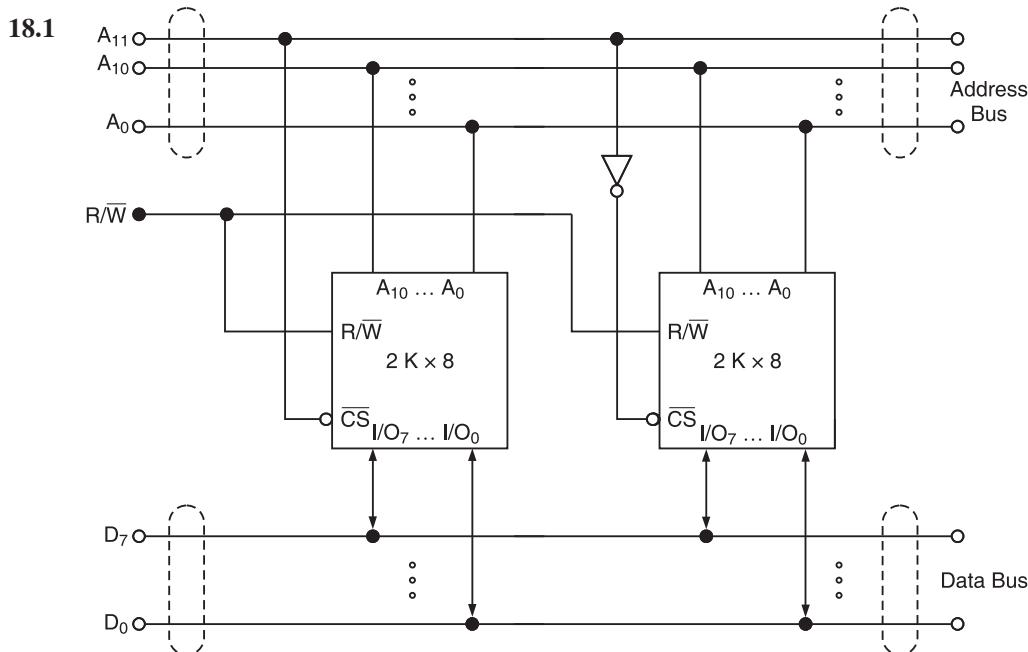
### Answers to Fill in the Blanks

- |   |                   |                       |
|---|-------------------|-----------------------|
| 1. stored program type                                  | 2. an address     | 3. word               |
| 4. cell   | 5. semiconductor  | 6. auxiliary          |
| 8. reading  | 9. software       | 10. firmware          |
| 12. access time   | 13. D, S          | 14. ECL RAMs          |
| 16. serial  | 17. dynamic       | 18. faster            |
| 20. CMOS  | 21. bit-organized | 22. word-organized    |
| 23. mass storage and backup                             |                   |                       |
| 24. speed, reduced complexity, space, power consumption |                   | 25. memory expansion. |

### Answers to Objective Type Questions

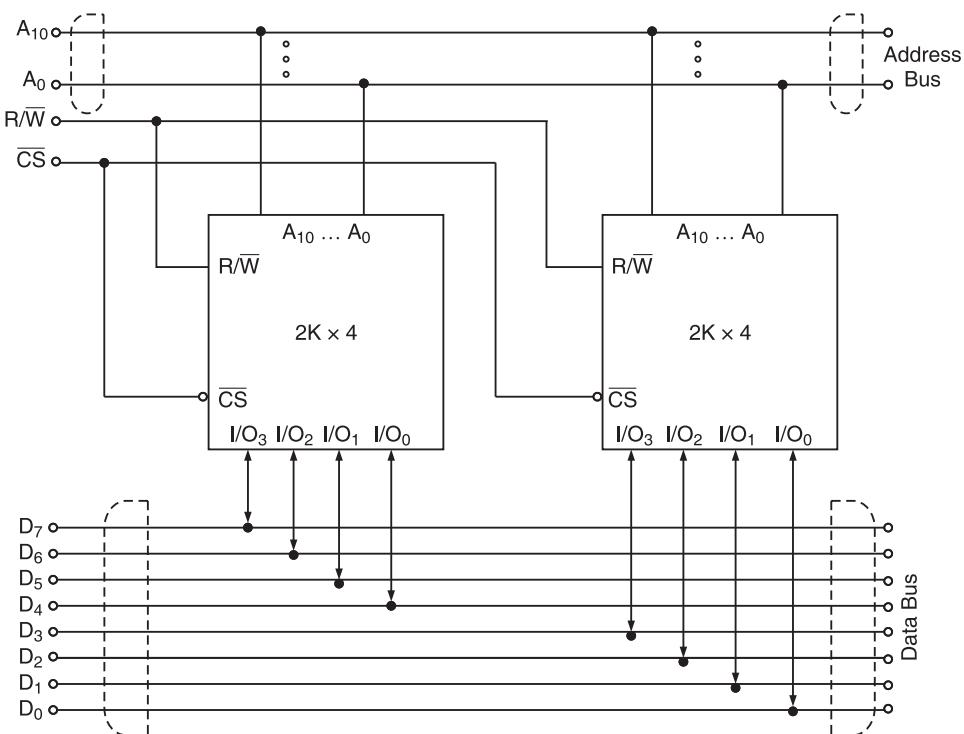
1. c      2. b      3. a      4. b      5. b      6. b      7. b

### Answers to Problems

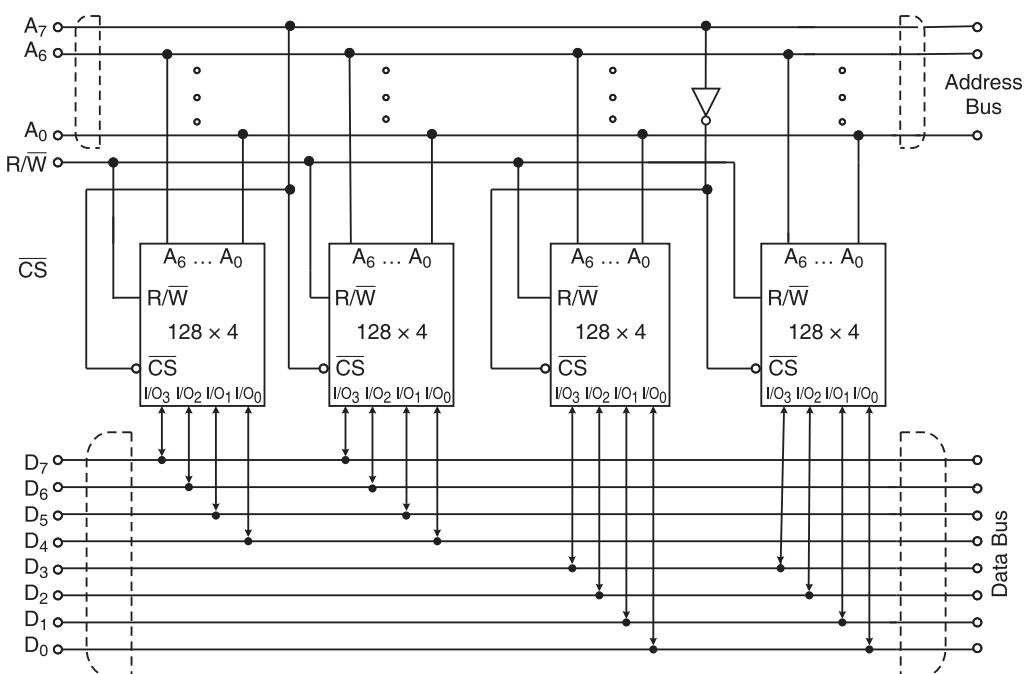


**1058 ANSWERS**

**18.2**



**18.3**



## CHAPTER 19

### Answers to Fill in the Blanks

- |   |                      |                                     |             |
|---|----------------------|-------------------------------------|-------------|
| 1. Schmitt trigger                      | 2. Hysteresis loop   | 3. Stable                           | 4. Stable   |
| 5. Stable, quasi-stable                 | 6. Quasi-stable      | 7. Retriggerable, non-retriggerable |             |
| 8. Retriggerable                        | 9. Non-retriggerable | 10. Monostable                      | 11. Astable |
| 12. Crystal controlled                  |                      | 13. Numbers, letters, symbols       |             |
| 14. Infra red                           | 15. High power       | 16. Low power consumption, low cost |             |
| 17. Less                                | 18. LCDs             | 19. LCDs                            | 20. CMOS    |
| 21. Incandescent seven segment displays |                      |                                     |             |

### Answers to Problems

- 19.1** Let  $C = 1 \mu\text{F}$ , then  $R = 900 \Omega$ .
- 19.2** Let  $R_{\text{ext}} = 10 \text{ k}\Omega$ , then  $C_{\text{ext}} = 714 \text{ pF}$ .
- 19.3** Let  $C = 0.05 \mu\text{F}$ , then  $R_A = 17.31 \text{ k}\Omega$ ,  $R_B = 11.54 \text{ k}\Omega$ .
- 19.4** Let  $C = 0.1 \mu\text{F}$ , then  $R_A = 1.47 \text{ k}\Omega$ .
- 19.5** 592  $\text{k}\Omega$  through 6087  $\text{k}\Omega$ .



# INDEX

- Active-HIGH gates, 211
- Active-LOW gates, 211
- Active pull-up, 868
- Adder, 5
  - BCD, 343
  - full, 329
  - half, 327
  - IC parallel, 330
  - look-ahead carry, 338
  - parallel binary, 336
  - ripple carry, 337
  - serial, 342
  - serial binary, 709
- Adder/subtractor, 342
- Addition, 5
- Adjacency, 235
- Adjacency ordering, 245
- All or nothing gate, 230
- Analog
  - circuit, 1
  - device, 1
  - ICs, 10
- Analog-to-digital converters (ADCs), 910
  - counter type, 926
  - digital ramp type, 926
  - dual-slope type, 933
  - flash type, 930
  - parallel type, 930
  - simultaneous type, 930
  - successive-approximation type, 931
  - tracking type, 928
  - voltage-to-frequency type, 936
- Anti-coincidence gate, 142
- Any or all gate, 135
- AOI logic, 206
- AOI logic to NAND/NOR logic, 206
- Applications of flip-flops, 584
  - counting, 585
  - frequency division, 585
  - parallel data storage, 584
  - parallel-to-serial conversion, 584
  - serial data storage, 584
  - serial-to-parallel conversion, 584
  - transfer of data, 584

## 1062 INDEX

- Application specific integrated circuit, 460
- Arbitration, 379
- Arithmetic circuits, 327
- ASM, 799
  - block, 802
  - chart, 800
- Asserted levels, 211
- Astable multivibrator, 988
  - using 555 timer, 989
  - using inverters, 990
  - using op-amps, 992
  - using schmitt trigger, 989
- Asynchronous
  - inputs, 563
  - latches, 553
  - systems, 555
- Axioms, 178
- Backplane, 999
- Base, 28
- Basic building blocks, 212
- Basic gates, 132
- Basic operations, 177
- BCD addition, 89
- BCD subtraction, 90
- Binary adder/subtractor, 342
- Binary codes, 86
  - 8421 BCD, 87
  - 5 bit, 103
  - alphanumeric, 110
  - ASCII, 110
  - BCD, 86
  - biquinary, 104
  - cyclic, 88
  - EBCDIC, 111
  - error correcting, 105
  - error detecting, 100
  - Gray, 96
  - Hamming, 105
  - Johnson, 104
  - natural, 88
  - negatively-weighted, 87
  - non-weighted, 86
  - numeric, 86
  - positively-weighted, 87
- reflective, 88
- ring counter, 104
- self-complementing, 108
- sequential, 108
- unit distance, 108
- weighted, 86
- XS-3, 92
- XS-3 Gray, 99
- Binary multipliers, 347
- Binary number system, 31
  - addition, 37
  - division, 39
  - multiplication, 38
  - subtraction, 37
- Bipolar ICs, 11
- Bistable multivibrator, 549
- Bit, 31
- Boolean algebra, 176
- Boolean algebraic laws, 178
  - absorption, 183
  - AND, 178
  - associative, 179
  - commutative, 179
  - complementation, 179
  - consensus, 183
  - De Morgan's, 185
  - distributive, 180
  - double complementation, 178
  - idempotence, 182
  - identity, 178
  - included factor, 183
  - null, 178
  - OR, 179
  - redundant literal rule, 182
  - Shannon's expansion, 186
  - transposition, 184
- Boolean expression to logic, 203
- Booting up, 949
- Bootstrap program, 949
- Branching method, 290
- Bubbled AND gate, 141
- Bubbled NAND gate, 139
- Bubbled NOR gate, 141
- Bubbled OR gate, 139
- Buffer/driver, 875
- Buffer register, 606
- Buses, 11

- Canonical POS form, 195, 232  
 Canonical SOP form, 194, 232  
 Carry flag, 40  
 Carry generate function, 338  
 Carry-in, 329  
 Carry-out, 329  
 Carry propagate function, 338  
 Cascading of  
     BCD adders, 344  
     parallel adders, 341  
     ripple counters, 640  
     synchronous counters, 664  
 Cathode ray tube, 976  
 Cell, 234, 943  
 Characteristic vector, 196  
 Check sums, 101  
 Chip, 10  
 Chip select, 955  
 Circuit design, 3  
 Classification of displays, 996  
 Classification of codes, 86  
 Clipping circuits, 993  
 Clock skew, 568  
 Clock transition times, 568  
 Clocked flip-flops, 553  
 CMOS  
     bilateral switch, 893  
     buffered and unbuffered gates, 893  
     inverter, 890  
     NAND gate, 891  
     NOR gate, 892  
     open-drain outputs, 894  
     transmission gate, 893  
 Code converters, 350  
     BCD-to-XS-3, 355  
     BCD-to-Gray, 356  
     binary-to-BCD, 354  
     binary-to-Gray, 351  
     Gray-to-binary, 352  
 Code converters using ICs, 366  
 Code word, 86  
 Coincidence gate, 143  
 Combinational circuits, 326  
 Combinational PLDs, 468  
 Common-anode type LED display, 388  
 Common-cathode type LED display, 388  
 Comparators, 371  
 Compatibility graph, 776  
 Compatible states, 765  
 Complement, 178  
 Complement arithmetic  
     1's complement, 48  
     2's complement, 45  
     9's complement, 30  
     10's complement, 30  
 Complement form, 41  
 Computation of total gate inputs, 202  
 Computer method of division, 40  
 Computer method of multiplication, 39  
 Conditional output box, 801  
 Conjunctive canonical form (CCF), 195  
 Conjunctive normal form (CNF), 194  
 Control path subsystem, 800  
 Control unit, 13  
 Controlled buffer register, 607  
 Controlled inverter, 143, 144  
 Conversion between canonical forms, 199  
 Conversion from  
     binary to decimal, 32  
     binary to Gray, 97  
     binary to hexadecimal, 60  
     binary to octal, 57  
     decimal to binary, 33  
     decimal to hexadecimal, 62  
     decimal to octal, 57  
     Gray to binary, 98  
     hexadecimal to binary, 61  
     hexadecimal to decimal, 61  
     hexadecimal to octal, 63  
     octal to binary, 56  
     octal to decimal, 57  
     octal to hexadecimal, 63  
 Conversion of flip-flops, 579  
     D to J-K, 583  
     D to S-R, 580  
     J-K to D, 583  
     J-K to S-R, 580  
     J-K to T, 582  
     S-R to D, 580  
     S-R to J-K, 579  
 Conversion time of ADC, 927  
 Counters  
     asynchronous, 633

- basic ring, 665
- BCD, 654
- decade, 637, 654
- design of, 636, 642
- divide-by-N, 632
- down, 632
- full modulus, 633
- Johnson, 667
- modulus, 633
- ring, 666
- ripple, 631
- self-starting, 643, 657
- serial, 632
- series, 632
- shift register, 665
- shortened modulus, 632
- switch tail ring, 665
- synchronous, 642
- twisted ring, 667
- up, 632
- Cross point, 463
- Crystal-controlled clock generator, 993
- Current injection logic (CIL), 878
- Current injector transistor, 879
- Current mode logic (CML), 880
- Current steering logic (CSL), 880
- Cut set hazard , 408
  
- Data lock-out FF, 576
- Data selector, 390
- Datapath, 799
- Datapath subsystem, 800
- Decimal number system, 28
- Decision box, 801
- Decoders, 382
  - 1-of-8, 384
  - 3-line to 8-line, 384
  - applications, 387
  - BCD-to-decimal, 384
  - BCD-to-seven segment, 388
- Decoding, 7
- Demorganization, 186
- Demultiplexer tree, 401
- Demultiplexers
  - 1-line to 4-line, 399
  - 1-line to 8-line, 399
- Demultiplexing, 8
- Derived operations, 177
- Digit, 22
- Digital
  - circuits, 1
  - computer, 12
  - ICs, 10
  - quantity, 910
  - systems, 2
- Digital-to-analog (D/A) conversion, 911
- Digital-to-analog converters (DACs), 910
  - bipolar, 915
  - parameters of
    - accuracy, 912
    - monotonicity, 913
    - offset voltage, 913
    - resolution, 912
    - settling time, 913
  - R-2R* ladder type, 915
  - switched-capacitor type, 924
  - switched current-source type, 923
  - using BCD input code, 914
  - weighted-resistor type, 920
- Diode matrix, 378
- Direct logic, 669
- Discrete
  - AND gate, 134
  - NAND gate, 139
  - NOR gate, 141
  - NOT gate, 137
  - OR gate, 135
- Disjunctive canonical form (DCF), 194
- Disjunctive normal form (DNF), 194
- Display devices, 996
- Distinguishable states, 752
- Distinguishing sequences, 752
- Divider, 6
- Dominating columns, 290
- Dominating rows, 290
- Don't care
  - combinations, 260
  - conditions, 273
  - terms, 289
- Double dabble method, 34
- Double precision, 55
- Dual-in-line package, 10

- Duality, 188
- Duals, 189
- Duty cycle, 5
- Dynamic
  - MOS inverter, 897
  - MOS logic, 896
  - MOS registers, 614
  - NAND gate, 898
  - NOR gate, 898
  - RAMs, 952
    - address multiplexing, 953
    - combining, 959
    - refreshing, 953
    - shift register, 620
  - Dynamic hazard, 406
  - Dynamic memory controllers, 952
  - Dynamic RAM controllers, 953
  - Dynamic triggering, 555
- ECL RAMs, 951
- Edge-triggered flip-flops, 555
  - D, 558
  - J-K, 559
  - S-R, 556
  - T, 561
- Edge triggering, 555
- EEPROM, 468
- Encoders, 375
  - decimal-to-BCD, 377
  - keyboard, 378
  - octal-to-binary, 376
  - priority, 379
    - decimal-to-BCD, 380
    - octal-to-binary, 382
- Encoding, 6, 375
- End around carry, 30, 48
- EPLD, 478
- EPROM, 452
- Equality detector, 143
- Equivalence partition, 754
- Essential false prime implicants, 251
- Essential hazards, 409
- Essential prime implicants, 249
- Essential row, 290
- Even parity checker, 367
- Excess-3 addition, 92
- Excess-3 subtraction, 93
- Excess-3 subtractor, 346
- Excess-n notation, 56
- Excitation table, 562, 645
- Excitation variables, 548
- Excitations, 550
- Expandable gates, 210
- Expanded POS form, 195
- Expanded SOP form, 194
- Exponent, 55
- Falling edge, 4
- Fall time, 4
- False prime implicants, 251
- Fan-in, 863
- Fan-out, 863
  - high state, 869
  - low state, 869
- Fast (F) TTL, 877
- Field programmable logic array, 461
- Fifteen-bit hamming code, 108
- Figure of merit, 866
- Finite state machine, 745
- Finite state model, 699
- Firmware, 948
- Flat package, 10
- Flip-flops, 546
  - non-clocked, 550
  - operating characteristics, 566
    - clock transition times, 568
    - hold time, 567
    - maximum clock frequency, 567
    - power dissipation, 568
    - propagation delay time, 566
    - set-up time, 567
- Floating point numbers, 55
- Floppy disk, 967
- Flying head, 968
- Four-bit comparator, 373
- FPLA, 461
- Frequency stability, 993
- Full modulus counter, 633
- Full-subtractor, 334
- Fundamental building blocks, 132

- Fundamental mode circuits, 549  
Functionally complete set, 193  
Fuse map, 468
- Gated latches, 553  
D, 554  
S-R, 553  
Gating, 988  
Generation of narrow spikes, 555  
Giant scale integration, 861  
Glitch, 406
- Half-subtractor, 332  
Hard disk, 968  
Hazards, 406  
Hazard-free realization, 408  
Hex dabble method, 62  
Hexadecimal number system, 59  
Hybrid counters, 663  
Hybrid logic, 266  
Hybrid systems, 3  
Hysteresis loop, 982
- IC comparator, 374  
Illegal code, 89  
Implication chart method, 768  
Inclusive OR gate, 135  
Incompletely specified functions, 278  
Incompletely specified machines, 764  
Index of the term, 288  
Indirect logic, 672  
Inequality detector, 142  
Inhibit circuits, 145  
Instruction code, 56  
Interfacing,  
    CMOS to TTL, 901  
    ECL to TTL, 900  
    TTL to CMOS, 900  
    TTL to ECL 900  
Inverter, 136
- J-K flip-flop with active-LOW PRESET and  
CLEAR, 574
- Karnaugh map, 196  
    five-variable, 253  
    four-variable, 245  
    six-variable, 257  
    three-variable, 238  
    two-variable, 233
- Latch, 550  
    active-HIGH input, 550  
    active-LOW input, 550
- LCD displays, 998  
    dynamic scattering type, 998  
    field effect type, 998  
    operation of, 998
- Leading edge, 4  
Least significant digit, 28  
LEDs, 997  
Level  
    logic, 132  
    shifters, 883  
    translator, 883
- Levels of integration, 10  
    LSI, 11  
    MSI, 11  
    SSI, 11  
    ULSI, 11  
    VLSI, 11
- Level triggered flip-flops, 553  
Light emitting segments, 388  
Limitations of K-maps, 280  
Linear sequence generator, 678  
Link path, 802  
Liquid crystal cells, 998  
    reflective type, 998  
    transmittive type, 998
- Literals, 289  
Loading, 605, 642  
Loading factor, 863  
Lock-out, 633  
    elimination of, 658
- Logic, 5  
    circuits, 2  
    design, 3, 132  
    diagram, 203  
    levels, 3  
    operations, 177  
    race, 267

- Logic families, 861
  - CMOS (*see* CMOS), 889
  - ECL, 880
    - interfacing, 883
    - subfamilies, 882
    - wired OR operation, 882
  - I2L, 878
    - inverter, 878
    - NAND gate, 879
    - NOR gate, 880
  - MOS, 885
  - TTL, 866
    - characteristics, 867
  - current sinking, 869
    - current sourcing, 869
    - HIGH-state fan-out, 869
    - loading, 869
    - LOW-state fan-out, 869
    - open collector output, 869
    - passive pull-up, 873
    - subfamilies, 875
    - totem-pole output, 868
    - tri-state, 874
    - unit load, 870
    - wired AND operation, 873
- Logic-to-Boolean expressions, 203
- Logic function generator, 393
- Logic gates, 132
  - AND gate, 133
  - NAND gate, 138
  - NOR gate, 140
  - NOT gate, 136
  - OR gate, 135
  - X-NOR gate, 143
  - X-OR gate, 142
- Look-ahead carry adder, 338
- Looping, 240
- Lower trigger level, 982
  
- Machine equivalence, 754
- Magnetic recording formats, 964
- Mainframe, 14
- Mantissa, 55
- Master-slave flip-flops, 571
  - D, 574
  - J-K, 574
  - S-R, 572
    - with data lock-out, 576
- Maxterm, 195, 232
- Mealy model, 747
- Memory
  - add on, 944
  - auxiliary, 944
  - bit organized, 954
  - boot strap, 948
  - charge coupled, 972
  - data, 944
  - expansion of, 953
  - FIFO, 961
    - data-rate buffer, 962
  - internal, 944
  - loop arrangement, 970
  - magnetic, 962
  - magnetic bubble, 970
  - magnetic core, 963
  - magnetic disk, 963
  - magnetic tape, 969
  - main, 944
  - mass, 969
  - optical disk, 971
    - OROM, 971
    - CD-ROM, 971
    - WORM, 971
  - peripheral, 944
  - program, 943
  - random access, 947
  - read only, 947
  - read/write, 947
  - sequential access, 947, 960
  - volatile, 947
  - word organized, 954
- Memory elements, 706
- Memory unit, 13
- Merger chart methods, 766
- Merger graph, 766
- Merger table, 768
- Microcomputer, 13
- Microprocessor, 11
- Minicomputer, 13
- Minimal cover table, 779
- Minimum distance of a code, 101

- Minterm, 194, 232  
Modular design using IC chips, 402  
Modulus arithmetic, 45  
Monolithic IC, 10  
Monostable multivibrator, 983  
    applications, 988  
Moore model, 749  
Moore reduction procedure, 754  
MOS resistor, 886  
Most significant digit, 28  
Multi gate synthesis, 536  
Multiple output minimization, 270  
Multiplexers, 390  
    2-input, 390  
    4-input, 391  
    8-input, 392  
    16-input, 392  
    applications, 393  
Multiplexing, 6, 390  
Multiplier, 6  
Multiplier/quotient register, 39
- NAND logic, 207  
Negative AND gate, 141  
Negative edge-triggered FF, 555  
Negative logic system, 3  
Negative OR gate, 139  
Negative zero, 47  
Nibble, 60  
Nine's and Ten's complements, 29  
NMOS  
    inverter, 887  
    NAND gate, 888  
    NOR gate, 888  
Noise, 2, 865  
Noise immunity, 864  
Noise margin, 864  
Non-degenerate forms, 282  
Non-volatile RAMs, 937  
NOR logic, 207
- Octal number system, 56  
Octet, 240  
One-bit comparator, 372
- One-shot, 983  
    non-retriggerable, 985  
    triggerable, 985  
Optical ROM, 1074  
Optional combinations, 261  
Output unit, 13
- Pages, 970  
Pair, 235  
PAL programming table, 471  
Paper method of multiplication, 38  
Parallel data transfer, 605  
Parallel subtractor, 337  
Parity, 99  
    bit, 100, 367  
    block, 101  
    even, 100  
    odd, 100  
    two-dimensional, 101  
Parity bit generator/checker, 367  
Parity bit generator for Hamming code, 368  
Parity word, 102  
Partition technique, 752  
Paull-Unger method, 768  
Perfect induction, 177  
Period, 5  
PLA programming table, 481  
POS form, 194  
Positional-weighted system, 28  
Positive edge-triggered FF, 555  
Positive logic system, 3  
Positive zero, 47  
Postulates of Boolean algebra, 178  
Potential well, 972  
Power dissipation in logic gates, 862  
Prime implicants, 249, 292  
    chart, 289  
    essential, 250  
Priority encoders, 379  
    decimal-to-BCD, 380  
    octal-to-binary, 382  
Program, 2  
Programmable array logic, 469  
Programmable counters, 664  
Programmable logic array, 476

Programmable logic devices (PLDs), 461

Programming, 461

PROM, 468

Propagation delay time, 862

Propositional, 5

Pseudo analog quantity, 912

Pull-down transistor, 869

Pull-up transistor, 869

Pulse, 4

Pulse mode circuits, 549

Pulse train generators, 669

using shift registers, 674

Pulse triggered FFs, 571

Pulsed operation, 148

Quad, 235

Quantization error, 927

Quartz crystal oscillator, 993

Quasi-stable state, 983

Quine-McClusky method, 287

Race around conditions, 569

Radix, 28

Reading, 614

Read/Write input, 945

Recirculating shift register, 960

Redundant false prime implicants, 251

Redundant prime implicants, 249

Register, 605

Retriggerable IC one-shot, 985

Ripple carry adder, 337

Rise time, 4

Rising edge, 4

ROM 462

applications of, 951

block diagram, 462

mask programmed, 467

organization of, 463

programming the, 467

types of, 467

Schmitt trigger, 982

Secondary essential prime implicants, 290

Secondary variables, 548

Selective false prime implicants, 251

Selective prime implicants, 250

Sequence detector, 712

Sequence generator, 669, 678

Sequential circuits, 547

asynchronous, 547

synchronous, 547, 699

Sequential machines, 699

Serial binary adder, 709

Serial data transfer, 584

Serial/parallel data conversion, 615

Serial parity-bit generator, 716

Seven-bit Hamming code, 108

Seven segment display, 388

Shannon's expansion theorem, 186

Shared minterm K-map, 270

Shift registers, 605

applications, 615

bidirectional, 611

data transmission in, 597

dynamic, 614

parallel-in, parallel-out, 611

parallel-in, serial-out, 610

serial-in, parallel-out, 610

serial-in, serial-out, 608

static, 614

universal, 612

Sign bit, 42

Sign magnitude form, 42

Single-shot, 983

Soft ware, 951

SOP form, 194

Speed power product, 865

S-R latch, 550

NAND gate, 552

NOR gate, 551

State, 632

Stable state, 983

Standard POS form, 195

Standard SOP form, 194

State assignment, 703

State box, 800

State compatibility, 765

State diagram, 701

State equivalence theorem, 752

- State machine, 800  
State reduction, 702  
State table, 701  
Static hazards, 407  
  static 0-hazard, 406  
  static 1-hazard, 406  
Static RAMs, 950  
Step size, 912  
Straight binary code, 88  
Strongly connected machine, 751  
Subcube, 249  
Subgraph, 779  
Substrate, 10  
Subtraction, 6  
Subtractor, 6  
Successor, 751  
Sum-of-weights method, 34  
Switching algebra, 176  
Switching circuits, 1  
Synthesis of threshold function, 533  
System design, 3
- Tabular method, 287  
Tautology, 515  
Terminal count, 633  
Terminal state, 751  
Threshold element, 512  
Threshold function, 528  
Threshold gate, 512  
Threshold voltage, 800  
Tie set hazard, 408  
Time delays, 615  
Time race, 568  
Timing diagrams, 148  
Trailing edge, 4  
Transition and output table, 703  
Transition table, 704  
Transition time, 1  
Transmission gate, 893  
Trigger, 561
- Triggering, 561  
  edge, 561  
  level, 561  
Triple precision, 55  
Truth table, 132  
Twelve bit hamming code, 108  
Two-bit comparator, 373  
Two-level implementation, 280  
Two-level logic, 207, 266  
Two-state devices, 31
- UART, 616  
Unasserted, 211  
Unate function, 531  
Unipolar ICs, 11  
Universal  
  building blocks, 137  
  gates, 137  
  logic, 207  
Universality of T gate, 525  
Upper trigger level, 982
- Variable mapping, 275  
Variable modulus counter, 633  
Venn diagram, 195
- Weighing machine, 824  
Weight of the term, 288  
Wired AND operation, 873  
Wired OR operation, 882  
Word, 944  
Word size, 944  
Wrapping around, 240  
Writing, 614
- XS-3  
  addition, 92  
  subtraction, 93

# Fundamentals of Digital Circuits

Fourth Edition

A. Anand Kumar

The Fourth edition of this well-received text continues to provide coherent and comprehensive coverage of digital circuits. It is designed for the undergraduate students pursuing courses in areas of engineering disciplines such as Electrical and Electronics, Electronics and Communication, Electronics and Instrumentation, Telecommunications, Medical Electronics, Computer Science and Engineering, Electronics, and Computers and Information Technology. It is also useful as a text for MCA, M.Sc. (Electronics) and M.Sc. (Computer Science) students. Appropriate for self study, the book is useful even for AMIE and grad IETE students.

Written in a student-friendly style, the book provides an excellent introduction to digital concepts and basic design techniques of digital circuits. It discusses Boolean algebra concepts and their application to digital circuitry, and elaborates on both combinational and sequential circuits. It provides numerous fully worked-out, laboratory tested examples to give students a solid grounding in the related design concepts. It includes a number of short questions with answers, review questions, fill in the blanks with answers, multiple choice questions with answers and exercise problems at the end of each chapter.

As the book requires only an elementary knowledge of electronics to understand most of the topics, it can also serve as a textbook for the students of polytechnics, B.Sc. (Electronics) and B.Sc. (Computer Science).

## NEW TO THIS EDITION

Now, based on the readers' demand, this new edition incorporates VERILOG programs in addition to VHDL programs at the end of each chapter.

## THE AUTHOR

A. ANAND KUMAR, Ph.D., is Principal, K.L. University College of Engineering, K.L. University, Green Fields, Vaddeswaram, Guntur District, Andhra Pradesh. Earlier, from 2006 to 2011 he served as Director, Sasi Institute of Technology and Engineering, Tadepalligudem, West Godavari District, Andhra Pradesh. From 2000 to 2006, he served as Principal of Sir C.R. Reddy College of Engineering, Eluru, West Godavari District, Andhra Pradesh and from 1983 to 2000 as Professor at S.D.M. College of Engineering and Technology, Dharwad, Karnataka. He has nearly four decades of teaching experience. His research interest includes radar signal processing.

## Other books by the same author

*Pulse and Digital Circuits*, 2nd ed.

*Digital Signal Processing*, 2nd ed.

*Control Systems*, 2nd ed.

*Signals and Systems*, 3rd ed.

*Switching Theory and Logic Design*, 3rd ed.

ISBN:978-81-203-5268-1

